

Chapter 9

Motion estimation

9.1	Translational alignment	558
9.1.1	Hierarchical motion estimation	562
9.1.2	Fourier-based alignment	563
9.1.3	Incremental refinement	566
9.2	Parametric motion	570
9.2.1	<i>Application:</i> Video stabilization	573
9.2.2	Spline-based motion	575
9.2.3	<i>Application:</i> Medical image registration	577
9.3	Optical flow	578
9.3.1	Deep learning approaches	584
9.3.2	<i>Application:</i> Rolling shutter wobble removal	587
9.3.3	Multi-frame motion estimation	587
9.3.4	<i>Application:</i> Video denoising	589
9.4	Layered motion	589
9.4.1	<i>Application:</i> Frame interpolation	593
9.4.2	Transparent layers and reflections	594
9.4.3	Video object segmentation	597
9.4.4	Video object tracking	598
9.5	Additional reading	600
9.6	Exercises	602

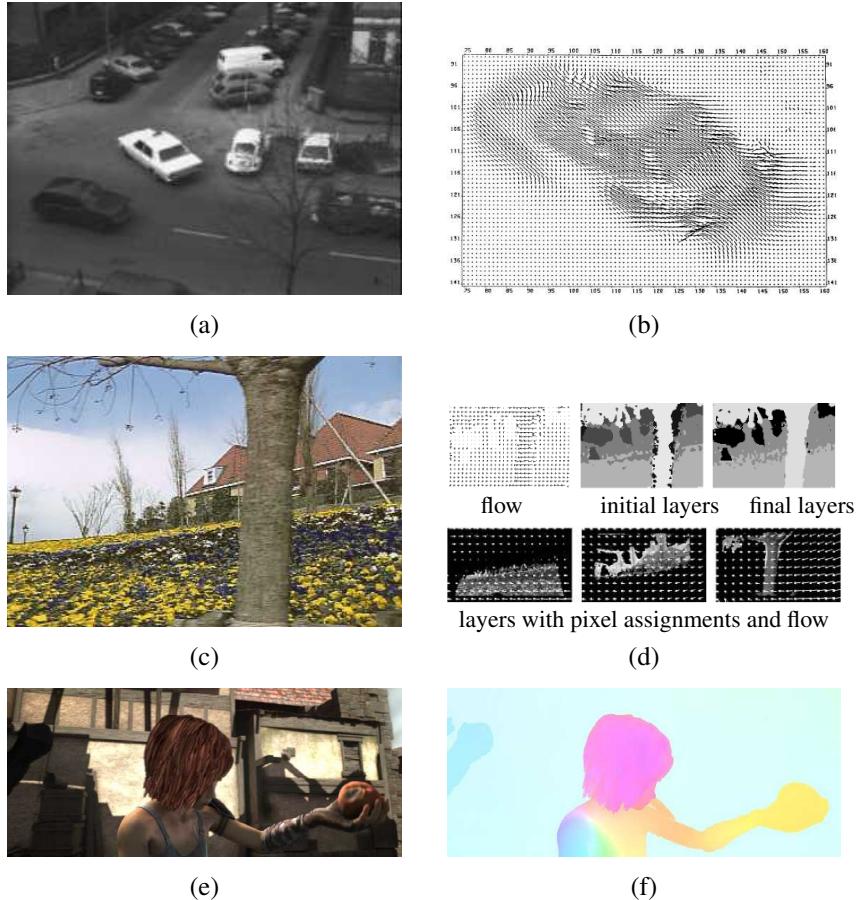


Figure 9.1 Motion estimation: (a–b) regularization-based optical flow (Nagel and Enkelmann 1986) © 1986 IEEE; (c–d) layered motion estimation (Wang and Adelson 1994) © 1994 IEEE; (e–f) sample image and ground truth flow from evaluation database (Butler, Wulff et al. 2012) © 2012 Springer.

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 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/H.263 (Le Gall 1991) and HEVC/H.265 (Sullivan, Ohm *et al.* 2012). 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 *et al.* 1994; Srinivasan, Chellappa *et al.* 2005; Matsushita, Ofek *et al.* 2006), and video compression (Irani, Hsu, and Anandan 1995; Lee, Chen *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 9.1). 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 9.1.1) based on image pyramids are normally used. Alternatively, Fourier transforms (Section 9.1.2) can be used to speed up the computation.

To get sub-pixel precision in the alignment, *incremental* methods (Section 9.1.3) based on a Taylor series expansion of the image function are often used. These can also be applied to *parametric motion models* (Section 9.2), 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 9.2). For more complex motions, piecewise parametric *spline motion models* (Section 9.2.2) 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, as described in Section 9.3. In recent years, the best-performing techniques have started using deep neural networks (Section 9.3.1). For even more complex motions that include a lot of occlusions, *layered motion models* (Section 9.4), which decompose the scene into coherently moving layers, can work well. Such representations can also be used to perform video object segmentation (Section 9.4.3) and object tracking (Section 9.4.4).

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, Scharstein *et al.* 2011; Sun, Yang *et al.* 2018; Janai, Güney *et al.* 2020; Hur and Roth 2020).

9.1 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 $I_0(\mathbf{x})$ sampled at discrete pixel locations $\{\mathbf{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\mathbf{x})$. A least squares solution to this problem is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{\text{SSD}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 = \sum_i e_i^2, \quad (9.1)$$

where $\mathbf{u} = (u, v)$ is the *displacement* and $e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$ 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 \mathbf{u} can be fractional, so a suitable interpolation function must be applied to image $I_1(\mathbf{x})$. 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 *et al.* 1986; Black and Anandan 1996; Stewart 1999) to obtain

$$E_{\text{SRD}}(\mathbf{u}) = \sum_i \rho(I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)) = \sum_i \rho(e_i). \quad (9.2)$$

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

¹The 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.

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

coding because of its speed, is the *sum of absolute differences* (SAD) metric³ or L_1 norm, i.e.,

$$E_{\text{SAD}}(\mathbf{u}) = \sum_i |I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)| = \sum_i |e_i|. \quad (9.3)$$

However, because this function is not differentiable at the origin, it is not well suited to gradient-descent approaches such as the ones presented in Section 9.1.3.

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,

$$\rho_{\text{GM}}(x) = \frac{x^2}{1 + x^2/a^2}, \quad (9.4)$$

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 *et al.* 1986; Rousseeuw and Leroy 1987), e.g., by computing the *median absolute deviation*, $MAD = \text{med}_i|e_i|$, and multiplying it by 1.4 to obtain a robust estimate of the standard deviation of the inlier noise process (Stewart 1999). Barron (2019) proposes a generalized robust loss function that can model various outlier distributions and thresholds, as discussed in more detail in Sections 4.1.3 and Appendix B.3, and also has a Bayesian method for estimating the loss function parameters.

Spatially varying weights. The error metrics above ignore the 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 with each of the two images being matched. The error metric then becomes the weighted (or *windowed*) SSD function,

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u})[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2, \quad (9.5)$$

³In video compression, e.g., the H.264 standard (<https://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, as it more accurately predicts quality (Richardson 2003).

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

$$A = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u}) \quad (9.6)$$

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

$$RMS = \sqrt{E_{\text{WSSD}}/A} \quad (9.7)$$

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,

$$I_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha)I_0(\mathbf{x}) + \beta, \quad (9.8)$$

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

$$E_{\text{BG}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha)I_0(\mathbf{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\mathbf{x}_i) + \beta - e_i]^2. \quad (9.9)$$

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 are also used in video codecs, where they are 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), and 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 *et al.* 1992), or gradients (Scharstein 1994; Papenberg, Bruhn *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,

$$E_{\text{CC}}(\mathbf{u}) = \sum_i I_0(\mathbf{x}_i) I_1(\mathbf{x}_i + \mathbf{u}). \quad (9.10)$$

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 $I_1(\mathbf{x})$, the maximum product may actually lie in that area.

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

$$E_{\text{NCC}}(\mathbf{u}) = \frac{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0] [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]}{\sqrt{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0]^2} \sqrt{\sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]^2}}, \quad (9.11)$$

where

$$\bar{I}_0 = \frac{1}{N} \sum_i I_0(\mathbf{x}_i) \quad \text{and} \quad (9.12)$$

$$\bar{I}_1 = \frac{1}{N} \sum_i I_1(\mathbf{x}_i + \mathbf{u}) \quad (9.13)$$

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 (9.9), is the *normalized SSD* score

$$E_{\text{NSSD}}(\mathbf{u}) = \frac{1}{2} \frac{\sum_i [(I_0(\mathbf{x}_i) - \bar{I}_0) - (I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1)]^2}{\sqrt{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0]^2 + [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]^2}} \quad (9.14)$$

proposed by Criminisi, Shotton *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).

9.1.1 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 *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

$$I_k^{(l)}(\mathbf{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\mathbf{x}_j) \quad (9.15)$$

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 $\mathbf{u}^{(l)}$ 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 $\mathbf{u}^{(l)} \in 2^{-l}[-S, S]^2$, where S is the desired *search range* at the finest (original) resolution level, optionally followed by the incremental refinement step described in Section 9.1.3.

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

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)} \quad (9.16)$$

⁴In stereo matching (Section 12.1.2), an explicit search over all possible disparities (i.e., a *plane sweep*) is almost always performed, as the number of search hypotheses is much smaller due to the 1D nature of the potential displacements.

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 $\hat{\mathbf{u}}^{(l-1)} \pm 1$, 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 9.2), is provided by Bergen, Anandan *et al.* (1992).

9.1.2 Fourier-based alignment

When the search range corresponds to a significant fraction of the larger image (as is the case in image stitching, see Section 8.2), the hierarchical approach may not work that well, as 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.,

$$\mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} = \mathcal{F}\{I_1(\mathbf{x})\} e^{-ju \cdot \boldsymbol{\omega}} = \mathcal{I}_1(\boldsymbol{\omega}) e^{-ju \cdot \boldsymbol{\omega}}, \quad (9.17)$$

where $\boldsymbol{\omega}$ is the vector-valued angular frequency of the Fourier transform and we use calligraphic notation $\mathcal{I}_1(\boldsymbol{\omega}) = \mathcal{F}\{I_1(\mathbf{x})\}$ 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 The Fourier transform of the cross-correlation function E_{CC} can thus be written as

$$\mathcal{F}\{E_{CC}(\mathbf{u})\} = \mathcal{F}\left\{\sum_i I_0(\mathbf{x}_i) I_1(\mathbf{x}_i + \mathbf{u})\right\} = \mathcal{F}\{I_0(\mathbf{u}) \bar{*} I_1(\mathbf{u})\} = \mathcal{I}_0(\boldsymbol{\omega}) \mathcal{I}_1^*(\boldsymbol{\omega}), \quad (9.18)$$

where

$$f(\mathbf{u}) \bar{*} g(\mathbf{u}) = \sum_i f(\mathbf{x}_i) g(\mathbf{x}_i + \mathbf{u}) \quad (9.19)$$

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).

⁵This doubling of displacements is only necessary if displacements are defined in integer *pixel* coordinates, which is the usual case in the literature (Bergen, Anandan *et al.* 1992). If *normalized device coordinates* (Section 2.1.4) 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 or less.

⁶In fact, the Fourier shift property (9.17) derives from the convolution theorem by observing that shifting is equivalent to convolution with a displaced delta function $\delta(\mathbf{x} - \mathbf{u})$.

To efficiently evaluate E_{CC} over the range of all possible values of \mathbf{u} , we take the Fourier transforms of both images $I_0(\mathbf{x})$ and $I_1(\mathbf{x})$, 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 $O(NM \log NM)$ operations (Bracewell 1986). This can be significantly faster than the $O(N^2 M^2)$ 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 (9.1). Its Fourier transform can be written as

$$\begin{aligned}\mathcal{F}\{E_{SSD}(\mathbf{u})\} &= \mathcal{F}\left\{\sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2\right\} \\ &= \delta(\omega) \sum_i [I_0^2(\mathbf{x}_i) + I_1^2(\mathbf{x}_i)] - 2\mathcal{I}_0(\omega)\mathcal{I}_1^*(\omega).\end{aligned}\quad (9.20)$$

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 (or patches).

Windowed correlation. Unfortunately, the Fourier convolution theorem only applies when the summation over \mathbf{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 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,

$$E_{WCC}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)I_0(\mathbf{x}_i) w_1(\mathbf{x}_i + \mathbf{u})I_1(\mathbf{x}_i + \mathbf{u}), \quad (9.21)$$

$$= [w_0(\mathbf{x})I_0(\mathbf{x})] \bar{*} [w_1(\mathbf{x})I_1(\mathbf{x})] \quad (9.22)$$

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 (9.5),

$$E_{WSSD}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u})[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2. \quad (9.23)$$

Expanding this as a sum of correlations and deriving the appropriate set of Fourier transforms is left for Exercise 9.1.

The same kind of derivation can also be applied to the bias–gain corrected sum of squared difference function E_{BG} (9.9). 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 9.1). It is also possible to use Fourier transforms to estimate the rotation and scale between two patches that are centered on the same pixel, as described in De Castro and Morandi (1987) and Szeliski (2010, Section 8.1.2).

Phase correlation. A variant of regular correlation (9.18) 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 (9.18) by the magnitudes of the Fourier transforms,

$$\mathcal{F}\{E_{\text{PC}}(\mathbf{u})\} = \frac{\mathcal{I}_0(\omega)\mathcal{I}_1^*(\omega)}{\|\mathcal{I}_0(\omega)\| \|\mathcal{I}_1(\omega)\|} \quad (9.24)$$

before taking the final inverse Fourier transform. In the case of noiseless signals with perfect (cyclic) shift, we have $I_1(\mathbf{x} + \mathbf{u}) = I_0(\mathbf{x})$ and hence, from Equation (9.17), we obtain

$$\begin{aligned} \mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} &= \mathcal{I}_1(\omega)e^{-2\pi j u \cdot \omega} = \mathcal{I}_0(\omega) \quad \text{and} \\ \mathcal{F}\{E_{\text{PC}}(\mathbf{u})\} &= e^{-2\pi j u \cdot \omega}. \end{aligned} \quad (9.25)$$

The output of phase correlation (under ideal conditions) is therefore a single spike (impulse) located at the correct value of \mathbf{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 9.1).

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.

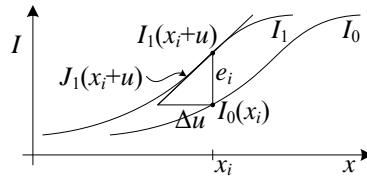


Figure 9.2 Taylor series approximation of a function and the incremental computation of the optical flow correction amount. $\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})$ is the image gradient at $(\mathbf{x}_i + \mathbf{u})$ and e_i is the current intensity difference.

9.1.3 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 (Szeliski and Scharstein 2004).

A more commonly used approach, first proposed by Lucas and Kanade (1981), is to perform *gradient descent* on the SSD energy function (9.1), using a Taylor series expansion of the image function (Figure 9.2),

$$E_{LK-SSD}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u} + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \quad (9.26)$$

$$\approx \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) + \mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} - I_0(\mathbf{x}_i)]^2 \quad (9.27)$$

$$= \sum_i [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2, \quad (9.28)$$

where

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) = \nabla I_1(\mathbf{x}_i + \mathbf{u}) = \left(\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y} \right) (\mathbf{x}_i + \mathbf{u}) \quad (9.29)$$

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

$$e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i), \quad (9.30)$$

first introduced in (9.1), is the current intensity error.⁷ The gradient at a particular sub-pixel location $(\mathbf{x}_i + \mathbf{u})$ can be computed using a variety of techniques, the simplest of which is

⁷We 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.

simply to take the horizontal and vertical differences between pixels \mathbf{x} and $\mathbf{x} + (1, 0)$ or $\mathbf{x} + (0, 1)$. More sophisticated derivatives can sometimes lead to noticeable performance improvements.

The linearized form of the incremental update to the SSD error (9.28) is often called the *optical flow constraint* or *brightness constancy constraint* equation (Horn and Schunck 1981)

$$I_x u + I_y v + I_t = 0, \quad (9.31)$$

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 optical flow (Horn and Schunck 1981).

The above least squares problem (9.28) can be minimized by solving the associated *normal equations* (Appendix A.2),

$$\mathbf{A}\Delta\mathbf{u} = \mathbf{b} \quad (9.32)$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) \mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \quad (9.33)$$

and

$$\mathbf{b} = - \sum_i e_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) \quad (9.34)$$

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

$$\mathbf{A} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}. \quad (9.35)$$

The gradients required for $\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\mathbf{x}_i + \mathbf{u})$ (Section 3.6.1 (3.75)) 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,

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \approx \mathbf{J}_0(\mathbf{x}_i), \quad (9.36)$$

because near the correct alignment, the template and displaced target images should look similar. This has the advantage of allowing the precomputation of the Hessian and Jacobian

⁸The true Hessian is the full second derivative of the error function E , which may not be positive definite—see Section 8.1.3 and Appendix A.3.

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 $I_1(\mathbf{x}_i + \mathbf{u})$ used to compute e_i in (9.30) 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 \mathbf{J}_1 using a least squares fit to a series of larger displacements 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 *et al.* 2008; Özysal, Calonder *et al.* 2010) as discussed in Section 7.1.5.

A commonly used stopping criterion for incremental updating is to monitor the magnitude of the displacement correction $\|\mathbf{u}\|$ and to stop when it drops below a certain threshold (say, $1/10$ 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 9.1.1.

Conditioning and aperture problems. Sometimes, the inversion of the linear system (9.32) 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 7.4). Only the *normal* component of the velocity (displacement) can be reliably recovered in this case. This manifests itself in (9.32) as a *rank-deficient* matrix \mathbf{A} , i.e., one whose smaller eigenvalue is very close to zero.⁹

When Equation (9.32) 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 \mathbf{A} , which essentially biases the solution towards smaller $\Delta\mathbf{u}$ 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

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

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. As discussed in Section 8.1.4 and Appendix B.6, under small amounts of additive Gaussian noise, it can be shown that the covariance matrix $\Sigma_{\mathbf{u}}$ is proportional to the inverse of the Hessian \mathbf{A} ,

$$\Sigma_{\mathbf{u}} = \sigma_n^2 \mathbf{A}^{-1}, \quad (9.37)$$

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 (9.28) 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 \mathbf{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 7.5 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 (9.9) to obtain

$$E_{LK-BG}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i - \alpha I_0(\mathbf{x}_i) - \beta]^2 \quad (9.38)$$

(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 $\Delta\mathbf{u}$ and the bias and gain parameters β and α .

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

$$I_1(\mathbf{x} + \mathbf{u}) \approx I_0(\mathbf{x}) + \sum_j \lambda_j B_j(\mathbf{x}), \quad (9.39)$$

where the $B_j(\mathbf{x})$ are the *basis images* and the λ_j are the unknown coefficients (Hager and Belhumeur 1998; Baker, Gross *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; Kambhamettu, Goldgof *et al.* 2003).

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

$$E_{\text{LK-WSSD}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u})[\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2. \quad (9.40)$$

Note that here, in deriving the Lucas–Kanade update from the original weighted SSD function (9.5), we have neglected taking the derivative of the $w_1(\mathbf{x}_i + \mathbf{u})$ weighting function with respect to \mathbf{u} , which is usually acceptable in practice, especially if the weighting function is a binary mask with relatively few transitions.

Baker, Gross *et al.* (2003) only use the $w_0(\mathbf{x})$ 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 $\nabla I(\mathbf{x})$, 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). Baker, Gross *et al.* (2003) show how evaluating Equation (9.40) 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 9.1,

$$E_{\text{LK-SRD}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i \rho(\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i), \quad (9.41)$$

which can be solved using the *iteratively reweighted least squares* technique described in Section 8.1.4.

9.2 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 2.1.1. As 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 *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 \mathbf{u} , we use a spatially varying *motion field* or *correspondence map*, $\mathbf{x}'(\mathbf{x}; \mathbf{p})$, parameterized by a low-dimensional vector \mathbf{p} , where \mathbf{x}' can be any of the motion models presented in Section 2.1.1. The parametric incremental motion update rule now becomes

$$E_{LK-PM}(\mathbf{p} + \Delta\mathbf{p}) = \sum_i [I_1(\mathbf{x}'(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p})) - I_0(\mathbf{x}_i)]^2 \quad (9.42)$$

$$\approx \sum_i [I_1(\mathbf{x}'_i) + \mathbf{J}_1(\mathbf{x}'_i)\Delta\mathbf{p} - I_0(\mathbf{x}_i)]^2 \quad (9.43)$$

$$= \sum_i [\mathbf{J}_1(\mathbf{x}'_i)\Delta\mathbf{p} + e_i]^2, \quad (9.44)$$

where the Jacobian is now

$$\mathbf{J}_1(\mathbf{x}'_i) = \frac{\partial I_1}{\partial \mathbf{p}} = \nabla I_1(\mathbf{x}'_i) \frac{\partial \mathbf{x}'}{\partial \mathbf{p}}(\mathbf{x}_i), \quad (9.45)$$

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

The motion Jacobians $\mathbf{J}_{x'}$ for the 2D planar transformations introduced in Section 2.1.1 and Table 2.1 are given in Table 8.1. Note how we have re-parameterized the motion matrices so that they are always the identity at the origin $\mathbf{p} = 0$. 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

$$\mathbf{A} = \sum_i \mathbf{J}_{x'}^T(\mathbf{x}_i) [\nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i)] \mathbf{J}_{x'}(\mathbf{x}_i) \quad (9.46)$$

and

$$\mathbf{b} = - \sum_i \mathbf{J}_{x'}^T(\mathbf{x}_i) [e_i \nabla I_1^T(\mathbf{x}'_i)]. \quad (9.47)$$

Note how the expressions inside the square brackets are the same ones evaluated for the simpler translational motion case (9.33–9.34).

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 \mathbf{A} and \mathbf{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),

$$\mathbf{A}_j = \sum_{i \in P_j} \nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i) \quad (9.48)$$

$$\mathbf{b}_j = \sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}'_i). \quad (9.49)$$

The full Hessian and residual can then be approximated as

$$\mathbf{A} \approx \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) [\sum_{i \in P_j} \nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i)] \mathbf{J}_{\mathbf{x}'}(\hat{\mathbf{x}}_j) = \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) \mathbf{A}_j \mathbf{J}_{\mathbf{x}'}(\hat{\mathbf{x}}_j) \quad (9.50)$$

and

$$\mathbf{b} \approx - \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) [\sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}'_i)] = - \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) \mathbf{b}_j, \quad (9.51)$$

where $\hat{\mathbf{x}}_j$ 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.

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 $\mathbf{x}'(\mathbf{x}; \mathbf{p})$,

$$\tilde{I}_1(\mathbf{x}) = I_1(\mathbf{x}'(\mathbf{x}; \mathbf{p})), \quad (9.52)$$

and then comparing this *warped* image against the template $I_0(\mathbf{x})$. Subsequently Hager and Belhumeur (1998) suggested replacing the gradient of $\tilde{I}_1(\mathbf{x})$ with the gradient of $I_0(\mathbf{x})$, as described previously in (9.36), which allows the precomputation (and inversion) of the Hessian matrix \mathbf{A} given in (9.46). The residual vector \mathbf{b} (9.47) can also be partially pre-computed, i.e., the *steepest descent* images $\nabla I_0(\mathbf{x}) \mathbf{J}_{\tilde{\mathbf{x}}}(\mathbf{x})$ can be precomputed and stored for later multiplication with the $e(\mathbf{x}) = \tilde{I}_1(\mathbf{x}) - I_0(\mathbf{x})$ error images, as described in (Szeliski 2010, Section 8.2) and (Baker and Matthews 2004), where this is called the *inverse additive* scheme. Baker and Matthews (2004) also introduce one more variant they call the *inverse compositional* algorithm where they warp the template image $I_0(\mathbf{x})$ and precompute the inverse Hessian and the steepest descent images, which makes it the preferred approach. They 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 *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 and have been widely used as components of subsequent object trackers, which are discussed in Section 9.4.4. Evangelidis and Psarakis (2008) provide some detailed experimental evaluations of these and other related approaches.

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 *et al.* 1997). First, a set of dense motion fields (Section 9.3) is computed from a set of training videos. Next, singular value decomposition (SVD) is applied to the stack of motion fields $\mathbf{u}_t(\mathbf{x})$ to compute the first few singular vectors $\mathbf{v}_k(\mathbf{x})$. 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

$$\mathbf{u}(\mathbf{x}) = \sum_k a_k \mathbf{v}_k(\mathbf{x}). \quad (9.53)$$

Figure 9.3a shows a set of basis fields learned by observing videos of walking motions. Figure 9.3b 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 7.1.5 and 13.6.4, as well as video stabilization (Yu and Ramamoorthi 2020).

9.2.1 Application: Video stabilization

Video stabilization is one of the most widely used applications of parametric motion estimation (Hansen, Anandan *et al.* 1994; Irani, Rousso, and Peleg 1997; Morimoto and Chellappa 1997; Srinivasan, Chellappa *et al.* 2005; Grundmann, Kwatra, and Essa 2011). 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.

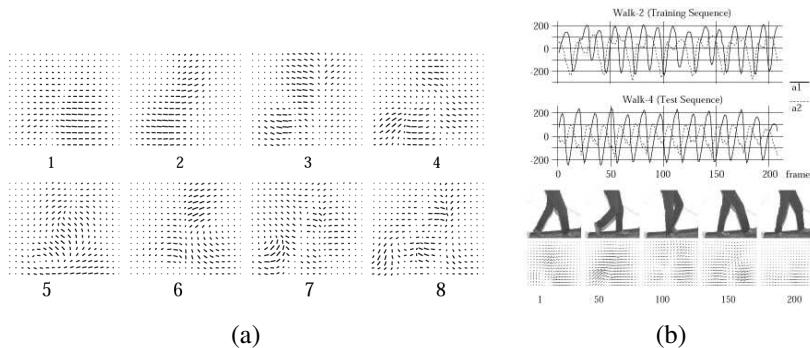


Figure 9.3 Learned parameterized motion fields for a walking sequence (Black, Yacoob et al. 1997) © 1997 IEEE: (a) learned basis flow fields; (b) plots of motion coefficients over time and corresponding estimated motion fields.

In their paper on full-frame video stabilization, Matsushita, Ofek *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 independently moving foreground objects (Yu and Ramamoorthi 2018, 2020; Yu, Ramamoorthi *et al.* 2021). 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. While quadratic penalties on motion derivatives are commonly used, more realistic virtual camera motions (locked and linear) can be obtained using L_1 minimization of derivatives (Grundmann, Kwatra, and Essa 2011). 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 by using deblurring techniques (Section 10.3) or by stealing sharper pixels from other frames with less motion or better focus (Matsushita, Ofek *et al.* 2006). Exercise 9.3 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. One or more smooth 3D camera paths 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 in what is sometimes called *content preserving warps* (Liu, Gleicher *et al.* 2009, 2011; Liu, Yuan *et al.* 2013; Kopf, Cohen, and Szeliski 2014). 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 14.3) (Smith, Zhang *et al.* 2009).

9.2.2 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 9.3) 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 (9.1) can thus be written as

$$E_{\text{SSD-OF}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2. \quad (9.54)$$

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

The two classic approaches to this problem, which we study in Section 9.3, are to perform the summation over overlapping regions (the *patch-based* or *window-based* approach) or to add smoothness terms on the $\{\mathbf{u}_i\}$ field using *regularization* or *Markov random fields* (Chapter 4). 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* controlled by a smaller number of *control vertices* $\{\hat{\mathbf{u}}_j\}$ (Figure 9.4),

$$\mathbf{u}_i = \sum_j \hat{\mathbf{u}}_j B_j(\mathbf{x}_i) = \sum_j \hat{\mathbf{u}}_j w_{i,j}, \quad (9.55)$$

where the $B_j(\mathbf{x}_i)$ are called the *basis functions* and are only non-zero over a small *finite support* interval (Szeliski and Coughlan 1997). We call the $w_{i,j} = B_j(\mathbf{x}_i)$ *weights* to emphasize that the $\{\mathbf{u}_i\}$ are known linear combinations of the $\{\hat{\mathbf{u}}_j\}$.

Substituting the formula for the individual per-pixel flow vectors \mathbf{u}_i (9.55) into the SSD error metric (9.54) yields a parametric motion formula similar to Equation (9.43). The biggest

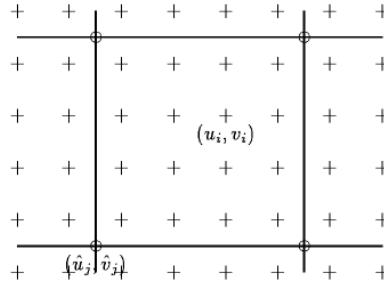


Figure 9.4 Spline motion field: the displacement vectors $\mathbf{u}_i = (u_i, v_i)$ are shown as pluses (+) and are controlled by the smaller number of control vertices $\hat{\mathbf{u}}_j = (\hat{u}_j, \hat{v}_j)$, which are shown as circles (\circ).

difference is that the Jacobian $\mathbf{J}_1(\mathbf{x}'_i)$ (9.45) now consists of the sparse entries in the weight matrix $\mathbf{W} = [w_{ij}]$.

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 2.1.4) can be naturally combined with a spline-based motion representation, where the in-plane motion is represented by a homography (8.19) 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 4.2). The simplest way to do this is to directly add squared difference penalties between adjacent vertices in the spline control mesh $\{\hat{\mathbf{u}}_j\}$, as in (4.24). 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; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013) can be used instead (Appendix A.5).

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 (Sec-

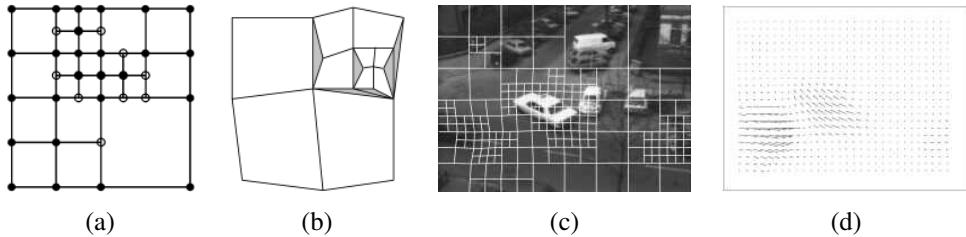


Figure 9.5 Quadtree spline-based motion estimation (Szeliski and Shum 1996) © 1996 IEEE: (a) quadtree spline representation, (b) which can lead to cracks, unless the white nodes are constrained to depend on their parents; (c) deformed quadtree spline mesh overlaid on grayscale image; (d) flow field visualized as a needle diagram.

tion 9.2.3) (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 are used. To remedy this situation, Szeliski and Shum (1996) propose using a *quadtree* representation embedded in the spline control grid (Figure 9.5a). Large cells are used to present regions of smooth motion, while smaller cells are added in regions of motion discontinuities (Figure 9.5c).

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 (9.54) is above a threshold, are subdivided into smaller patches. To avoid *cracks* in the resulting motion field (Figure 9.5b), 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).

9.2.3 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 de-

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

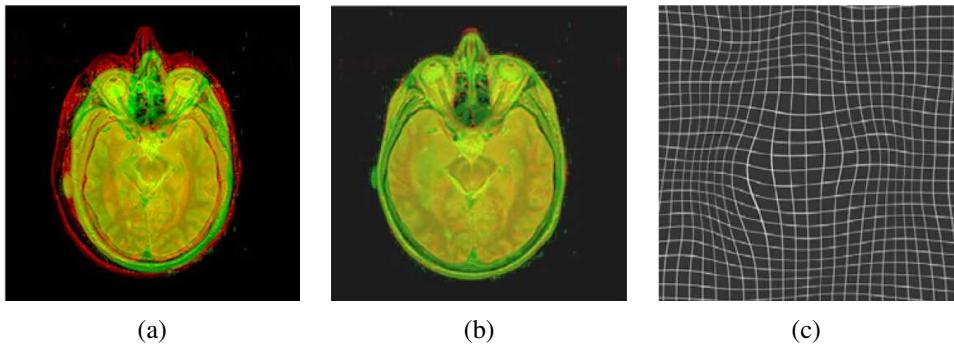


Figure 9.6 *Elastic brain registration (Kybic and Unser 2003) © 2003 IEEE:* (a) original brain atlas and patient MRI images overlaid in red–green; (b) after elastic registration with eight user-specified landmarks (not shown); (c) a cubic B-spline deformation field, shown as a deformed grid.

tect 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 *et al.* 1997).

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 9.6 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 *et al.* (2008), and the survey by Sotiras, Davatzikos, and Paragios (2013).

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 9.7).

9.3 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

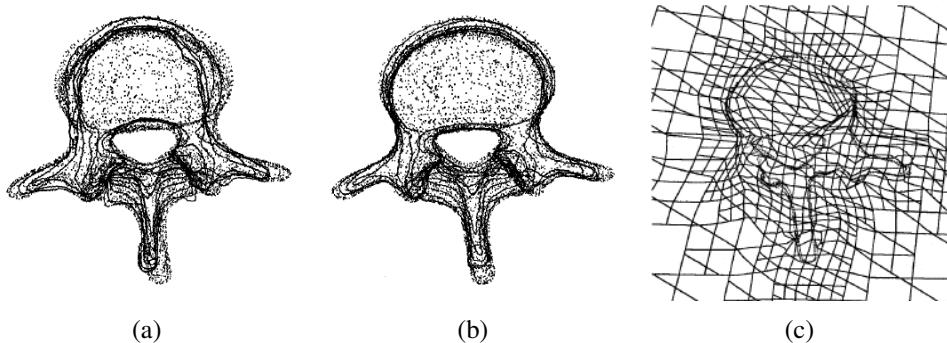


Figure 9.7 Octree spline-based image registration of two vertebral surface models (Szeliski and Lavallée 1996) © 1996 Springer: (a) after initial rigid alignment; (b) after elastic alignment; (c) a cross-section through the adapted octree spline deformation field.

we mentioned in the previous section, this generally involves minimizing the brightness or color difference between corresponding pixels summed over the image,

$$E_{\text{SSD-OF}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2. \quad (9.56)$$

Because the number of variables $\{\mathbf{u}_i\}$ 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 $\{\mathbf{u}_i\}$ field using regularization or Markov random fields (Chapter 4) and to search for a global minimum. Good overviews of recent optical flow algorithms can be found in Baker, Scharstein *et al.* (2011), Sun, Yang *et al.* (2018), Janai, Güney *et al.* (2020), and Hur and Roth (2020).

The patch-based approach usually involves using a Taylor series expansion of the displaced image function (9.28) 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 9.1.1. He also analyzes how the *uncertainty* in local motion estimates is related to the eigenvalues of the local Hessian matrix \mathbf{A}_i (9.37), as shown in Figures 7.4 and 7.5.

Bergen, Anandan *et al.* (1992) develop a unified framework for describing both parametric (Section 9.2) and patch-based optical flow algorithms and provide a nice introduction to this topic. After each iteration of optical flow estimation in a coarse-to-fine pyramid, they re-warp one of the images so that only incremental flow estimates are computed (Section 9.1.1).

When overlapping patches are used, an efficient implementation is to first compute the outer products of the gradients and intensity errors (9.33–9.34) 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 (9.56) is simultaneously minimized over all flow vectors $\{\mathbf{u}_i\}$. 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 (9.28), i.e., (9.31), is more commonly written as an analytic integral

$$E_{\text{HS}} = \int (I_x u + I_y v + I_t)^2 dx dy, \quad (9.57)$$

where $(I_x, I_y) = \nabla I_1 = \mathbf{J}_1$, $I_t = e_i$ is the *temporal derivative*, i.e., the brightness change between images, and $u(x, y)$ and $v(x, y)$ are the 2D optical flow functions. The Horn and Schunck model can also be viewed as the limiting case of spline-based motion estimation as the splines become 1×1 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 (9.28) to the analytic global energy (9.57),

$$E_{\text{HSD}} = \sum_i \mathbf{u}_i^T [\mathbf{J}_i \mathbf{J}_i^T] \mathbf{u}_i + 2e_i \mathbf{J}_i^T \mathbf{u}_i + e_i^2. \quad (9.58)$$

If we replace the per-pixel (rank 1) Hessians $\mathbf{A}_i = [\mathbf{J}_i \mathbf{J}_i^T]$ and residuals $\mathbf{b}_i = \mathbf{J}_i e_i$ with area-aggregated versions (9.33–9.34), we obtain a global minimization algorithm where region-based brightness constraints are used.

Another extension to the basic optical 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 *et al.* 1992; Szeliski and Coughlan 1997; Nir, Bruckstein, and Kimmel 2008; Wedel, Cremers *et al.* 2009). Such techniques are closely related to stereo matching (Chapter 12). 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

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

Jepson 1996; Chang, Tekalp, and Sezan 1997; Mémin and Pérez 2002). We revisit this topic in Section 9.4.

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 9.1, 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 *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 is 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 *et al.* 2006; Zach, Pock, and Bischof 2007b; Zimmer, Bruhn, and Weickert 2011). 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 *et al.* 2008; Werlberger, Trobin *et al.* 2009; Werlberger, Pock, and Bischof 2010). 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 *et al.* 2008). Many of these techniques are discussed in more detail by Baker, Scharstein *et al.* (2011) and Sun, Roth, and Black (2014).

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 were the method of choice until the advent of deep neural networks.¹³ One way to deal with this complexity is to start with efficient patch-based correspondences (Kroeger, Timofte *et al.* 2016). Another way to deal with the large two-dimensional search space is to integrate sparse feature matches into a variational formulation, as was initially proposed by Brox and Malik (2010a). This approach was later extended by several authors, including

¹²Robust brightness metrics (Section 9.1, (9.2)) can also help improve the performance of window-based approaches (Black and Anandan 1996).

¹³Some exceptions to this trend of not exploring the full 4D cost volume can be found in Xu, Ranftl, and Koltun (2017) and Teed and Deng (2020b).

		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 in0 in1 all disc untext			Mequon (Hidden texture) GT in0 in1 all disc untext			Schefflera (Hidden texture) GT in0 in1 all disc untext			Wooden (Hidden texture) GT in0 in1 all disc untext			Grove (Synthetic) GT in0 in1 all disc untext			Urban (Synthetic) GT in0 in1 all disc untext			Yosemite (Synthetic) GT in0 in1 all disc untext			Teddy (Stereo) GT in0 in1 all disc untext					
		4.4	0.09	0.26	0.06	0.23	0.78	0.54	0.10	0.21	0.18	0.91	0.10	0.08	1.25	0.73	0.50	1.28	0.31	0.14	0.16	0.22	0.63	1.37	0.79			
Adaptive [20]	4.4	0.11	0.28	0.10	0.18	0.63	0.12	0.11	0.75	0.18	0.19	0.97	0.12	0.07	1.31	1.00	1.78	1.73	0.87	0.11	0.12	0.22	0.68	1.48	0.95			
Complementary OF [21]	5.7	0.11	0.28	0.10	0.18	0.63	0.12	0.11	0.75	0.18	0.19	0.97	0.12	0.07	1.31	1.00	1.78	1.73	0.87	0.11	0.12	0.22	0.68	1.48	0.95			
Aniso. Huber-L1 [22]	5.8	0.10	0.28	0.08	0.31	0.85	0.28	0.11	0.75	0.29	0.20	0.92	0.13	0.04	1.20	0.70	0.39	1.23	0.28	0.17	0.15	0.27	0.64	1.36	0.74			
DPOF [18]	6.1	0.13	0.35	0.10	0.09	0.25	0.79	0.07	0.14	0.91	0.21	0.19	0.62	0.15	0.04	1.09	0.49	0.66	1.80	0.63	0.19	0.17	0.35	0.50	1.08	0.55		
TV-L1-improved [17]	7.2	0.09	0.26	0.07	0.20	0.73	0.16	0.53	1.18	0.22	0.21	1.24	0.11	0.04	1.31	0.73	1.51	1.93	0.84	0.18	0.17	0.31	0.73	1.62	0.87			
CBF [12]	7.8	0.10	0.28	0.06	0.34	0.98	0.06	0.37	0.26	0.21	1.14	0.13	0.04	1.27	0.70	0.41	1.23	0.30	0.23	0.16	0.39	0.76	1.56	1.02				
Brox et al. [5]	8.4	0.11	0.32	0.11	0.27	0.93	0.22	0.39	0.94	0.24	0.24	1.25	0.13	0.10	1.39	1.47	0.89	1.77	0.55	0.10	0.13	0.11	0.91	1.83	1.13			
Rannacher [23]	8.5	0.11	0.31	0.09	0.25	0.84	0.21	0.57	1.27	0.26	0.24	1.32	0.13	0.01	1.33	0.72	1.49	1.95	0.78	0.15	0.14	0.26	0.59	1.58	0.86			
F-TV-L1 [15]	8.8	0.14	0.35	0.14	0.34	0.98	0.26	0.59	1.19	0.26	0.27	1.36	0.16	0.04	1.30	0.76	0.54	1.62	0.36	0.13	0.15	0.20	0.62	1.56	0.66			
Second-order prior [8]	9.0	0.11	0.31	0.09	0.26	0.93	0.20	0.57	1.25	0.26	0.24	1.04	0.12	0.04	1.34	0.83	0.61	1.93	0.47	0.20	0.16	0.34	0.77	1.64	0.77			
Fusion [6]	9.4	0.11	0.34	0.10	0.19	0.89	0.16	0.29	0.86	0.23	0.20	1.19	0.14	0.07	1.42	1.22	1.35	1.49	0.86	0.20	0.19	0.20	0.26	1.07	0.47	2.07	1.39	
Dynamic MRF [7]	11.1	0.12	0.34	0.11	0.22	0.24	0.89	0.16	0.44	1.13	0.20	0.24	1.29	0.14	0.11	1.52	1.13	1.54	2.37	0.93	0.13	0.12	0.31	1.27	1.8	2.33	0.16	
SegOf [10]	11.7	0.15	0.36	0.10	0.57	1.16	0.59	0.68	1.18	1.24	0.64	0.32	0.86	0.26	0.18	1.15	1.47	1.63	2.09	0.96	0.08	0.13	0.12	0.70	1.50	0.69		
Learning Flow [11]	13.3	0.11	0.32	0.09	0.29	0.99	0.23	0.55	1.24	0.29	0.36	1.56	0.25	0.15	1.64	1.41	1.55	2.32	0.85	0.14	0.16	0.24	0.09	1.29	0.17			
Filter Flow [19]	14.3	0.17	0.39	0.13	0.43	1.01	0.09	0.38	0.75	1.34	0.78	0.79	1.54	0.68	0.13	1.38	1.51	0.57	1.32	0.44	0.22	0.23	0.23	0.26	0.95	1.66	1.12	
GraphCuts [14]	14.5	0.16	0.38	0.14	0.59	1.16	0.46	0.56	1.07	0.64	0.26	1.14	0.17	0.06	1.35	0.84	0.25	1.79	1.22	0.22	0.17	0.43	0.22	1.7	2.05	1.78		
Block & Anandan [4]	15.0	0.18	0.42	0.17	0.58	1.17	0.50	0.95	1.58	0.70	0.49	1.59	0.45	0.08	1.42	1.22	1.43	2.28	0.83	0.15	0.17	0.17	1.11	1.98	1.30			
SPSA-learn [13]	15.7	0.18	0.45	0.17	0.57	1.32	0.51	0.84	1.50	0.72	0.52	1.64	0.49	1.12	1.42	1.39	1.75	2.14	1.06	0.13	0.13	0.19	1.32	2.08	1.17			
GroupFlow [9]	15.9	0.21	0.51	0.21	0.79	1.69	0.72	0.86	1.64	0.74	0.39	1.07	0.26	1.29	1.81	0.82	1.94	2.30	1.36	0.11	0.14	0.19	1.06	1.96	1.35			
2D-CLG [1]	17.4	0.28	0.62	0.21	0.67	1.21	0.16	0.70	1.12	21	0.80	0.99	0.17	0.26	1.12	22	1.23	1.52	1.62	1.54	2.15	0.96	0.19	0.41	0.16	1.38	2.26	1.83
Horn & Schunck [3]	18.6	0.22	0.55	0.22	0.61	1.53	0.52	1.01	1.73	0.80	0.78	2.02	0.77	1.26	2.0	1.58	1.00	1.43	2.59	1.05	0.16	0.18	0.15	1.51	2.50	1.88		
TI-DOFE [24]	19.6	0.38	0.64	0.47	1.16	1.72	1.26	1.39	2.05	1.73	1.29	2.21	1.41	1.27	1.61	1.57	1.28	2.57	1.01	0.13	0.15	0.16	1.87	2.7	2.11	2.53		
FOLKI [16]	22.6	0.29	0.73	0.33	1.52	1.96	1.80	1.23	2.0	0.95	0.98	2.20	1.08	1.53	1.85	2.07	2.14	2.32	1.60	0.26	0.21	0.68	0.27	3.27	4.32			
Pyramid LK [2]	23.7	0.39	0.61	0.61	1.67	2.18	2.00	1.50	1.97	1.38	1.57	2.39	1.78	2.94	3.72	2.43	3.33	2.74	2.43	0.39	0.24	0.24	0.73	3.86	2.0	5.08	4.88	

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



Figure 9.8 Evaluation of the results of 24 optical flow algorithms, October 2009, <https://vision.middlebury.edu/flow>, (Baker, Scharstein 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.

Weinzaepfel, Revaud *et al.* (2013), whose DeepFlow system use a hand-crafted (non-learnt) convolutional network to compute initial quasi-dense correspondences, and Revaud, Weinzaepfel *et al.* (2015), whose EpicFlow system added an edge and occlusion-aware interpolation step before the variational optimization.

Combinatorial optimization methods based on Markov random fields were among the better-performing methods on the optical flow database of Baker, Scharstein *et al.* (2011)¹⁴ when it was originally released, but have now been overtaken by deep neural networks. Examples of such techniques include the one developed by Glocker, Paragios *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. 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.

A careful empirical analysis of these kinds of “classic” coarse-to-fine energy-minimization approaches is provided in the meticulously executed paper by Sun, Roth, and Black (2014).¹⁵ Figure 9.9a shows the main components of the framework they examine, including an initial warping based on the previous level’s flow (or a grid search at the coarsest level), followed by energy minimizing flow updates, and then an optional post-processing step. In their paper, the authors not only review dozens of variational (energy-minimization) approaches developed from the 1980s (Horn and Schunck 1981) through to 2013, but also show that algorithmic details such as median filtering post-processing, often glossed over by previous authors, have a strong influence on the results. In addition to performing their analysis on the Middlebury Flow dataset (Baker, Scharstein *et al.* 2011), they also evaluate on the newer Sintel dataset (Butler, Wulff *et al.* 2012).¹⁶

The field of accurate motion estimation continues to evolve at a rapid pace, with significant advances in performance occurring every year. While the Middlebury optical flow website (Figure 9.8) continues to be a good source of pointers to high-performing algorithms, more recent publications tend to focus (both training and evaluation) on the MPI Sintel dataset developed by Butler, Wulff *et al.* (2012), some samples of which are shown in Figure 9.1e–f. Some algorithms also train and test on the KITTI flow benchmark (Geiger, Lenz, and Urtasun 2012), although that dataset focuses on video acquired from a driving vehicle. In general, it appears that learning-based algorithms trained on one dataset still have trouble when applied to a different dataset.¹⁷

¹⁴<https://vision.middlebury.edu/flow>

¹⁵The earlier conference version of this paper had the eye-catching title of “Secrets of optical flow estimation and their principles” (Sun, Roth, and Black 2010).

¹⁶<http://sintel.is.tue.mpg.de>

¹⁷<http://www.robustvision.net>

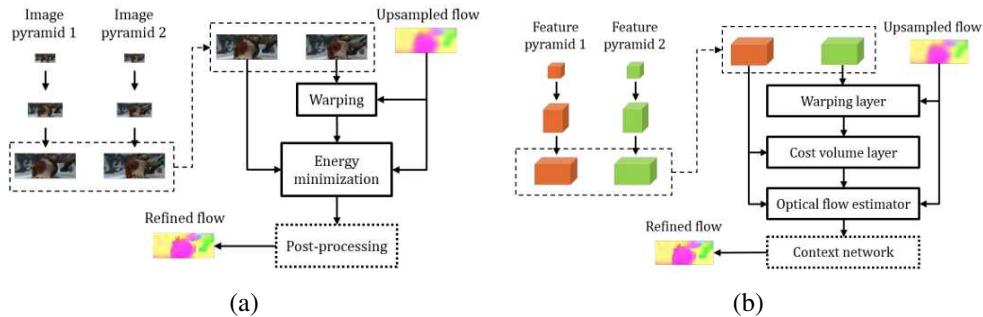


Figure 9.9 Iterative coarse-to-fine optical flow estimation (Sun, Yang et al. 2018) © 2018 IEEE: (a) “classic” variational (energy minimization) approach (Sun, Roth, and Black 2014); (b) newer neural network approach trained with end-to-end deep learning (Sun, Yang et al. 2018). Both figures show the processing at a single level of the coarse-to-fine pyramid, taking as input the flow computed by the previous (coarser) level and passing the refined flow onto the finer level below.

9.3.1 Deep learning approaches

Over the last decade, deep neural networks have become an essential component of all highly-performant optical flow algorithms, as described in the survey articles by Janai, Güney *et al.* (2020, Chapter 11) and Hur and Roth (2020). An early approach to use non-linear aggregation inspired by deep convolutional networks is the DeepFlow system of Weinzaepfel, Revaud *et al.* (2013), which uses a hand-crafted (non-learned) convolutions and pooling to compute multi-level response maps (matching costs), which are then optimized using a classic energy-minimizing variational framework.

The first system to use full deep end-to-end learning in an encoder-decoder network was FlowNetS (Dosovitskiy, Fischer *et al.* 2015), which was trained on the authors’ synthetic FlyingChairs dataset. The paper also introduced FlowNetC, which uses a correlation network (local cost volume). The follow-on FlowNet 2.0 system uses the initial flow estimates to warp the images and then refines the flow estimates using cascaded encoder-decoder networks (Ilg, Mayer *et al.* 2017), while subsequent papers also deal with occlusions and uncertainty modeling (Ilg, Saikia *et al.* 2018; Ilg, Çiçek *et al.* 2018).

An alternative to stacking full-resolution networks in series is to use image and flow pyramids together with coarse-to-fine warping and refinement, as first explored in the SPyNet paper by Ranjan and Black (2017). The more recent PWC-Net of Sun, Yang *et al.* (2018, 2019) shown in Figure 9.9b extends this idea by first computing a feature pyramid from each frame, warping the second set of features by the flow interpolated from the previous resolution

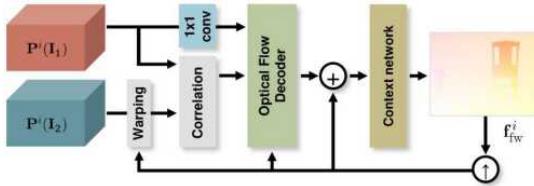


Figure 9.10 Iterative residual refinement optical flow estimation (Hur and Roth 2019) © 2019 IEEE. The coarse-to-fine cascade of Sun, Yang et al. (2018) in Figure 9.9b is replaced with a recurrent neural network (RNN) that cycles interpolated coarser level flow estimates as warping inputs to the next finer level but uses the same convolutional weights at each level.

level, and then computing a cost volume by correlating these features using a dot product between feature maps shifted by up to $d = \pm 4$ pixels. The refined optical flow estimates at the current level are produced using a multi-layer CNN whose inputs are the cost volume, the image features, and the interpolated flow from the previous level. A final context network takes as input the flow estimate and features from the second to last level and uses dilated convolutions to endow the network with a broader context. If you compare Figures 9.9a–b, you will see a pleasing correspondence between the various processing stages of classic and deep coarse-to-fine flow estimation algorithms.¹⁸

A variant on the coarse-to-fine PWC-Net developed by Hur and Roth (2019) is the Iterative Residual Refinement network shown in Figure 9.10. Instead of cascading a set of different deep networks as in FlowNet 2.0 and PWC-Net, IRRs re-use the same structure and convolution weights at each layer, which allows the network to be re-drawn in the “rolled up” version, as shown in this figure. The network can thus be thought of as a simple recurrent neural network (RNN) that upsamples the output flow estimates after each stage. In addition to having fewer parameters, this weight sharing also improves accuracy. In their paper, the authors also show how this network can be extended (doubled) to simultaneously compute forward and backward flows as well as occlusions.

In more recent work, Jonschkowski, Stone *et al.* (2020) take PWC-Net as their basic architecture and systematically study all of the components involved in training the flow estimator in an *unsupervised* manner, i.e., using regular real-world videos with no ground truth flow, which can enable much larger training sets to be used (Ahmadi and Patras 2016; Meister, Hur, and Roth 2018). In their paper, Jonschkowski *et al.* systematically compare photometric losses, occlusion estimation, self-supervision, and smoothness constraints, and analyze the

¹⁸Note that as with other coarse-to-fine warping approaches, these algorithms struggle with fast-moving fine structures that may not be visible at coarser levels (Brox and Malik 2010a).

effect of other choices, such as pre-training, image resolution, data augmentation, and batch size. They also propose four improvements to these key components, including cost volume normalization, gradient stopping for occlusion estimation, applying smoothness at the native flow resolution, and image resizing for self-supervision. Another recent paper that explicitly deals with occlusions is [Jiang, Campbell *et al.* \(2021\)](#).

Another recent trend has been to model the uncertainty that arises in flow field estimation due to homogeneous and occluded regions ([Ilg, Çiçek *et al.* 2018](#)). The HD³ network developed by [Yin, Darrell, and Yu \(2019\)](#) models correspondence distributions across multiple resolution levels, while the LiteFlowNet3 network of [Hui and Loy \(2020\)](#) extends their small and fast LiteFlowNet2 network ([Hui, Tang, and Loy 2021](#)) with cost volume modulation and flow field deformation modules to significantly improve accuracy at minimal cost. In concurrent work, [Hofinger, Rota Bulò *et al.* \(2020\)](#) introduce novel components such as replacing warping by sampling, smart gradient blocking, and knowledge distillation, which not only improve the quality of their flow estimates but can also be used in other applications such as stereo matching. [Teed and Deng \(2020b\)](#) build on the idea of a recurrent network ([Hur and Roth 2019](#)), but instead of warping feature maps, they precompute a full $(W \times H)^2$ multi-resolution correlation volume (Recurrent All-Pairs Field Transforms or RAFT), which is accessed at each iteration based on the current flow estimates. Computing a sparse correlation volume storing only the k closest matches for each reference image feature can further accelerate the computation ([Jiang, Lu *et al.* 2021](#)).

Given the rapid evolution in optical flow techniques, which is the best one to use? The answer is highly problem-dependent. One way to assess this is to look across a number of datasets, as is done in the Robust Vision Challenge.¹⁹ On this aggregated benchmark, variants of RAFT, IRR, and PWC all perform well. Another is to specifically evaluate a flow algorithm based on its intended use, and, if possible, to fine-tune the network on problem-specific data. [Xue, Chen *et al.* \(2019\)](#) describe how they fine-tune a SPyNet coarse-to-fine network on their synthetically degraded Vimeo-90K dataset to estimate *task-oriented flow* (TOFlow), which outperforms “higher accuracy” networks (and even ground truth flow) on three different video processing tasks, namely frame interpolation (Section 9.4.1), video denoising (Section 9.3.4), and video super-resolution. It is also possible to significantly improve the performance of learning-based flow algorithms by tailoring the synthetic training data to a target dataset ([Sun, Vlasic *et al.* 2021](#)).

¹⁹<http://www.robustvision.net/leaderboard.php?benchmark=flow>

9.3.2 Application: Rolling shutter wobble removal

To save on silicon circuitry and enable greater photo sensitivity or fill factors, many CMOS imaging sensors such as those found in mobile phones use a *rolling shutter*, where different rows or columns are exposed in sequence. When photographing or filming a scene with fast scene or camera motions, this can result in straight lines becoming slanted or curved (e.g., the propeller blades on a plane or helicopter) or rigid parts of the scene wobbling (also known as the *jello effect*), e.g., when the camera is rapidly vibrating during action photography.

To compensate for these distortion, which are caused by different exposure times for different scanlines, accurate per-pixel optical flow must be estimated, as opposed to the whole-frame parametric motion that can sometimes be used for slower-motion video stabilization (Section 9.2.1). Baker, Bennett *et al.* (2010) and Forssén and Ringaby (2010) were among the first computer vision researchers to study this problem. In their paper, Baker, Bennett *et al.* (2010) recover a high-frequency motion field from the lower-frequency inter-frame motions and use this to resample each output scanline. Forssén and Ringaby (2010) perform similar computations using models of camera rotation, which require intrinsic lens calibration. Grundmann, Kwatra *et al.* (2012) remove the need for such calibration using mixtures of homographies to model the camera and scene motions, while Liu, Gleicher *et al.* (2011) use subspace constraints. Accurate rolling shutter correction is also required to produce high-quality image stitching results (Zhuang and Tran 2020).

While in some modern imaging systems such as action cameras, inertial measurements units (IMUs) can provide high-frequency estimates of camera motion, they cannot directly provide estimates of depth-dependent parallax and independent object motions. For this reason, the best in-camera image stabilizers use a combination of IMU data and sophisticated image processing.²⁰ Modeling rolling shutter is also important to obtain accurate pose estimates in structure from motion (Hedborg, Forssén *et al.* 2012; Kukelova, Albl *et al.* 2018; Albl, Kukelova *et al.* 2020; Kukelova, Albl *et al.* 2020) and visual-inertial fusion in SLAM (Patron-Perez, Lovegrove, and Sibley 2015; Schubert, Demmel *et al.* 2018), which are discussed in Sections 11.4.2 and 11.5.

9.3.3 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.

²⁰<https://gopro.com/en/us/news/hero7-black-hypersmooth-technology>

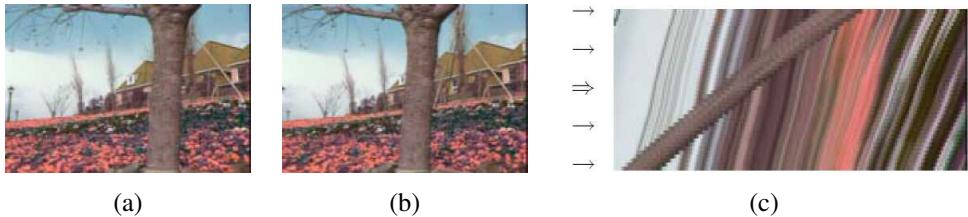


Figure 9.11 *Slice through a spatio-temporal volume (Szeliski 1999a) © 1999 IEEE: (a–b) two frames from the flower garden sequence; (c) a horizontal slice through the complete spatio-temporal volume, with the arrows indicating locations of potential key frames where flow is estimated. Note that the colors for the flower garden sequence are incorrect; the correct colors (yellow flowers) are shown in Figure 9.13.*

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 9.11 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, 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 1999a). Figure 9.11c shows schematically one potential approach to this problem. The horizontal arrows show the locations of 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 12.7), where the unknowns at each pixel are disparities and occlusion relationships can be determined directly from pixel depths (Szeliski 1999a; Kolmogorov and Zabih 2002). However, it is also applicable to general motion, with the addition of models for occlusion relationships, as in the MirrorFlow system of Hur and Roth (2017) as well as multi-frame versions (Janai, Guney *et al.* 2018; Neoral, Šochman, and Matas 2018; Ren, Gallo *et al.* 2019).

9.3.4 Application: Video denoising

Video denoising is the process of removing noise and other artifacts such as scratches from film and video (Kokaram 2004; Gai and Kang 2009; Liu and Freeman 2010). 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, to do this without introducing blur or jitter (irregular motion), they need accurate per-pixel motion estimates. One way to do this is to use task-oriented flow, where the flow network is specifically tuned end-to-end to provide the best denoising performance (Xue, Chen *et al.* 2019).

Exercise 9.6 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. And while some recent papers continue to estimate flow as part of the multi-frame denoising pipeline (Tassano, Delon, and Veit 2019; Xue, Chen *et al.* 2019), others either concatenate similar patches from different frames (Maggioni, Boracchi *et al.* 2012) or concatenate small subsets of frames into a deep network that never explicitly estimates a motion representation (Claus and van Gemert 2019; Tassano, Delon, and Veit 2020). A more general form of video enhancement and restoration called *video quality mapping* has also recently started being investigated (Fuoli, Huang *et al.* 2020).

9.4 Layered motion

In many situations, 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 9.12 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*

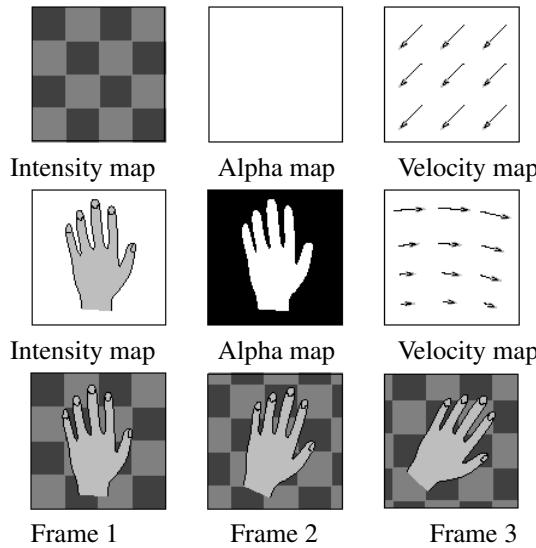


Figure 9.12 *Layered motion estimation framework (Wang and Adelson 1994) © 1994 IEEE: The top two rows describe the two layers, each of which consists of an intensity (color) image, an alpha mask (black=transparent), and a parametric motion field. The layers are composited with different amounts of motion to recreate the video sequence.*

objects) and the parametric motion corresponding to each layer. Displacing and compositing 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, Chen *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 14.2.1) (Shade, Gortler *et al.* 1998; Zitnick, Kang *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,

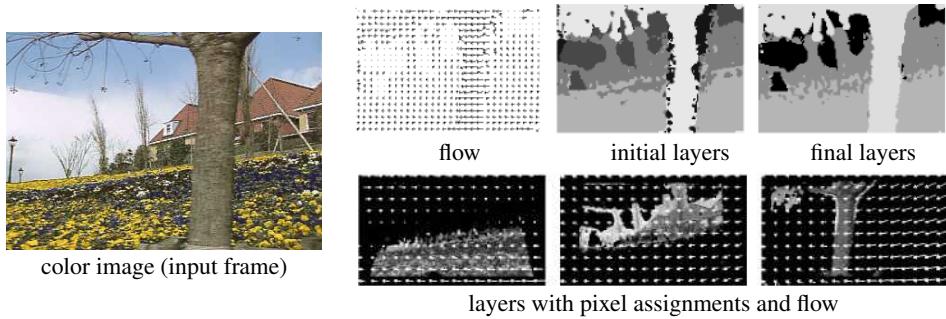


Figure 9.13 Layered motion estimation results (Wang and Adelson 1994) © 1994 IEEE.

as well as to infer occlusion relationships between the layers. Figure 9.13 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 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 2.1.4)). The initial layer estimation proceeds in a manner similar to that of Wang and Adelson (1994), 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 , \mathbf{n}_l , and \mathbf{P}_t by minimizing the discrepancy between the re-synthesized and observed motion sequences (Baker, Szeliski, and Anandan 1998).

Figure 9.14 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 9.13. 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,

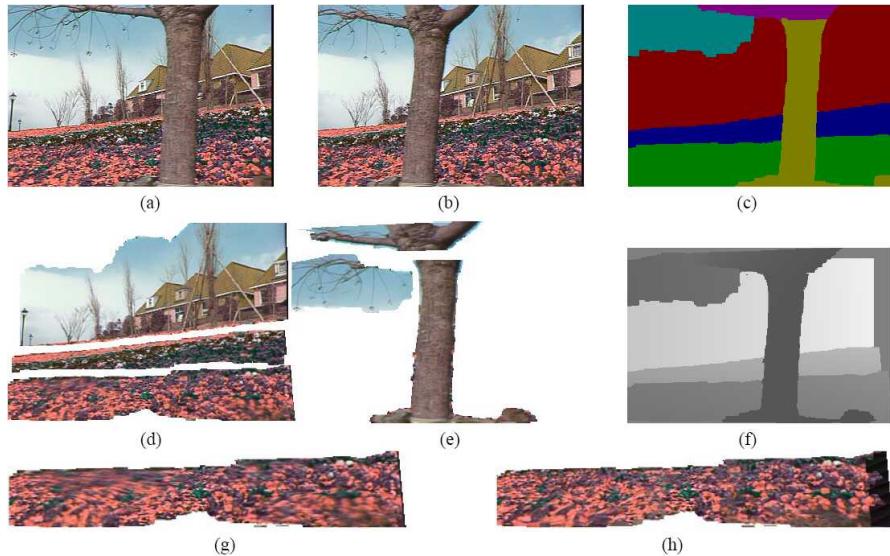


Figure 9.14 Layered stereo reconstruction (Baker, Szeliski, and Anandan 1998) © 1998 IEEE: (a) first and (b) last input images; (c) initial segmentation into six layers; (d) and (e) the six layer sprites; (f) depth map for planar sprites (darker denotes closer); front layer (g) before and (h) after residual depth estimation. Note that the colors for the flower garden sequence are incorrect; the correct colors (yellow flowers) are shown in Figure 9.13.

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 from the 2000s include (Sawhney and Ayer 1996; Jovic and Frey 2001; Xiao and Shah 2005; Kumar, Torr, and Zisserman 2008; Thayananthan, Iwasaki, and Cipolla 2008; Schoenemann and Cremers 2008), while more recent papers include (Sun, Suderth, and Black 2012; Sun, Wulff *et al.* 2013; Sun, Liu, and Pfister 2014; Wulff and Black 2015) and (Sevilla-Lara, Sun *et al.* 2016), which jointly performs semantic segmentation and motion estimation.

Layers are not the only way to introduce segmentation into motion estimation. A large number of algorithms have been developed that alternate between estimating optical 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 these 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 *et al.* 2004; Zitnick, Jovic, and Kang 2005; Stein, Hoiem, and Hebert 2007; Thayanathan, Iwasaki, and Cipolla 2008). In fact, the segmentation of videos into coherently moving parts has evolved into its own topic, namely *video object segmentation*, which we study in Section 9.4.3.

9.4.1 Application: Frame interpolation

Frame interpolation is a widely used application of motion estimation, often implemented in hardware to match an incoming video to a monitor’s actual refresh rate, where information in 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 9.11c 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 \mathbf{u}_0 is obtained at location \mathbf{x}_0 in image I_0 as is obtained at location $\mathbf{x}_0 + \mathbf{u}_0$ in image I_1 , the flow vectors are said to be *consistent*. This motion estimate can be transferred to location $\mathbf{x}_0 + t\mathbf{u}_0$ in the image I_t being generated, where $t \in (0, 1)$ is the time of interpolation. The final color value at pixel $\mathbf{x}_0 + t\mathbf{u}_0$ can be computed as a linear blend,

$$I_t(\mathbf{x}_0 + t\mathbf{u}_0) = (1 - t)I_0(\mathbf{x}_0) + tI_1(\mathbf{x}_0 + \mathbf{u}_0). \quad (9.59)$$

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 by Shade, Gortler *et al.* (1998) and Zitnick, Kang *et al.* (2004), is the one used in the flow evaluation paper of Baker, Scharstein *et al.* (2011). An even higher-quality frame interpolation algorithm, which uses gradient-based reconstruction, is presented by Mahajan, Huang *et al.* (2009). Accuracy on frame interpolation tasks is also sometimes used to gauge the quality of motion estimation algorithms (Szeliski 1999b; Baker, Scharstein *et al.* 2011).

More recent frame interpolation techniques use deep neural networks as part of their architectures. Some approaches use spatio-temporal convolutions (Niklaus, Mai, and Liu 2017), while others use DNNs to compute bi-directional optical flow (Xue, Chen *et al.* 2019)

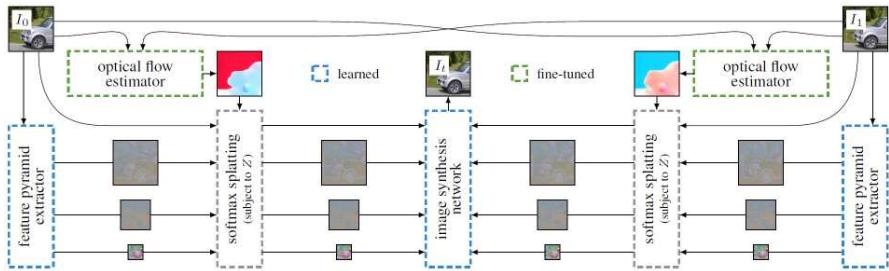


Figure 9.15 Deep feature video interpolation network (Niklaus and Liu 2020) © 2020 IEEE. This multi-stage network first computes bi-directional flow, encodes each frame using feature pyramids, and then warps and combines these features using softmax splatting. The combined features are then fed into a final image synthesis network (decoder).

and then combine the contributions from the two original frames using either context features (Niklaus and Liu 2018) or soft visibility maps (Jiang, Sun *et al.* 2018). The system by Niklaus and Liu (2020) encodes the input frames as deep multi-resolution neural features, forward warps these using bi-directional flow, combines these features using softmax splatting, and then uses a final deep network to decode these combined features, as shown in Figure 9.15. A similar architecture can also be used to create temporally textured looping videos from a single still image (Holynski, Curless *et al.* 2021). Other recently developed frame interpolation networks include Choi, Choi *et al.* (2020), Lee, Kim *et al.* (2020), Kang, Jo *et al.* (2020), and Park, Ko *et al.* (2020).

9.4.2 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 9.16 and 9.17).

Some of the early work in this area handles transparent motion by either just estimating the component motions (Shizawa and Mase 1991; Bergen, Burt *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 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).



Figure 9.16 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.

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 and saturated pixels provide an inequality constraint on the summed values. 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 9.16 and 9.17 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.

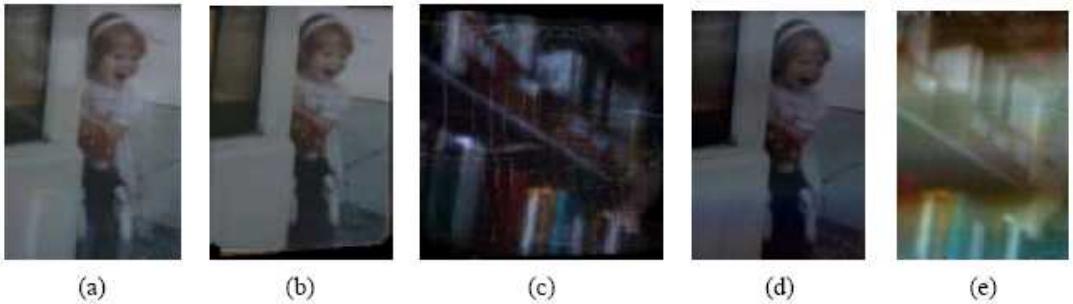


Figure 9.17 *Transparent motion separation (Szeliski, Avidan, and Anandan 2000) © 2000 IEEE:* (a) first image from input sequence; (b) dominant motion layer min-composite; (c) secondary motion residual layer max-composite; (d–e) final estimated picture and reflection layers. Note that the reflected layers in (c) and (e) are doubled in intensity to better show their structure.

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 *et al.* 2002; Criminisi, Kang *et al.* 2005; Jacquet, Hane *et al.* 2013), as has the extension to scenes with more complex 3D depth (Tsin, Kang, and Szeliski 2006). While motion sequences used to evaluate optical flow techniques have also started to include reflection and transparency (Baker, Scharstein *et al.* 2011; Butler, Wulff *et al.* 2012), the ground truth flow estimates they provide and use for evaluation only include the dominant motion at each pixel, e.g., ignoring mist and reflections.

In more recent work, Sinha, Kopf *et al.* (2012) model 3D scenes with reflections captured from a moving camera using two layers with varying depth and reflectivity and then use these to produce image-based renderings (novel view synthesis), which we discuss in more detail in Section 14.2.1. Kopf, Langguth *et al.* (2013) extend the modeling and rendering component of this system to recover colored image gradients for each layer and then use gradient-domain rendering to reconstruct the novel views. Xue, Rubinstein *et al.* (2015) extend these models with a gradient sparsity prior to enable *obstruction-free photography* when looking through windows and fences. More recent papers on this topic include Yang, Li *et al.* (2016), Nandoriya, Elgarib *et al.* (2017), and Liu, Lai *et al.* (2020a). The advent of dual-pixel imaging sensors, originally designed to provide fast focusing, can also be used to remove reflections by separating gradients into different depth planes (Punnappurath and Brown 2019).

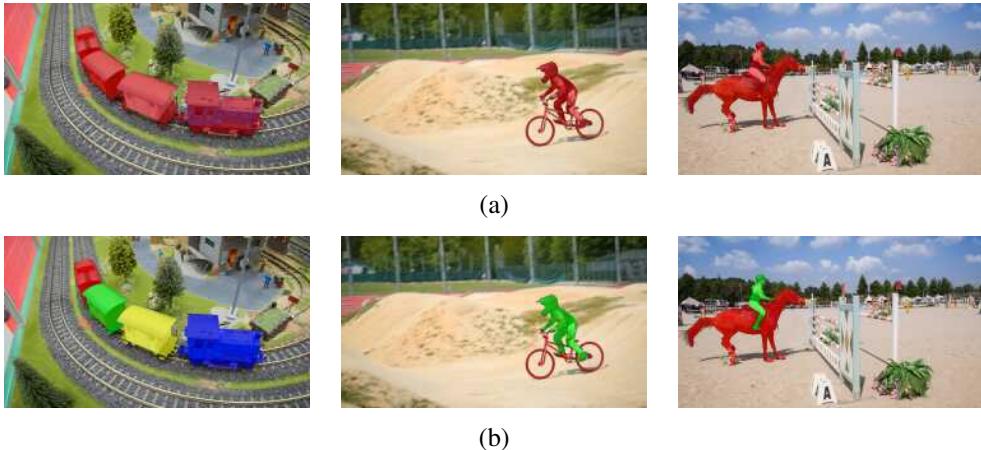


Figure 9.18 Sample sequences from the Densely Annotated Video Segmentation (DAVIS) datasets © Pont-Tuset, Perazzi et al. (2017). The DAVIS 2016 dataset (a) only contains foreground-background segmentations (red regions), while the DAVIS 2017 dataset (b) contains multiple annotated objects in each sequence (brightly colored regions).

While all of these techniques are useful for separating or eliminating reflections that appear as coherent images, more complex 3D geometries often give rise to spatially distributed *specularities* (Section 2.2.2) that are not amenable to layer-based representation. In such cases, lightfield representations such as surface lightfields (Section 14.3.2 and Figure 14.13) and neural light fields (Section 14.6 and Figure 14.24b) may be more appropriate.

9.4.3 Video object segmentation

As we have seen throughout this chapter, the accurate estimation of motion usually requires the segmentation of a video into coherently moving regions or objects as well as the correct modeling of occlusions. Segmenting a video clip into coherent objects is the temporal analog to still image segmentation, which we studied in Section 7.5. In addition to providing more accurate motion estimates, video object segmentation supports a variety of editing tasks, such as object removal and insertion (Section 10.4.5) as well as video understanding and interpretation.

While the segmentation of foreground and background layers has been studied for a long time (Bergen, Anandan et al. 1992; Wang and Adelson 1994; Gorelick, Blank et al. 2007; Lee and Grauman 2010; Brox and Malik 2010b; Lee, Kim, and Grauman 2011; Fragkiadaki, Zhang, and Shi 2012; Papazoglou and Ferrari 2013; Wang, Shen, and Porikli 2015; Perazzi,

Wang *et al.* 2015), the introduction of DAVIS (Densely Annotated VIdeo Segmentation) by Perazzi, Pont-Tuset *et al.* (2016) greatly accelerated research in this area. Figure 9.18a shows some frames from the original DAVIS 2016 dataset, where the first frame is annotated with a foreground pixel mask (shown in red) and the task is to estimate foreground masks for the remaining frames. The DAVIS 2017 dataset (Pont-Tuset, Perazzi *et al.* 2017) increased the number of video clips from 50 to 150, added more challenging elements such as motion blur and foreground occlusions, and most importantly, added more than one annotated object per sequence (Figure 9.18b).

Algorithm for video object segmentation such as OSVOS (Caelles, Maninis *et al.* 2017), FusionSeg (Jain, Xiong, and Grauman 2017), MaskTrack (Perazzi, Khoreva *et al.* 2017), and SegFlow (Cheng, Tsai *et al.* 2017), usually consist of a deep per-frame segmentation network as well as a motion estimation algorithm, which is used to link and refine the segmentations. Some approaches (Caelles, Maninis *et al.* 2017; Khoreva, Benenson *et al.* 2019) also fine-tune the segmentation networks based on the first frame annotations. More recent approaches have focused on increasing the computational efficiency of the pipelines (Chen, Pont-Tuset *et al.* 2018; Cheng, Tsai *et al.* 2018; Wug Oh, Lee *et al.* 2018; Wang, Zhang *et al.* 2019; Meinhardt and Leal-Taixé 2020).

Since 2017, an annual challenge and workshop on the DAVIS dataset have been held in conjunction with CVPR. More recent additions to the challenges have been segmentation with weaker annotations/scribbles (Caelles, Montes *et al.* 2018) or completely unsupervised segmentation, where the algorithms compute temporally linked segmentations of the video frames (Caelles, Pont-Tuset *et al.* 2019). There is also a newer, larger, dataset called YouTube-VOS (Xu, Yang *et al.* 2018) with its own associated set of challenges and leaderboards. The number of papers published on the topic continues to be high. The best sources for recent work are the challenge leaderboards at <https://davischallenge.org> and <https://youtube-vos.org>, which are accompanied by short papers describing the techniques, as well as the large number of conference papers, which usually have “Video Object Segmentation” in their titles.

9.4.4 Video object tracking

One of the most widely used applications of computer vision to video analysis is *video object tracking*. These applications include surveillance (Benfold and Reid 2011), animal and cell tracking (Khan, Balch, and Dellaert 2005), sports player tracking (Lu, Ting *et al.* 2013), and automotive safety (Janai, Güney *et al.* 2020, Chapter 6).

We have already discussed simpler examples of tracking in previous chapters, including feature (patch) tracking in Section 7.1.5 and contour tracking in Section 7.3. Surveys and

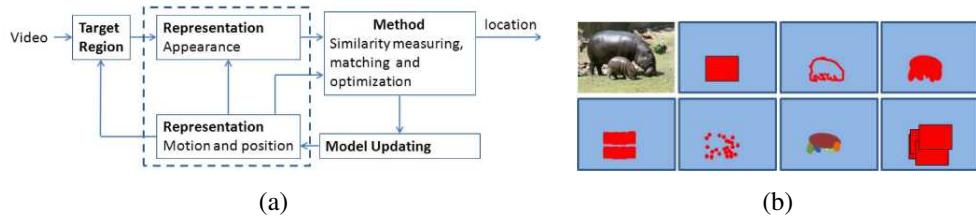


Figure 9.19 Visual object tracking (Smeulders, Chu et al. 2014) ©2014 IEEE: (a) high-level model showing main tracker components; (b) some tracked region representations, including a single bounding box, contour, blob, patch-based, sparse features, parts, and multiple bounding boxes.

experimental evaluation of such techniques include Lepetit and Fua (2005), Yilmaz, Javed, and Shah (2006), Wu, Lim, and Yang (2013), and Janai, Güney et al. (2020, Chapter 6).

A great starting point for learning more about tracking is the survey and tutorial by Smeulders, Chu et al. (2014), which was also one of the first large-scale tracking datasets, with over 300 video clips, ranging from a few seconds to a few minutes. Figure 9.19a shows some of the main components usually present in an online tracking system, which include choosing representations for shape, motion, position, and appearance, as well as similarity measures, optimization, and optional model updating. Figure 9.19b shows some of the choices for representing shapes and appearance, including a single bounding box, contours, patches, features, and parts.

The paper includes a discussion of previous surveys and techniques, as well as datasets, evaluation measures, and the above-mentioned model choices. It then categorizes a selection of well-known and more recent algorithms into a taxonomy that includes simple matching with fixed templates, extended and constrained (sparse) appearance models, discriminative classifiers, and tracking by detection. The algorithms discussed and evaluated include KLT, as implemented by Baker and Matthews (2004), mean-shift (Comaniciu and Meer 2002) and fragments-based (Adam, Rivlin, and Shimshoni 2006) tracking, online PCA appearance models (Ross, Lim et al. 2008), sparse bases (Mei and Ling 2009), and Struct (Hare, Golodetz et al. 2015), which uses kernelized structured output support vector machine.

Around the same time (2013), a series of annual challenges and workshops on single-target short-term tracking called VOT (visual object tracking) began.²¹ In their journal paper describing the evaluation methodology, Kristan, Matas et al. (2016) evaluate recent trackers and find that variants of Struct as well as extensions of kernelized correlation filters (KCF),

²¹<https://www.votchallenge.net>

originally developed by [Henriques, Caseiro *et al.* \(2014\)](#), performed the best. Other highly influential papers from this era include ([Bertinetto, Valmadre *et al.* 2016a,b](#); [Danelljan, Robinson *et al.* 2016](#)). Since that time, deep networks have played an essential role in visual object tracking, often using Siamese networks (Section [5.3.4](#); [Bromley, Guyon *et al.* 1994](#); [Chopra, Hadsell, and LeCun 2005](#)) to map regions being tracked into neural embeddings. Lists and descriptions of more recent tracking algorithms can be found in the annual reports that accompany the VOT challenges and workshops, the most recent of which is [Kristan, Leonardis *et al.* \(2020\)](#).

In parallel with the single-object VOT challenges and workshops, a multiple object tracking was introduced as part of the KITTI vision benchmark ([Geiger, Lenz, and Urtasun 2012](#)) and a separate benchmark was developed by [Leal-Taixé, Milan *et al.* \(2015\)](#) along with a series of challenges, with the most recent results described in [Dendorfer, Ošep *et al.* \(2021\)](#).²² A survey of multiple object tracking papers through 2016 can be found in [Luo, Xing *et al.* \(2021\)](#). Simple and fast multiple object trackers include [Bergmann, Meinhardt, and Leal-Taixé \(2019\)](#) and [Zhou, Koltun, and Krähenbühl \(2020\)](#). Until recently, however, tracking datasets have focused mostly on people, vehicles, and animals. To expand the range of objects that can be tracked, [Dave, Khurana *et al.* \(2020\)](#) created the TAO (tracking any object) dataset, consisting of 2,907 videos, which were annotated “bottom-up” by first having users tag anything that moves and then classifying such objects into 833 categories.

While in this section, we have focused mostly on object tracking, the primary goal of which is to locate an object in contiguous video frames, it is also possible to simultaneously track and segment ([Voigtlaender, Krause *et al.* 2019](#); [Wang, Zhang *et al.* 2019](#)) or to track non-rigidly deforming objects such as T-shirts with deformable models from either video ([Kambhamettu, Goldgof *et al.* 2003](#); [White, Crane, and Forsyth 2007](#); [Pilet, Lepetit, and Fua 2008](#); [Furukawa and Ponce 2008](#); [Salzmann and Fua 2010](#)) or RGB-D streams ([Božič, Zollhöfer *et al.* 2020](#); [Božič, Palafox *et al.* 2020, 2021](#)). The recent TrackFormer paper by [Meinhardt, Kirillov *et al.* \(2021\)](#) includes a nice review of recent work of multi-object tracking and segmentation.

9.5 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](#)

²²<https://motchallenge.net>

2003).²³ 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 *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 *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 *et al.* 1994; Srinivasan, Chellappa *et al.* 2005; Matsushita, Ofek *et al.* 2006), and video compression (Irani, Hsu, and Anandan 1995; Lee, Chen *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 optical flow algorithms include those by Aggarwal and Nandhakumar (1988), Barron, Fleet, and Beauchemin (1994), Otte and Nagel (1994), Mitiche and Boutheny (1996), Stiller and Konrad (1999), McCane, Novins *et al.* (2001), Szeliski (2006a), and Baker, Scharstein *et al.* (2011), Sun, Yang *et al.* (2018), Janai, Güney *et al.* (2020), and Hur and Roth (2020). 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 *et al.* 1992; Scharstein 1994; Zabih and Woodfill 1994; Cox, Roy, and Hingorani 1995; Viola and Wells III 1997; Negahdaripour 1998; Kim, Kolmogorov, and Zabih 2003; Jia and Tang 2003; Papenberg, Bruhn *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 the first large benchmark for evaluating optical flow algorithms by Baker, Scharstein *et al.* (2011) led to rapid advances in the quality of estimation algorithms. While most of the best performing algorithms used robust data and smoothness norms such as L_1 or TV and continuous variational optimization techniques, some algorithms used discrete optimization or segmentation (Papenberg, Bruhn *et al.* 2006; Trobin, Pock *et al.* 2008; Xu, Chen, and Jia 2008; Lempitsky, Roth, and Rother 2008; Werlberger, Trobin *et al.* 2009; Lei and Yang 2009; Wedel, Cremers *et al.* 2009).

The creation of the Sintel (Butler, Wulff *et al.* 2012) and KITTI (Geiger, Lenz, and Urtasun 2012)

²³<https://www.itu.int/rec/T-REC-H.264>.

sun 2012) datasets further accelerated progress in optical flow algorithms. Significant papers from this past decade include Weinzaepfel, Revaud *et al.* (2013), Sun, Roth, and Black (2014), Revaud, Weinzaepfel *et al.* (2015), Ilg, Mayer *et al.* (2017), Xu, Ranftl, and Koltun (2017), Sun, Yang *et al.* (2018, 2019), Hur and Roth (2019), and Teed and Deng (2020b). Good review of flow papers from the last decade can be found in Sun, Yang *et al.* (2018), Janai, Güney *et al.* (2020), and Hur and Roth (2020).

Good starting places to read about video object segmentation and video object tracking are recent workshops associated with the main datasets and challenges on these topics (Pont-Tuset, Perazzi *et al.* 2017; Xu, Yang *et al.* 2018; Kristan, Leonardis *et al.* 2020; Dave, Khurana *et al.* 2020; Dendorfer, Ošep *et al.* 2021).

9.6 Exercises

Ex 9.1: Correlation. Implement and compare the performance of the following correlation algorithms:

- sum of squared differences (9.1)
- sum of robust differences (9.2)
- sum of absolute differences (9.3)
- bias–gain compensated squared differences (9.9)
- normalized cross-correlation (9.11)
- windowed versions of the above (9.22–9.23)
- Fourier-based implementations of the above measures (9.18–9.20)
- phase correlation (9.24)
- 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 <https://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 (9.5) into a sum of pairwise correlations, just as in (9.22). (This is part of the exercise.)
3. For the bias–gain corrected variant of squared differences (9.9), 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 9.2: 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 *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 9.3: Stabilization. Write a program to stabilize an input video sequence. You could implement the following steps, as described in Section 9.2.1:

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 9.4: 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 <https://vision.middlebury.edu/flow> or <http://sintel.is.tue.mpg.de> and compare your results (visually) to those available on these websites. 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 9.5).
3. What can you say about the relative efficiency (speed) of your approach?

Ex 9.5: Automated morphing and frame interpolation. Write a program to automatically morph between pairs of images. Implement the following steps, as sketched out in Section 9.4.1 and by Baker, Scharstein *et al.* (2011):

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 9.6: Video denoising. Implement the algorithm sketched in Application 9.3.4. 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 3.4.2 and Exercise 3.12).

4. Devise a fall-back strategy for areas where you don't think the flow estimates are accurate enough.

Ex 9.7: Layered motion estimation. Decompose into separate layers (Section 9.4) 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 by 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 *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 *et al.* 2008).

Ex 9.8: Transparent motion and reflection estimation. Take a video sequence looking through a window (or picture frame) and see if you can remove the reflection to better see what is inside.

The steps are described in Section 9.4.2 and by Szeliski, Avidan, and Anandan (2000). Alternative approaches can be found in work by Shizawa and Mase (1991), Bergen, Burt *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).

Ex 9.9: Motion segmentation. Write a program to segment an image into separately moving regions or to reliably find motion boundaries.

Use the DAVIS motion segmentation database (Pont-Tuset, Perazzi *et al.* 2017) as some of your test data.

Ex 9.10: Video object tracking. Write an object tracker and test it out on one of the latest video object tracking datasets (Leal-Taixé, Milan *et al.* 2015; Kristan, Matas *et al.* 2016; Dave, Khurana *et al.* 2020; Kristan, Leonardis *et al.* 2020; Dendorfer, Ošep *et al.* 2021).

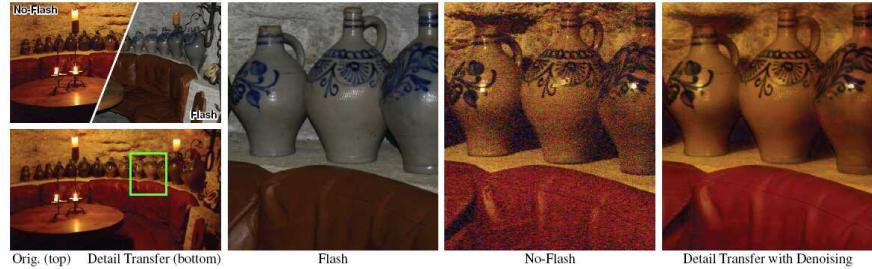
Chapter 10

Computational photography

10.1	Photometric calibration	610
10.1.1	Radiometric response function	611
10.1.2	Noise level estimation	614
10.1.3	Vignetting	615
10.1.4	Optical blur (spatial response) estimation	616
10.2	High dynamic range imaging	620
10.2.1	Tone mapping	627
10.2.2	<i>Application:</i> Flash photography	634
10.3	Super-resolution, denoising, and blur removal	637
10.3.1	Color image demosaicing	646
10.3.2	Lens blur (bokeh)	648
10.4	Image matting and compositing	650
10.4.1	Blue screen matting	651
10.4.2	Natural image matting	653
10.4.3	Optimization-based matting	656
10.4.4	Smoke, shadow, and flash matting	661
10.4.5	Video matting	662
10.5	Texture analysis and synthesis	663
10.5.1	<i>Application:</i> Hole filling and inpainting	665
10.5.2	<i>Application:</i> Non-photorealistic rendering	667
10.5.3	Neural style transfer and semantic image synthesis	669
10.6	Additional reading	671
10.7	Exercises	674



(a)



(b)



(c)



(d)

Figure 10.1 Computational photography: (a) merging multiple exposures to create high dynamic range images (Debevec and Malik 1997) © 1997 ACM; (b) merging flash and non-flash photographs; (Petschnigg, Agrawala et al. 2004) © 2004 ACM; (c) image matting and compositing; (Chuang, Curless et al. 2001) © 2001 IEEE; (d) hole filling with inpainting (Criminisi, Pérez, and Toyama 2004) © 2004 IEEE.

Of all the advances in computer vision in the last decade, computational photography has arguably had the most widespread commercial impact. In 2010, the seminal *Frankencamera* paper by Adams, Talvala *et al.* (2010) had just been released, as had one of the first widely used in-camera panoramic image stitching apps.¹ Fast forward to 2020, and every smartphone now has built-in panoramic stitching, high dynamic range (HDR) exposure merging, and multi-image denoising and super-resolution (Hasinoff, Sharlet *et al.* 2016; Wronski, Garcia-Dorado *et al.* 2019; Liba, Murthy *et al.* 2019), and the newest phones are also simulating shallow depth of field (bokeh) with multiple lenses or dual pixels (Barron, Adams *et al.* 2015; Wadhwa, Garg *et al.* 2018; Garg, Wadhwa *et al.* 2019; Zhang, Wadhwa *et al.* 2020).

In Section 8.2, we described how to stitch multiple images into wide field of view panoramas, allowing us to create photographs that could not be captured with a regular camera. This is just one instance of *computational photography*, where image analysis and processing algorithms are applied to one or more photographs to create images that go beyond the capabilities of traditional imaging systems.

In this chapter, we cover a number of additional computational photography algorithms. We begin with a review of photometric image calibration (Section 10.1), i.e., the measurement of camera and lens responses, which is a prerequisite for many of the algorithms we describe later. We then discuss *high dynamic range imaging* (Section 10.2), which captures the full range of brightness in a scene through the use of multiple exposures (Figure 10.1a). We also discuss *tone mapping operators*, which map wide-gamut images back into regular display devices such as screens and printers, as well as algorithms that merge flash and regular images to obtain better exposures (Figure 10.1b).

Next, we discuss how the resolution and visual quality of images can be improved either by merging multiple photographs together or using sophisticated image priors or deep networks (Section 10.3). This includes algorithms for extracting full-color images from the patterned Bayer mosaics present in most cameras.

In Section 10.4, we discuss algorithms for cutting pieces of images from one photograph and pasting them into others (Figure 10.1c). In Section 10.5, we describe how to generate novel textures from real-world samples for applications such as filling holes in images (Figure 10.1d). We close with a brief overview of *non-photorealistic rendering* (Section 10.5.2), which can turn regular photographs into artistic renderings that resemble traditional drawings and paintings, and a discussion of neural network approaches to style transfer and semantic image synthesis (Section 10.5.3).

One topic that we do not cover extensively in this book is novel computational sensors, optics, and cameras. A nice survey can be found in an article by Nayar (2006), the book by

¹https://en.wikipedia.org/wiki/Photosynth#Mobile_apps

Raskar and Tumblin (2010), and research papers such as Levin, Fergus *et al.* (2007). Some related discussion can also be found in Sections 10.2 and 14.3.

A good general-audience introduction to computational photography can be found in the article by Hayes (2008) as well as survey papers by Nayar (2006), Cohen and Szeliski (2006), Levoy (2006), andDebevec (2006).² Raskar and Tumblin (2010) give extensive coverage of topics in this area, with particular emphasis on computational cameras and sensors. The sub-field of high dynamic range imaging has its own book discussing research in this area (Reinhard, Heidrich *et al.* 2010), as well as a wonderful book aimed more at professional photographers (Freeman 2008).³ A good survey of image matting is provided by Wang and Cohen (2009).

There are also several courses on computational photography where the instructors have provided extensive online materials, e.g., Yannis Gkioulekas' class at Carnegie Mellon,⁴ Alyosha Efros' class at Berkeley,⁵ Frédo Durand's Computation Photography course at MIT,⁶ Marc Levoy's class at Stanford,⁷ and a series of SIGGRAPH courses on Computational Photography.⁸

10.1 Photometric calibration

Before we can successfully merge multiple photographs, we need to characterize the functions that map incoming irradiance into pixel values and also the amount of noise present in each image. In this section, we examine three components of the imaging pipeline (Figure 10.2) that affect this mapping. For a more comprehensive, tunable model of modern digital camera processing pipelines, see the recent paper by Tseng, Yu *et al.* (2019).

The first is the *radiometric response function* (Mitsunaga and Nayar 1999), which maps photons arriving at the lens into digital values stored in the image file (Section 10.1.1). The second is *vignetting*, which darkens pixel values near the periphery of images, especially at large apertures (Section 10.1.3). The third is the *point spread function*, which characterizes the blur induced by the lens, anti-aliasing filters, and finite sensor areas (Section 10.1.4).⁹ The material in this section builds on the image formation processes described in Sections 2.2.3

²See also the two special issue journals edited by Bimber (2006) and Durand and Szeliski (2007).

³Gulbins and Gulbins (2009) discuss related photographic techniques.

⁴CMU 15-463, <http://graphics.cs.cmu.edu/courses/15-463>

⁵Berkeley CS194-26/294-26, <https://inst.eecs.berkeley.edu/~cs194-26/fa20>

⁶MIT 6.815/6.865, <https://stellar.mit.edu/S/course/6/sp15/6.815>

⁷Stanford CS 448A, <https://graphics.stanford.edu/courses/cs448a-10>

⁸<https://web.media.mit.edu/~raskar/photo>.

⁹Additional photometric camera and lens effects include sensor glare, blooming, and chromatic aberration, which can also be thought of as a spectrally varying form of geometric aberration (Section 2.2.3).

and 2.3.3, so if it has been a while since you looked at those sections, please go back and review them.

10.1.1 Radiometric response function

As we can see in Figure 10.2, a number of factors affect how the intensity of light arriving at the lens ends up being mapped into stored digital values. Let us ignore for now any non-uniform attenuation that may occur inside the lens, which we cover in Section 10.1.3.

The first factors to affect this mapping are the aperture and shutter speed (Section 2.3), which can be modeled as global multipliers on the incoming light, most conveniently measured in *exposure values* (\log_2 brightness ratios). Next, the analog to digital (A/D) converter on the sensing chip applies an electronic gain, usually controlled by the ISO setting on your camera. While in theory this gain is linear, as with any electronics non-linearities may be present (either unintentionally or by design). Ignoring, for now, photon noise, on-chip noise, amplifier noise, and quantization noise, which we discuss shortly, you can often assume that the mapping between incoming light and the values stored in a RAW camera file (if your camera supports this) is roughly linear.

If images are being stored in the more common JPEG format, the camera's image signal processor (ISP) next performs Bayer pattern demosaicing (Sections 2.3.2 and 10.3.1), which is a mostly linear (but often non-stationary) process. Some sharpening is also often applied at this stage. Next, the color values are multiplied by different constants (or sometimes a 3×3 color twist matrix) to perform color balancing, i.e., to move the white point closer to pure white. Finally, a standard gamma is applied to the intensities in each color channel and the colors are converted into YCbCr format before being transformed by a DCT, quantized, and then compressed into the JPEG format (Section 2.3.3). Figure 10.2 shows all of these steps in pictorial form.

Given the complexity of all of this processing, it is difficult to model the camera response function (Figure 10.3a), i.e., the mapping between incoming irradiance and digital RGB values, from first principles. A more practical approach is to calibrate the camera by measuring correspondences between incoming light and final values.

The most accurate, but most expensive, approach is to use an *integrating sphere*, which is a large (typically 1m diameter) sphere carefully painted on the inside with white matte paint. An accurately calibrated light at the top controls the amount of radiance inside the sphere (which is constant everywhere because of the sphere's radiometry) and a small opening at the side allows for a camera/lens combination to be mounted. By slowly varying the current going into the light, an accurate correspondence can be established between incoming radiance and measured pixel values. The vignetting and noise characteristics of the camera can also be

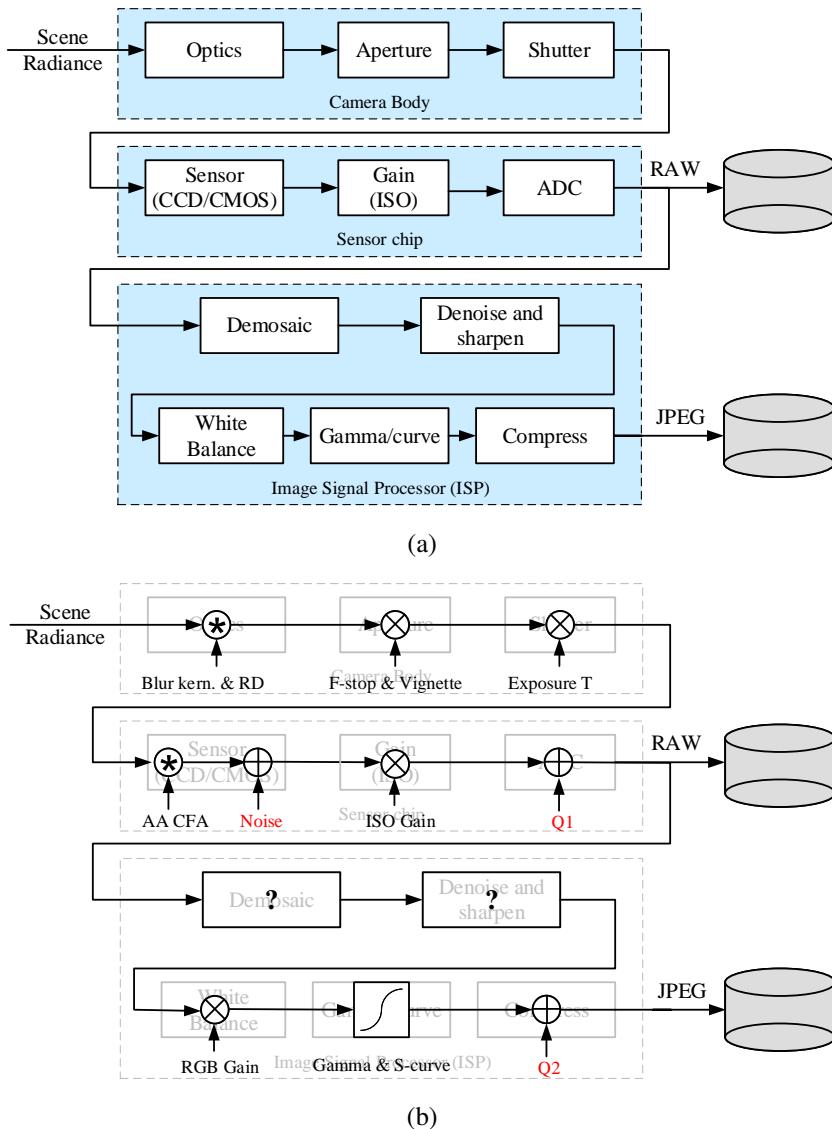


Figure 10.2 Image sensing pipeline: (a) block diagram showing the various sources of noise as well as the typical digital post-processing steps; (b) equivalent signal transforms, including convolution, gain, and noise injection. The abbreviations are: RD = radial distortion, AA = anti-aliasing filter, CFA = color filter array, $Q1$ and $Q2$ = quantization noise.

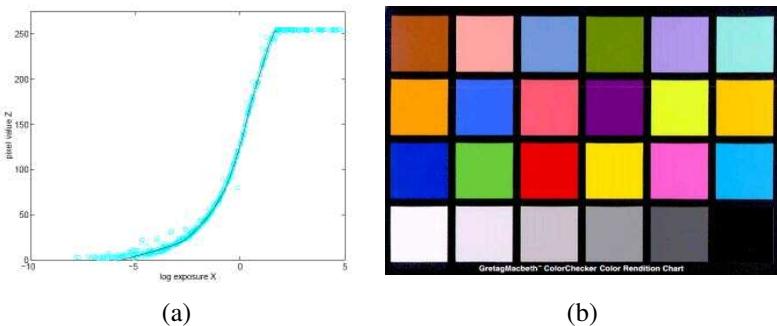


Figure 10.3 Radiometric response calibration: (a) typical camera response function, showing the mapping between incoming log irradiance (exposure) and output eight-bit pixel values, for one color channel (Debevec and Malik 1997) © 1997 ACM; (b) color checker chart.

simultaneously determined.

A more practical alternative is to use a calibration chart (Figure 10.3b) such as the Macbeth or Munsell ColorChecker Chart.¹⁰ The biggest problem with this approach is to ensure uniform lighting. One approach is to use a large dark room with a high-quality light source far away from (and perpendicular to) the chart. Another is to place the chart outdoors away from any shadows. (The results will differ under these two conditions, because the color of the illuminant will be different.)

The easiest approach is probably to take multiple exposures of the same scene while the camera is on a tripod and to recover the response function by simultaneously estimating the incoming irradiance at each pixel and the response curve (Mann and Picard 1995; Debevec and Malik 1997; Mitsunaga and Nayar 1999). This approach is discussed in more detail in Section 10.2 on high dynamic range imaging.

If all else fails, i.e., you just have one or more unrelated photos, you can use an International Color Consortium (ICC) profile for the camera (Fairchild 2013).¹¹ Even more simply, you can just assume that the response is linear if they are RAW files and that the images have a $\gamma = 2.2$ non-linearity (plus clipping) applied to each RGB channel if they are JPEG images.

¹⁰<https://www.xrite.com>.

¹¹See also the ICC *Information on Profiles*, https://www.color.org/info_profiles2.xalter.

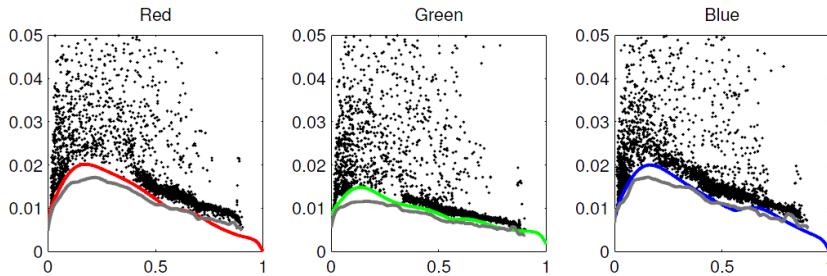


Figure 10.4 Noise level function estimates obtained from a single color photograph (Liu, Szeliski et al. 2008) © 2008 IEEE. The colored curves are the estimated NLF fit as the probabilistic lower envelope of the measured deviations between the noisy piecewise-smooth images. The ground truth NLFs obtained by averaging 29 images are shown in gray.

10.1.2 Noise level estimation

In addition to knowing the camera response function, it is also often important to know the amount of noise being injected under a particular camera setting (e.g., ISO/gain level). The simplest characterization of noise is a single standard deviation, usually measured in gray levels, independent of pixel value. A more accurate model can be obtained by estimating the noise level as a function of pixel value (Figure 10.4), which is known as the *noise level function* (Liu, Szeliski et al. 2008).

As with the camera response function, the simplest way to estimate these quantities is in the lab, using either an integrating sphere or a calibration chart. The noise can be estimated either at each pixel independently, by taking repeated exposures and computing the temporal variance in the measurements (Healey and Kondepudy 1994), or over regions, by assuming that pixel values should all be the same within some region (e.g., inside a color checker square) and computing a spatial variance.

This approach can be generalized to photos where there are regions of constant or slowly varying intensity (Liu, Szeliski et al. 2008). First, segment the image into such regions and fit a constant or linear function inside each region. Next, measure the (spatial) standard deviation of the differences between the noisy input pixels and the smooth fitted function away from large gradients and region boundaries. Plot these as a function of output level for each color channel, as shown in Figure 10.4. Finally, fit a lower envelope to this distribution to ignore pixels or deviations that are outliers. A fully Bayesian approach to this problem that models the statistical distribution of each quantity is presented by Liu, Szeliski et al. (2008). A simpler approach, which should produce useful results in most cases, is to fit a low-dimensional



Figure 10.5 *Single image vignetting correction (Zheng, Yu et al. 2008) © 2008 IEEE: (a) original image with strong visible vignetting; (b) vignetting compensation as described by Zheng, Zhou et al. (2006); (c–d) vignetting compensation as described by Zheng, Yu et al. (2008).*

function (e.g., positive valued B-spline) to the lower envelope (see Exercise 10.2).

Matsushita and Lin (2007b) present a technique for simultaneously estimating a camera's response and noise level functions based on skew (asymmetries) in level-dependent noise distributions. Their paper also contains extensive references to previous work in these areas.

10.1.3 Vignetting

A common problem with using wide-angle and wide-aperture lenses is that the image tends to darken in the corners (Figure 10.5a). This problem is generally known as *vignetting* and comes in several different forms, including natural, optical, and mechanical vignetting (Section 2.2.3) (Ray 2002). As with radiometric response function calibration, the most accurate way to calibrate vignetting is to use an integrating sphere or a picture of a uniformly colored and illuminated blank wall.

An alternative approach is to stitch a panoramic scene and to assume that the true radiance at each pixel comes from the central portion of each input image. This is easier to do if the radiometric response function is already known (e.g., by shooting in RAW mode) and if the exposure is kept constant. If the response function, image exposures, and vignetting function are unknown, they can still be recovered by optimizing a large least squares fitting problem (Litvinov and Schechner 2005; Goldman 2010). Figure 10.6 shows an example of simultaneously estimating the vignetting, exposure, and radiometric response function from a set of overlapping photographs (Goldman 2010). Note that unless vignetting is modeled and compensated, regular gradient-domain image blending (Section 8.4.4) will not create an attractive image.

If only a single image is available, vignetting can be estimated by looking for slow consistent intensity variations in the radial direction. The original algorithm proposed by Zheng, Lin, and Kang (2006) first pre-segmented the image into smoothly varying regions and then



Figure 10.6 Simultaneous estimation of vignetting, exposure, and radiometric response (Goldman 2010) © 2011 IEEE: (a) original average of the input images; (b) after compensating for vignetting; (c) using gradient domain blending only (note the remaining mottled look); (d) after both vignetting compensation and blending.

performed an analysis inside each region. Instead of pre-segmenting the image, Zheng, Yu *et al.* (2008) compute the radial gradients at all the pixels and use the asymmetry in this distribution (because gradients away from the center are, on average, slightly negative) to estimate the vignetting. Figure 10.5 shows the results of applying each of these algorithms to an image with a large amount of vignetting. Exercise 10.3 has you implement some of the above techniques.

10.1.4 Optical blur (spatial response) estimation

One final characteristic of imaging systems that you should calibrate is the spatial response function, which encodes the optical blur that gets convolved with the incoming image to produce the point-sampled image. The shape of the convolution kernel, which is also known as the *point spread function (PSF)* or *optical transfer function*, depends on several factors, including lens blur and radial distortion (Section 2.2.3), anti-aliasing filters in front of the sensor, and the shape and extent of each active pixel area (Section 2.3) (Figure 10.2). A good estimate of this function is required for applications such as multi-image super-resolution and deblurring (Section 10.3).

In theory, one could estimate the PSF by simply observing an infinitely small point light source everywhere in the image. Creating an array of samples by drilling through a dark plate and backlighting with a very bright light source is difficult in practice.

A more practical approach is to observe an image composed of long straight lines or

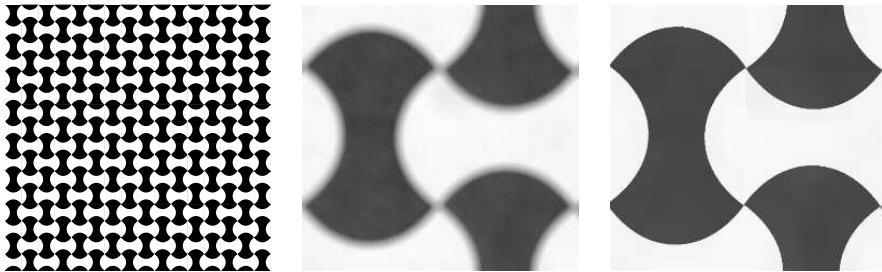


Figure 10.7 Calibration pattern with edges equally distributed at all orientations that can be used for PSF and radial distortion estimation (Joshi, Szeliski, and Kriegman 2008) © 2008 IEEE. A portion of an actual sensed image is shown in the middle and a close-up of the ideal pattern is on the right.

bars, as these can be fitted to arbitrary precision. Because the location of a horizontal or vertical edge can be *aliased* during acquisition, slightly slanted edges are preferred. The profile and locations of such edges can be estimated to sub-pixel precision, which makes it possible to estimate the PSF at sub-pixel resolutions (Reichenbach, Park, and Narayanswamy 1991; Burns and Williams 1999; Williams and Burns 2001; Goesele, Fuchs, and Seidel 2003). The thesis by Murphy (2005) contains a nice survey of all aspects of camera calibration, including the spatial frequency response (SFR), spatial uniformity, tone reproduction, color reproduction, noise, dynamic range, color channel registration, and depth of field. It also includes a description of a slant-edge calibration algorithm called `sfrm2`.

The slant-edge technique can be used to recover a 1D projection of the 2D PSF, e.g., slightly vertical edges are used to recover the horizontal *line spread function* (LSF) (Williams 1999). The LSF is then often converted into the Fourier domain and its magnitude plotted as a one-dimensional *modulation transfer function* (MTF), which indicates which image frequencies are lost (blurred) and aliased during the acquisition process (Section 2.3.1). For most computational photography applications, it is preferable to directly estimate the full 2D PSF, as it can be hard to recover from its projections (Williams 1999).

Figure 10.7 shows a pattern containing edges at all orientations, which can be used to directly recover a two-dimensional PSF. First, corners in the pattern are located by extracting edges in the sensed image, linking them, and finding the intersections of the circular arcs. Next, the ideal pattern, whose analytic form is known, is warped (using a homography) to fit the central portion of the input image and its intensities are adjusted to fit the ones in the sensed image. If desired, the pattern can be rendered at a higher resolution than the input image, which enables the estimation of the PSF to sub-pixel resolution (Figure 10.8a). Finally

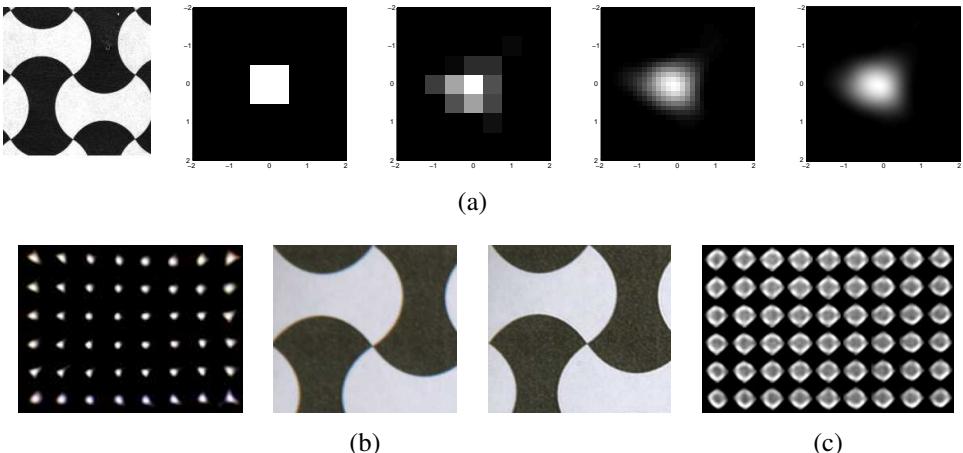


Figure 10.8 Point spread function estimation using a calibration target (Joshi, Szeliski, and Kriegman 2008) © 2008 IEEE. (a) Sub-pixel PSFs at successively higher resolutions (note the interaction between the square sensing area and the circular lens blur). (b) The radial distortion and chromatic aberration can also be estimated and removed. (c) PSF for a misfocused (blurred) lens showing some diffraction and vignetting effects.

a large linear least squares system is solved to recover the unknown PSF kernel K ,

$$K = \arg \min_K \|B - D(I * K)\|^2, \quad (10.1)$$

where B is the sensed (blurred) image, I is the predicted (sharp) image, and D is an optional downsampling operator that matches the resolution of the ideal and sensed images (Joshi, Szeliski, and Kriegman 2008). An alternative solution technique is to estimate 1D PSF profiles first and to then combine them using a Radon transform (Cho, Paris *et al.* 2011).

If the process of estimating the PSF is done locally in overlapping patches of the image, it can also be used to estimate the radial distortion and chromatic aberration induced by the lens (Figure 10.8b). Because the homography mapping the ideal target to the sensed image is estimated in the central (undistorted) part of the image, any (per-channel) shifts induced by the optics manifest themselves as a displacement in the PSF centers.¹² Compensating for these shifts eliminates both the achromatic radial distortion and the inter-channel shifts that result in visible chromatic aberration. The color-dependent blurring caused by chromatic aberration (Figure 2.21) can also be removed using the deblurring techniques discussed in

¹²This process confounds the distinction between geometric and photometric calibration. In principle, any geometric distortion could be modeled by spatially varying displaced PSFs. In practice, it is easier to fold any large shifts into the geometric correction component.

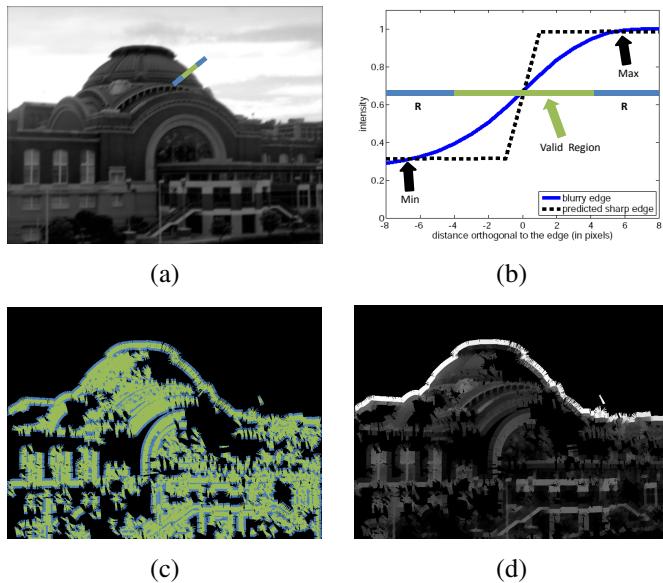


Figure 10.9 Estimating the PSF without using a calibration pattern (Joshi, Szeliski, and Kriegman 2008) © 2008 IEEE: (a) Input image with blue cross-section (profile) location, (b) Profile of sensed and predicted step edges, (c–d) Locations and values of the predicted colors near the edge locations.

Section 10.3. Figure 10.8b shows how the radial distortion and chromatic aberration manifest themselves as elongated and displaced PSFs, along with the result of removing these effects in a region of the calibration target.

The local 2D PSF estimation technique can also be used to estimate vignetting. Figure 10.8c shows how the mechanical vignetting manifests itself as clipping of the PSF in the corners of the image. For the overall dimming associated with vignetting to be properly captured, the modified intensities of the ideal pattern need to be extrapolated from the center, which is best done with a uniformly illuminated target.

When working with RAW Bayer-pattern images, the correct way to estimate the PSF is to only evaluate the least squares terms in (10.1) at sensed pixel values, while interpolating the ideal image to all values. For JPEG images, you should linearize your intensities first, e.g., remove the gamma and any other non-linearities in your estimated radiometric response function.

What if you have an image that was taken with an uncalibrated camera? Can you still recover the PSF and use it to correct the image? In fact, with a slight modification, the previous



Figure 10.10 Sample indoor image where the areas outside the window are overexposed and inside the room are too dark.

algorithms still work.

Instead of assuming a known calibration image, you can detect strong elongated edges and fit ideal step edges in such regions (Figure 10.9b), resulting in the sharp image shown in Figure 10.9d. For every pixel that is surrounded by a complete set of valid estimated neighbors (green pixels in Figure 10.9c), apply the least squares formula (10.1) to estimate the kernel K . The resulting locally estimated PSFs can be used to correct for chromatic aberration (because the relative displacements between per-channel PSFs can be computed), as shown by Joshi, Szeliski, and Kriegman (2008).

Exercise 10.4 provides some more detailed instructions for implementing and testing edge-based PSF estimation algorithms. An alternative approach, which does not require the explicit detection of edges but uses image statistics (gradient distributions) instead, is presented by Fergus, Singh *et al.* (2006).

10.2 High dynamic range imaging

As we mentioned earlier in this chapter, registered images taken at different exposures can be used to calibrate the radiometric response function of a camera. More importantly, they can help you create well-exposed photographs under challenging conditions, such as brightly lit scenes where any single exposure contains saturated (overexposed) and dark (underexposed) regions (Figure 10.10). This problem is quite common, because the natural world contains a range of radiance values that is far greater than can be captured with any photographic sensor or film (Figure 10.11). Taking a set of *bracketed exposures* (exposures taken by a camera in automatic exposure bracketing (AEB) mode to deliberately under- and over-expose the image) gives you the material from which to create a properly exposed photograph, as shown in Figure 10.12 (Freeman 2008; Gulbins and Gulbins 2009; Hasinoff, Durand, and Freeman



Figure 10.11 Relative brightness of different scenes, ranging from 1 inside a dark room lit by a monitor to 2,000,000 looking at the Sun. Photos courtesy of Paul Debevec.



Figure 10.12 A bracketed set of shots (using the camera’s automatic exposure bracketing (AEB) mode) and the resulting high dynamic range (HDR) composite.

2010; Reinhard, Heidrich *et al.* 2010).

While it is possible to combine pixels from different exposures directly into a final composite (Burt and Kolczynski 1993; Mertens, Kautz, and Reeth 2007), this approach runs the risk of creating contrast reversals and halos. Instead, the more common approach is to proceed in three stages:

1. Estimate the radiometric response function from the aligned images.
2. Estimate a *radiance map* by selecting or blending pixels from different exposures.
3. Tone map the resulting high dynamic range (HDR) image back into a displayable gamut.

The idea behind estimating the radiometric response function is relatively straightforward (Mann and Picard 1995; Debevec and Malik 1997; Mitsunaga and Nayar 1999; Reinhard, Heidrich *et al.* 2010). Suppose you take three sets of images at different exposures (shutter speeds), say at ± 2 exposure values.¹³ If we were able to determine the irradiance (exposure) E_i at each pixel (2.102), we could plot it against the measured pixel value z_{ij} for each exposure time t_j , as shown in Figure 10.13.

¹³Changing the shutter speed is preferable to changing the aperture, as the latter can modify the vignetting and focus. Using ± 2 “f-stops” (technically, exposure values, or EVs, as f-stops refer to apertures) is usually the right compromise between capturing a good dynamic range and having properly exposed pixels everywhere.

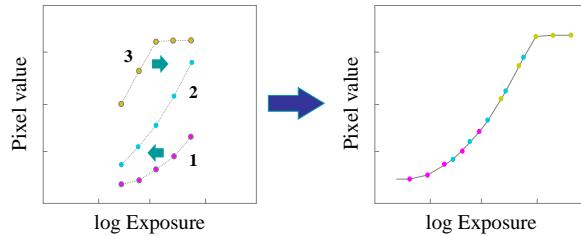


Figure 10.13 Radiometric calibration using multiple exposures (Debevec and Malik 1997). Corresponding pixel values are plotted as functions of log exposures (irradiance). The curves on the left are shifted to account for each pixel’s unknown radiance until they all line up into a single smooth curve.

Unfortunately, we do not know the irradiance values E_i , so these have to be estimated at the same time as the radiometric response function f , which can be written (Debevec and Malik 1997) as

$$z_{ij} = f(E_i t_j), \quad (10.2)$$

where t_j is the exposure time for the j th image. The inverse response curve f^{-1} is given by

$$f^{-1}(z_{ij}) = E_i t_j. \quad (10.3)$$

Taking logarithms of both sides (base 2 is convenient, as we can now measure quantities in EVs), we obtain

$$g(z_{ij}) = \log f^{-1}(z_{ij}) = \log E_i + \log t_j, \quad (10.4)$$

where $g = \log f^{-1}$ (which maps pixel values z_{ij} into log irradiance) is the curve we are estimating (Figure 10.13 turned on its side).

Debevec and Malik (1997) assume that the exposure times t_j are known. (Recall that these can be obtained from a camera’s EXIF tags, but that they actually follow a power of 2 progression $\dots, 1/128, 1/64, 1/32, 1/16, 1/8, \dots$ instead of the marked $\dots, 1/125, 1/60, 1/30, 1/15, 1/8, \dots$ values—see Exercise 2.5.) The unknowns are therefore the per-pixel exposures E_i and the response values $g_k = g(k)$, where g can be discretized according to the 256 pixel values commonly observed in eight-bit images. (The response curves are calibrated separately for each color channel.)

In order to make the response curve smooth, Debevec and Malik (1997) add a second-order smoothness constraint

$$\lambda \sum_k g''(k)^2 = \lambda \sum [g(k-1) - 2g(k) + g(k+1)]^2, \quad (10.5)$$

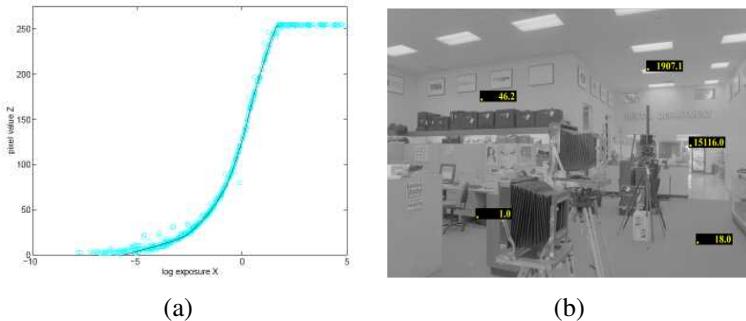


Figure 10.14 Recovered response function and radiance image for a real digital camera (DCS460) (Debevec and Malik 1997) © 1997 ACM.

which is similar to the one used in snakes (7.27). Because pixel values are more reliable in the middle of their range (and the g function becomes singular near saturation values), they also add a weighting (hat) function $w(k)$ that decays to zero at both ends of the pixel value range,

$$w(z) = \begin{cases} z - z_{\min} & z \leq (z_{\min} + z_{\max})/2 \\ z_{\max} - z & z > (z_{\min} + z_{\max})/2. \end{cases} \quad (10.6)$$

Putting all of these terms together, they obtain a least squares problem in the unknowns $\{g_k\}$ and $\{E_i\}$,

$$E = \sum_i \sum_j w(z_{i,j}) [g(z_{i,j}) - \log E_i - \log t_j]^2 + \lambda \sum_k w(k) g''(k)^2. \quad (10.7)$$

(To remove the overall shift ambiguity in the response curve and irradiance values, the middle of the response curve is set to 0.) Debevec and Malik (1997) show how this can be implemented in 21 lines of MATLAB code, which partially accounts for the popularity of their technique.

While Debevec and Malik (1997) assume that the exposure times t_j are known exactly, there is no reason why these additional variables cannot be thrown into the least squares problem, constraining their final estimated values to lie close to their nominal values \hat{t}_j with an extra term $\eta \sum_j (t_j - \hat{t}_j)^2$.

Figure 10.14 shows the recovered radiometric response function for a digital camera along with select (relative) radiance values in the overall radiance map. Figure 10.15 shows the bracketed input images captured on color film and the corresponding radiance map. Note that while most research on high dynamic range imaging assumes that the radiometric (or camera) response function is independent of exposure, this is not actually the case. Rodríguez,

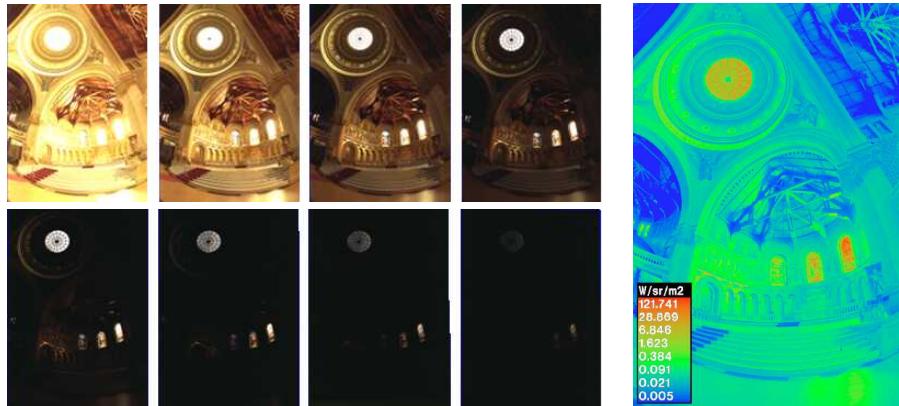


Figure 10.15 Bracketed set of exposures captured with a film camera and the resulting radiance image displayed in pseudocolor (Debevec and Malik 1997) © 1997 ACM.

Vazquez-Corral, and Bertalmío (2019) describe how to take this into account to get improved results.

While Debevec and Malik (1997) use a general second-order smooth curve g to parameterize their response curve, Mann and Picard (1995) use a three-parameter function

$$f(E) = \alpha + \beta E^\gamma, \quad (10.8)$$

while Mitsunaga and Nayar (1999) use a low-order ($N \leq 10$) polynomial for the inverse response function g . Pal, Szeliski *et al.* (2004) derive a Bayesian model that estimates an independent smooth response function for each image, which can better model the more sophisticated (and hence less predictable) automatic contrast and tone adjustment performed in today's digital cameras.

Once the response function has been estimated, the second step in creating high dynamic range photographs is to merge the input images into a composite *radiance map*. If the response function and images were known exactly, i.e., if they were noise free, you could use any non-saturated pixel value to estimate the corresponding radiance by mapping it through the inverse response curve $E = g(z)$.

Unfortunately, pixels are noisy, especially under low-light conditions when fewer photons arrive at the sensor. To compensate for this, Mann and Picard (1995) use the derivative of the response function as a weight in determining the final radiance estimate, because “flatter” regions of the curve tell us less about the incoming irradiance. Debevec and Malik (1997) use a hat function (10.6) which accentuates mid-tone pixels while avoiding saturated values. Mitsunaga and Nayar (1999) show that to maximize the signal-to-noise ratio (SNR),

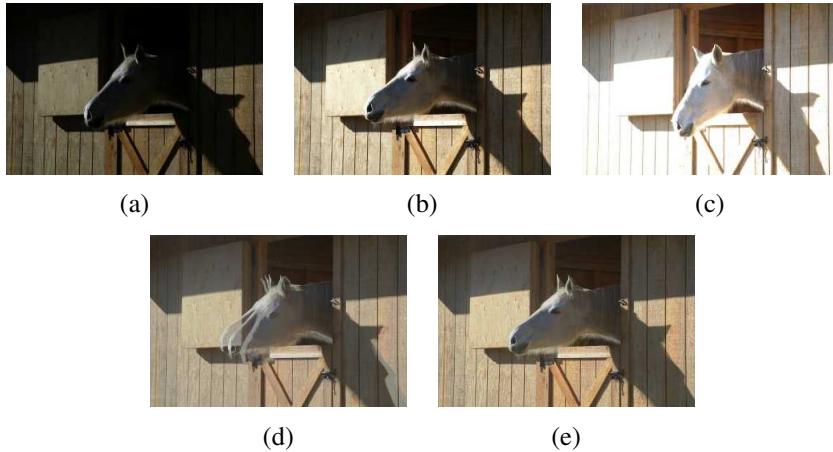


Figure 10.16 *Merging multiple exposures to create a high dynamic range composite (Kang, Uyttendaele *et al.* 2003): (a–c) three different exposures; (d) merging the exposures using classic algorithms (note the ghosting due to the horse’s head movement); (e) merging the exposures with motion compensation.*

the weighting function must emphasize both higher pixel values and larger gradients in the transfer function, i.e.,

$$w(z) = g(z)/g'(z), \quad (10.9)$$

where the weights w are used to form the final irradiance estimate

$$\log E_i = \frac{\sum_j w(z_{ij})[g(z_{ij}) - \log t_j]}{\sum_j w(z_{ij})}. \quad (10.10)$$

Exercise 10.1 has you implement one of the radiometric response function calibration techniques and then use it to create radiance maps.

Under real-world conditions, casually acquired images may not be perfectly registered and may contain moving objects. Ward (2003) uses a global (parametric) transform to align the input images, while Kang, Uyttendaele *et al.* (2003) present an algorithm that combines global registration with local motion estimation (optical flow) to accurately align the images before blending their radiance estimates (Figure 10.16). Because the images may have widely different exposures, care must be taken when estimating the motions, which must themselves be checked for consistency to avoid the creation of ghosts and object fragments.

Even this approach, however, may not work when the camera is simultaneously undergoing large panning motions and exposure changes, which is a common occurrence in casually acquired panoramas. Under such conditions, different parts of the image may be seen at one

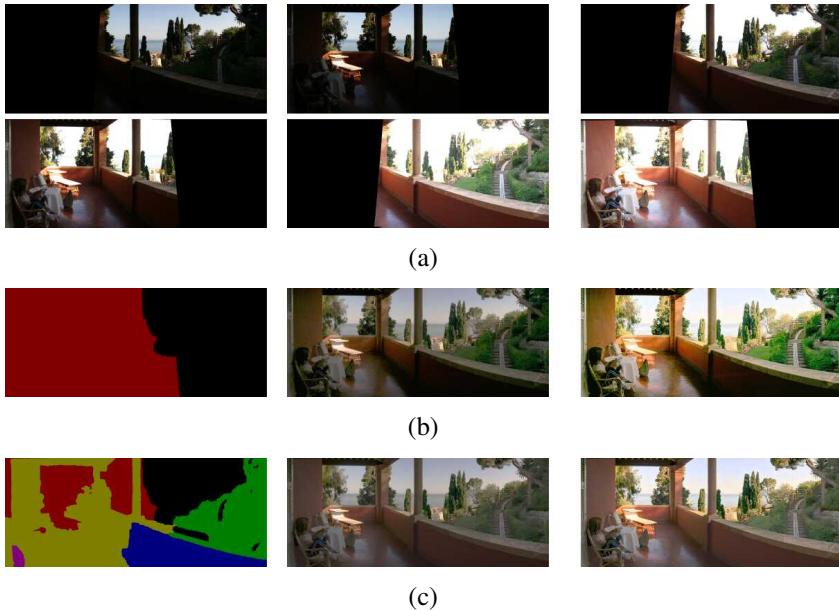


Figure 10.17 *HDR merging with large amounts of motion (Eden, Uyttendaele, and Szeliski 2006) © 2006 IEEE: (a) registered bracketed input images; (b) results after the first pass of image selection: reference labels, image, and tone-mapped image; (c) results after the second pass of image selection: final labels, compressed HDR image, and tone-mapped image*

or more exposures. Devising a method to blend all of these different sources while avoiding sharp transitions and dealing with scene motion is a challenging problem. One approach is to first find a consensus mosaic and to then selectively compute radiances in under- and over-exposed regions (Eden, Uyttendaele, and Szeliski 2006), as shown in Figure 10.17. Additional techniques for constructing and displaying high dynamic range video are discussed in Myszkowski, Mantiuk, and Krawczyk (2008), Tocci, Kiser *et al.* (2011), Sen, Kalantari *et al.* (2012), Dufaux, Le Callet *et al.* (2016), Banterle, Artusi *et al.* (2017), and Kalantari and Ramamoorthi (2017). Another approach is to use deep learning techniques to infer the high dynamic range radiance image from a single low dynamic range image (Liu, Lai *et al.* 2020b).

Some cameras, such as the Sony α550 and Pentax K-7, have started integrating multiple exposure merging and tone mapping directly into the camera body. In the future, the need to compute high dynamic range images from multiple exposures may be eliminated by advances in camera sensor technology (Yang, El Gamal *et al.* 1999; Nayar and Mitsunaga 2000; Nayar

and Branzoi 2003; Kang, Uyttendaele *et al.* 2003; Narasimhan and Nayar 2005; Tumblin, Agrawal, and Raskar 2005). However, the need to blend such images and to tone map them to lower-gamut displays is likely to remain.

HDR image formats. Before we discuss techniques for mapping HDR images back to a displayable gamut, we should discuss the commonly used formats for storing HDR images.

If storage space is not an issue, storing each of the R, G, and B values as a 32-bit IEEE float is the best solution. The commonly used Portable PixMap (.ppm) format, which supports both uncompressed ASCII and raw binary encodings of values, can be extended to a Portable FloatMap (.pfm) format by modifying the header. TIFF also supports full floating point values.

A more compact representation is the Radiance format (.pic, .hdr) (Ward 1994), which uses a single common exponent and per-channel mantissas. An intermediate encoding, OpenEXR from ILM,¹⁴ uses 16-bit floats for each channel, which is a format supported natively on most modern GPUs. Ward (2004) describes these and other data formats such as LogLuv (Larson 1998) in more detail, as do the books by Freeman (2008) and Reinhard, Heidrich *et al.* (2010). An even more recent HDR image format is the JPEG XR standard.

10.2.1 Tone mapping

Once a radiance map has been computed, it is usually necessary to display it on a lower gamut (i.e., eight-bit) screen or printer. A variety of *tone mapping* techniques has been developed for this purpose, which involve either computing spatially varying transfer functions or reducing image gradients to fit the available dynamic range (Reinhard, Heidrich *et al.* 2010).

The simplest way to compress a high dynamic range radiance image into a low dynamic range gamut is to use a global transfer curve (Larson, Rushmeier, and Piatko 1997). Figure 10.18 shows one such example, where a gamma curve is used to map an HDR image back into a displayable gamut. If gamma is applied separately to each channel (Figure 10.18b), the colors become muted (less saturated), as higher-valued color channels contribute less (proportionately) to the final color. Extracting the luminance channel from the color image using (2.104), applying the global mapping to the luminance channel, and then reconstituting the color image using (10.19) works better (Figure 10.18c).

Unfortunately, when the image has a really wide range of exposures, this global approach still fails to preserve details in regions with widely varying exposures. What is needed, instead, is something akin to the dodging and burning performed by photographers in the dark-

¹⁴<https://www.openexr.net>.



Figure 10.18 Global tone mapping: (a) input HDR image, linearly mapped; (b) gamma applied to each color channel independently; (c) gamma applied to intensity (colors are less washed out). Original HDR image courtesy of Paul Debevec, <https://www.pauldebevec.com/Research/HDR>. Processed images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

room. Mathematically, this is similar to dividing each pixel by the *average* brightness in a region around that pixel.

Figure 10.19 shows how this process works. As before, the image is split into its luminance and chrominance channels. The log luminance image

$$H(x, y) = \log L(x, y) \quad (10.11)$$

is then low-pass filtered to produce a *base layer*

$$H_L(x, y) = B(x, y) * H(x, y), \quad (10.12)$$

and a high-pass *detail layer*

$$H_H(x, y) = H(x, y) - H_L(x, y). \quad (10.13)$$

The base layer is then contrast reduced by scaling to the desired log-luminance range,

$$H'_H(x, y) = s H_H(x, y) \quad (10.14)$$

and added to the detail layer to produce the new log-luminance image

$$I(x, y) = H'_H(x, y) + H_L(x, y), \quad (10.15)$$

which can then be exponentiated to produce the tone-mapped (compressed) luminance image. Note that this process is equivalent to dividing each luminance value by (a monotonic mapping of) the average log-luminance value in a region around that pixel.

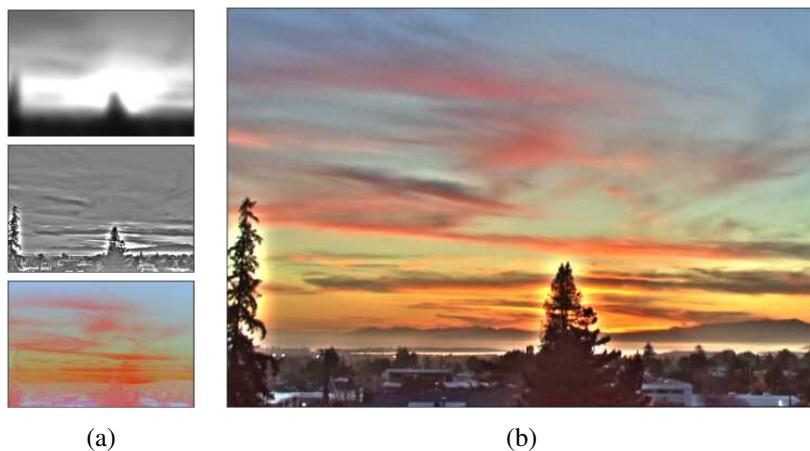


Figure 10.19 Local tone mapping using linear filters: (a) low-pass and high-pass filtered log luminance images and color (chrominance) image; (b) resulting tone-mapped image (after attenuating the low-pass log luminance image) shows visible halos around the trees. Processed images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.



Figure 10.20 Local tone mapping using a bilateral filter (Durand and Dorsey 2002): (a) low-pass and high-pass bilateral filtered log luminance images and color (chrominance) image; (b) resulting tone-mapped image (after attenuating the low-pass log luminance image) shows no halos. Processed images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

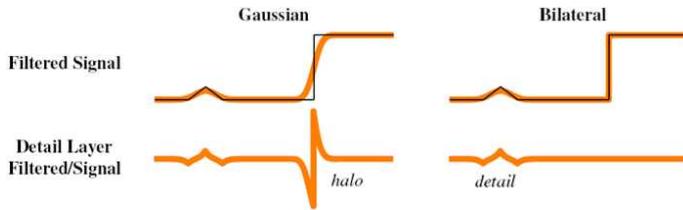


Figure 10.21 Gaussian vs. bilateral filtering (Petschnigg, Agrawala et al. 2004) © 2004 ACM: A Gaussian low-pass filter blurs across all edges and therefore creates strong peaks and valleys in the detail image that cause halos. The bilateral filter does not smooth across strong edges and thereby reduces halos while still capturing detail.

Figure 10.19 shows the low-pass and high-pass log luminance image and the resulting tone-mapped color image. Note how the detail layer has visible *halos* around the high-contrast edges, which are visible in the final tone-mapped image. This is because linear filtering, which is not edge preserving, produces halos in the detail layer (Figure 10.21).

The solution to this problem is to use an edge-preserving filter to create the base layer. Durand and Dorsey (2002) study a number of such edge-preserving filters, including anisotropic and robust anisotropic diffusion, and select bilateral filtering (Section 3.3.1) as their edge-preserving filter. (The paper by Farbman, Fattal et al. (2008) argues in favor of using a weighted least squares (WLF) filter as an alternative to the bilateral filter and Paris, Kornprobst et al. (2008) reviews bilateral filtering and its applications in computer vision and computational photography.) Figure 10.20 shows how replacing the linear low-pass filter with a bilateral filter produces tone-mapped images with no visible halos. Figure 10.22 summarizes the complete information flow in this process, starting with the decomposition into log luminance and chrominance images, bilateral filtering, contrast reduction, and re-composition into the final output image.

An alternative to compressing the base layer is to compress its *derivatives*, i.e., the gradient of the log-luminance image (Fattal, Lischinski, and Werman 2002). Figure 10.23 illustrates this process. The log-luminance image is differentiated to obtain a gradient image

$$H'(x, y) = \nabla H(x, y). \quad (10.16)$$

This gradient image is then attenuated by a spatially varying attenuation function $\Phi(x, y)$,

$$G(x, y) = H'(x, y) \Phi(x, y). \quad (10.17)$$

The attenuation function $I(x, y)$ is designed to attenuate large-scale brightness changes (Figure 10.24a) and is designed to take into account gradients at different spatial scales (Fattal,

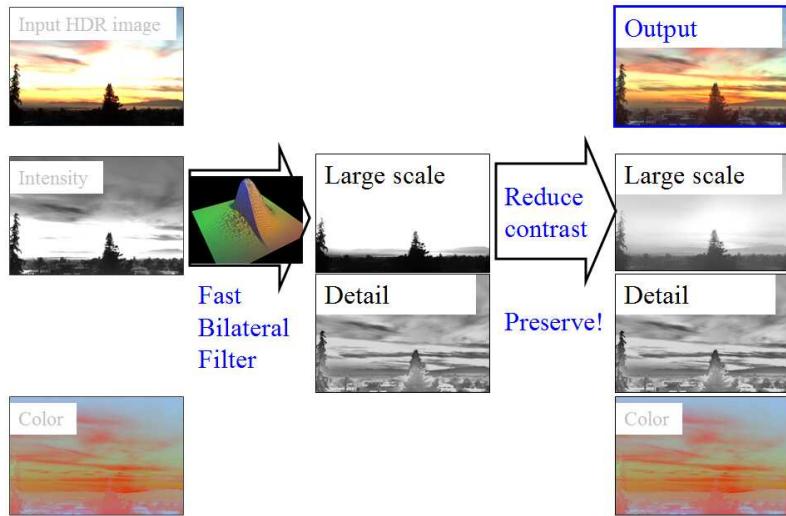


Figure 10.22 Local tone mapping using a bilateral filter (Durand and Dorsey 2002): summary of algorithm workflow. Images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

Lischinski, and Werman 2002).

After attenuation, the resulting gradient field is re-integrated by solving a first-order variational (least squares) problem,

$$\min \int \int \|\nabla I(x, y) - G(x, y)\|^2 dx dy \quad (10.18)$$

to obtain the compressed log-luminance image $I(x, y)$. This least squares problem is the same that was used for Poisson blending (Section 8.4.4) and was first introduced in our study of regularization (Section 4.2, 4.24). It can efficiently be solved using techniques such as multigrid and hierarchical basis preconditioning (Fattal, Lischinski, and Werman 2002; Szeliski 2006b; Farbman, Fattal *et al.* 2008; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013). Once the new luminance image has been computed, it is combined with the original color image using

$$C_{\text{out}} = \left(\frac{C_{\text{in}}}{L_{\text{in}}} \right)^s L_{\text{out}}, \quad (10.19)$$

where $C = (R, G, B)$ and L_{in} and L_{out} are the original and compressed luminance images. The exponent s controls the saturation of the colors and is typically in the range $s \in [0.4, 0.6]$ (Fattal, Lischinski, and Werman 2002). Figure 10.24b shows the final tone-mapped color

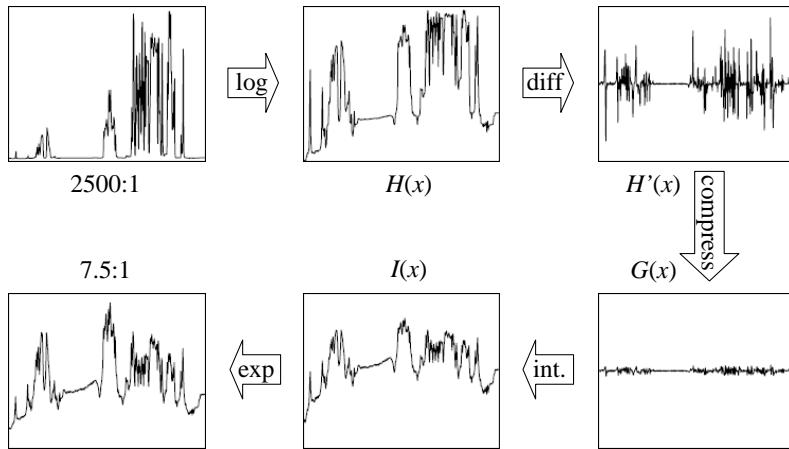


Figure 10.23 Gradient domain tone mapping (Fattal, Lischinski, and Werman 2002) © 2002 ACM. The original image with a dynamic range of 2415:1 is first converted into the log domain, $H(x)$, and its gradients are computed, $H'(x)$. These are attenuated (compressed) based on local contrast, $G(x)$, and integrated to produce the new logarithmic exposure image $I(x)$, which is exponentiated to produce the final intensity image, whose dynamic range is 7.5:1.

image, which shows no visible halos despite the extremely large variation in input radiance values.

Yet another alternative to these two approaches is to perform the local dodging and burning using a locally scale-selective operator (Reinhard, Stark *et al.* 2002). Figure 10.25 shows how such a scale selection operator can determine a radius (scale) that only includes similar color values within the inner circle while avoiding much brighter values in the surrounding circle. In practice, a difference of Gaussians normalized by the inner Gaussian response is evaluated over a range of scales, and the largest scale whose metric is below a threshold is selected (Reinhard, Stark *et al.* 2002).

Another recently developed approach to tone mapping based on multi-resolution decomposition is the Local Laplacian Filter (Paris, Hasinoff, and Kautz 2011), which we introduced in Section 3.5.3. Coefficients in a Laplacian pyramid are constructed from locally contrast-adjusted patches, which enables the technique to not only tone map HDR images, but also to enhance local details and do style transfer (Aubry, Paris *et al.* 2014).

What all of these techniques have in common is that they adaptively attenuate or brighten different regions of the image so that they can be displayed in a limited gamut without loss of contrast. Lischinski, Farbman *et al.* (2006) introduce an *interactive* technique that performs



Figure 10.24 Gradient domain tone mapping (Fattal, Lischinski, and Werman 2002) © 2002 ACM: (a) attenuation map, with darker values corresponding to more attenuation; (b) final tone-mapped image.

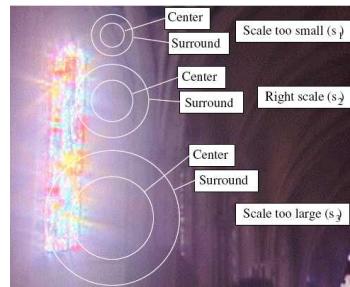


Figure 10.25 Scale selection for tone mapping (Reinhard, Stark et al. 2002) © 2002 ACM.



Figure 10.26 Interactive local tone mapping (Lischinski, Farbman et al. 2006) © 2006 ACM: (a) user-drawn strokes with associated exposure values $g(x, y)$; (b) corresponding piecewise-smooth exposure adjustment map $f(x, y)$.

this operation by interpolating a set of sparse user-drawn adjustments (strokes and associated exposure value corrections) to a piecewise-continuous exposure correction map (Figure 10.26). The interpolation is performed by minimizing a locally weighted least squares (WLS) variational problem,

$$\min \int \int w_d(x, y) \|f(x, y) - g(x, y)\|^2 dx dy + \lambda \int \int w_s(x, y) \|\nabla f(x, y)\|^2 dx dy, \quad (10.20)$$

where $g(x, y)$ and $f(x, y)$ are the input and output log exposure (attenuation) maps (Figure 10.26). The data weighting term $w_d(x, y)$ is 1 at stroke locations and 0 elsewhere. The smoothness weighting term $w_s(x, y)$ is inversely proportional to the log-luminance gradient,

$$w_s = \frac{1}{\|\nabla H\|^\alpha + \epsilon} \quad (10.21)$$

and hence encourages the $f(x, y)$ map to be smoother in low-gradient areas than along high-gradient discontinuities.¹⁵ The same approach can also be used for fully automated tone mapping by setting target exposure values at each pixel and allowing the weighted least squares to convert these into piecewise smooth adjustment maps.

The weighted least squares algorithm, which was originally developed for image colortization applications (Levin, Lischinski, and Weiss 2004), has since been applied to general edge-preserving smoothing in applications such as contrast enhancement (Bae, Paris, and Durand 2006) and tone mapping (Farbman, Fattal *et al.* 2008) where the bilateral filtering was previously used. It can also be used to perform HDR merging and tone mapping simultaneously (Raman and Chaudhuri 2007, 2009).

Given the wide range of locally adaptive tone mapping algorithms that have been developed, which ones should be used in practice? Freeman (2008) provides a great discussion of commercially available algorithms, their artifacts, and the parameters that can be used to control them. He also has a wealth of tips for HDR photography and workflow. I highly recommend his book for anyone contemplating additional research (or personal photography) in this area.

10.2.2 Application: Flash photography

While high dynamic range imaging combines images of a scene taken at different exposures, it is also possible to combine flash and non-flash images to achieve better exposure and color balance and to reduce noise (Eisemann and Durand 2004; Petschnigg, Agrawala *et al.* 2004).

¹⁵In practice, the x and y discrete derivatives are weighted separately (Lischinski, Farbman *et al.* 2006). Their default parameter settings are $\lambda = 0.2$, $\alpha = 1$, and $\epsilon = 0.0001$.

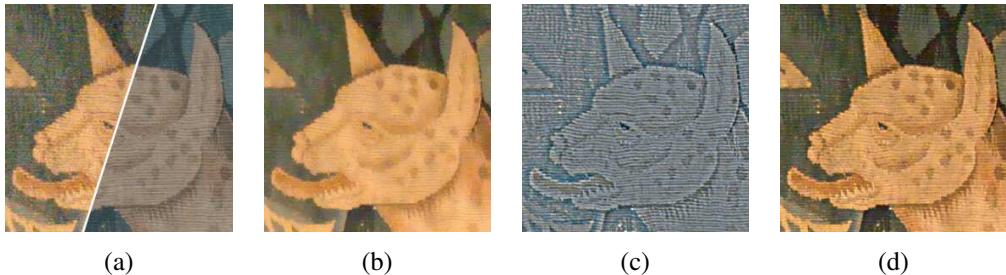


Figure 10.27 *Detail transfer in flash/no-flash photography (Petschnigg, Agrawala et al. 2004) © 2004 ACM:* (a) details of input ambient A and flash F images; (b) joint bilaterally filtered no-flash image A^{NR} ; (c) detail layer F^{Detail} computed from the flash image F ; (d) final merged image A^{Final} .

The problem with flash images is that the color is often unnatural (it fails to capture the ambient illumination), there may be strong shadows or specularities, and there is a radial falloff in brightness away from the camera (Figures 10.1b and 10.27a). Non-flash photos taken under low light conditions often suffer from excessive noise (because of the high ISO gains and low photon counts) and blur (due to longer exposures). Is there some way to combine a non-flash photo taken just before the flash goes off with the flash photo to produce an image with good color values, sharpness, and low noise? In fact, the discontinued FujiFilm FinePix F40fd camera takes a pair of flash and no flash images in quick succession; however, it only lets you decide to keep one of them.

Petschnigg, Agrawala *et al.* (2004) approach this problem by first filtering the no-flash (ambient) image A with a variant of the bilateral filter called the *joint bilateral filter*¹⁶ in which the range kernel (3.36)

$$r(i, j, k, l) = \exp\left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right) \quad (10.22)$$

is evaluated on the flash image F instead of the ambient image A , as the flash image is less noisy and hence has more reliable edges (Figure 10.27b). Because the contents of the flash image can be unreliable inside and at the boundaries of shadows and specularities, these are detected and a regular bilaterally filtered image A^{Base} is used instead (Figure 10.28).

The second stage of their algorithm computes a flash detail image

$$F^{Detail} = \frac{F + \epsilon}{F^{Base} + \epsilon}, \quad (10.23)$$

¹⁶Eisemann and Durand (2004) call this the *cross bilateral filter*.

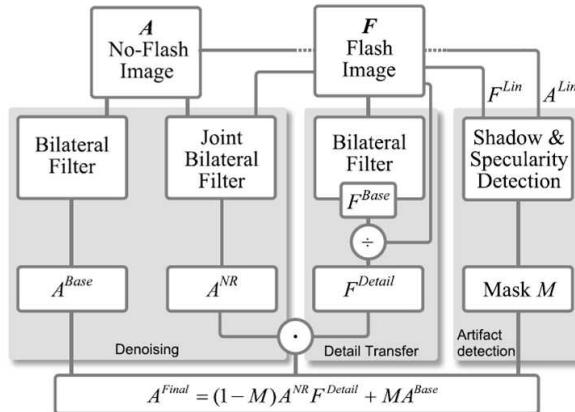


Figure 10.28 Flash/no-flash photography algorithm (Petschnigg, Agrawala et al. 2004) © 2004 ACM. The ambient (no-flash) image A is filtered with a regular bilateral filter to produce A^{Base} , which is used in shadow and specularity regions, and a joint bilaterally filtered noise reduced image A^{NR} . The flash image F is bilaterally filtered to produce a base image F^{Base} and a detail (ratio) image F^{Detail} , which is used to modulate the denoised ambient image. The shadow/specularity mask M is computed by comparing linearized versions of the flash and no-flash images.

where F^{Base} is a bilaterally filtered version of the flash image F and $\epsilon = 0.02$. This detail image (Figure 10.27c) encodes details that may have been filtered away from the noise-reduced no-flash image A^{NR} , as well as additional details created by the flash camera, which often add crispness. The detail image is used to modulate the noise-reduced ambient image A^{NR} to produce the final results

$$A^{Final} = (1 - M)A^{NR}F^{Detail} + M A^{Base} \quad (10.24)$$

shown in Figures 10.1b and 10.27d.

Eisemann and Durand (2004) present an alternative algorithm that shares some of the same basic concepts. Both papers are well worth reading and contrasting (Exercise 10.6).

Flash images can also be used for a variety of additional applications such as extracting more reliable foreground mattes of objects (Raskar, Tan et al. 2004; Sun, Li et al. 2006). Given a large enough training set, it is also possible to decompose single flash images into their ambient and flash illumination components, which can be used to adjust their appearance (Aksoy, Kim et al. 2018). Flash photography is just one instance of the more general topic of *active illumination*, which is discussed in more detail by Raskar and Tumblin (2010) and Ikeuchi, Matsushita et al. (2020).

10.3 Super-resolution, denoising, and blur removal

While high dynamic range imaging enables us to obtain an image with a larger dynamic range than a single regular image, super-resolution enables us to create images with higher *spatial* resolution and less noise than regular camera images (Chaudhuri 2001; Park, Park, and Kang 2003; Capel and Zisserman 2003; Capel 2004; van Ouwerkerk 2006; Anwar, Khan, and Barnes 2020). Most commonly, super-resolution refers to the process of aligning and combining several input images to produce such high-resolution composites (Irani and Peleg 1991; Cheeseman, Kanefsky *et al.* 1993; Pickup, Capel *et al.* 2009; Wronski, Garcia-Dorado *et al.* 2019). However, some techniques can super-resolve a single image (Freeman, Jones, and Pasztor 2002; Baker and Kanade 2002; Fattal 2007; Anwar, Khan, and Barnes 2020) and are hence closely related to techniques for removing blur (Sections 3.4.1 and 3.4.2). Anwar, Khan, and Barnes (2020) provide a comprehensive review of single image super-resolution techniques with a particular focus on recent deep learning-based approaches.

A traditional way to formulate the super-resolution problem is to write down the stochastic image formation equations and image priors and to then use Bayesian inference to recover the super-resolved (original) sharp image. We can do this by generalizing the image formation equations used for image deblurring (Section 3.4.1), which we also used for blur kernel (PSF) estimation (Section 10.1.4). In this case, we have several observed images $\{o_k(\mathbf{x})\}$, as well as an image warping function $\hat{\mathbf{h}}_k(\mathbf{x})$ for each observed image (Figure 3.46). Combining all of these elements, we get the (noisy) observation equations¹⁷

$$o_k(\mathbf{x}) = D\{b(\mathbf{x}) * s(\hat{\mathbf{h}}_k(\mathbf{x}))\} + n_k(\mathbf{x}), \quad (10.25)$$

where D is the downsampling operator, which operates *after* the super-resolved (sharp) warped image $s(\hat{\mathbf{h}}_k(\mathbf{x}))$ has been convolved with the blur kernel $b(\mathbf{x})$. The above image formation equations lead to the following least squares problem,

$$\sum_k \|o_k(\mathbf{x}) - D\{b_k(\mathbf{x}) * s(\hat{\mathbf{h}}_k(\mathbf{x}))\}\|^2. \quad (10.26)$$

In most super-resolution algorithms, the alignment (warping) $\hat{\mathbf{h}}_k$ is estimated using one of the input frames as the *reference frame*; either feature-based (Section 8.1.3) or direct (image-based) (Section 9.2) parametric alignment techniques can be used. (A few algorithms, such as those described by Schultz and Stevenson (1996), Capel (2004), and Wronski, Garcia-Dorado *et al.* (2019) use dense (per-pixel flow) estimates.) A better approach is to re-compute

¹⁷It is also possible to add an unknown bias–gain term to each observation (Capel 2004), as was done for motion estimation in (9.8).

the alignment by directly minimizing (10.26) once an initial estimate of $s(\mathbf{x})$ has been computed (Hardie, Barnard, and Armstrong 1997) or to *marginalize* out the motion parameters altogether (Pickup, Capel *et al.* 2007).

The point spread function (blur kernel) b_k is either inferred from knowledge of the image formation process (e.g., the amount of motion or defocus blur and the camera sensor optics) or calibrated from a test image or the observed images $\{\mathbf{o}_k\}$ using one of the techniques described in Section 10.1.4. The problem of simultaneously inferring the blur kernel and the sharp image is known as *blind image deconvolution* (Kundur and Hatzinakos 1996; Levin 2006; Levin, Weiss *et al.* 2011; Campisi and Egiazarian 2017).¹⁸

Given an estimate of $\hat{\mathbf{h}}_k$ and $b_k(\mathbf{x})$, (10.26) can be re-written using matrix/vector notation as a large sparse least squares problem in the unknown values of the super-resolved pixels \mathbf{s} ,

$$\sum_k \|\mathbf{o}_k - \mathbf{DB}_k \mathbf{W}_k \mathbf{s}\|^2. \quad (10.27)$$

(Recall from (3.75) that once the warping function $\hat{\mathbf{h}}_k$ is known, values of $s(\hat{\mathbf{h}}_k(\mathbf{x}))$ depend linearly on those in $s(\mathbf{x})$.) An efficient way to solve this least squares problem is to use preconditioned conjugate gradient descent (Capel 2004), although some earlier algorithms, such as the one developed by Irani and Peleg (1991), used regular gradient descent (also known as iterative back projection (IBP) in the computed tomography literature).

The above formulation assumes that warping can be expressed as a simple (sinc or bicubic) interpolated resampling of the super-resolved sharp image, followed by a stationary (spatially invariant) blurring (PSF) and area integration process. However, if the surface is severely foreshortened, we have to take into account the spatially varying filtering that occurs during the image warping (Section 3.6.1), before we can then model the PSF induced by the optics and camera sensor (Wang, Kang *et al.* 2001; Capel 2004).

How well does this least squares (MLE) approach to super-resolution work? In practice, this depends a lot on the amount of blur and aliasing in the camera optics, as well as the accuracy in the motion and PSF estimates (Baker and Kanade 2002; Jiang, Wong, and Bao 2003; Capel 2004). Less blurring and more aliasing means that there is more (aliased) high frequency information available to be recovered. However, because the least squares (maximum likelihood) formulation uses no image prior, a lot of high-frequency noise can be introduced into the solution (Figure 10.29c).

For this reason, classic super-resolution algorithms assume some form of image prior. The simplest of these is to place a penalty on the image derivatives similar to Equations (4.29) and

¹⁸Notice that there is a chicken-and-egg problem if both the blur kernel and the super-resolved image are unknown. This can be “broken” either using structural assumptions about the sharp image, e.g., the presence of edges (Joshi, Szeliski, and Kriegman 2008) or prior models for the image, such as edge sparsity (Fergus, Singh *et al.* 2006).



Figure 10.29 Super-resolution results using a variety of image priors (Capel 2001): (a) Low-res ROI (bicubic $3 \times$ zoom); (b) average image; (c) MLE @ $1.25 \times$ pixel-zoom; (d) simple $\|x\|^2$ prior ($\lambda = 0.004$); (e) GMRF ($\lambda = 0.003$); (f) HMRF ($\lambda = 0.01$, $\alpha = 0.04$). 10 images are used as input and a $3 \times$ super-resolved image is produced in each case, except for the MLE result in (c).

(4.42), e.g.,

$$\sum_{(i,j)} \rho_p(s(i,j) - s(i+1,j)) + \rho_p(s(i,j) - s(i,j+1)). \quad (10.28)$$

As discussed in Section 4.3, when ρ_p is quadratic, this is a form of Tikhonov regularization (Section 4.2), and the overall problem is still linear least squares. The resulting prior image model is a Gaussian Markov random field (GMRF), which can be extended to other (e.g., diagonal) differences, as in Capel (2004) and Figure 10.29.

Unfortunately, GMRFs tend to produce solutions with visible ripples, which can also be interpreted as increased noise sensitivity in middle frequencies. A better image prior is a robust prior that encourages piecewise continuous solutions (Black and Rangarajan 1996), see Appendix B.3. Examples of such priors include the Huber potential (Schultz and Stevenson 1996; Capel and Zisserman 2003), which is a blend of a Gaussian with a longer-tailed Laplacian, and the even sparser (heavier-tailed) hyper-Laplacians used by Levin, Fergus *et al.* (2007) and Krishnan and Fergus (2009). It is also possible to learn the parameters for such priors using cross-validation (Capel 2004; Pickup 2007).

While sparse (robust) derivative priors can reduce rippling effects and increase edge

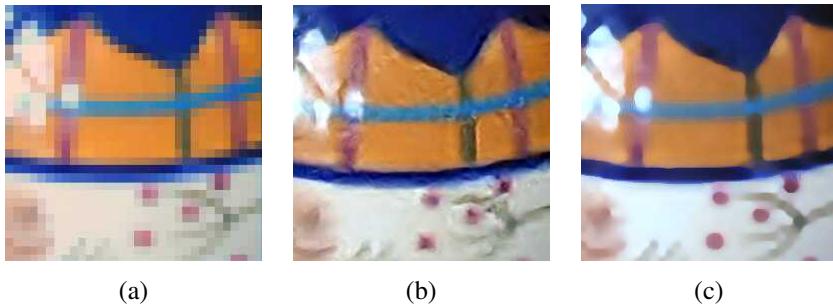


Figure 10.30 Example-based super-resolution: (a) original 32×32 low-resolution image; (b) example-based super-resolved 256×256 image (Freeman, Jones, and Pasztor 2002) © 2002 IEEE; (c) upsampling via imposed edge statistics (Fattal 2007) © 2007 ACM.

sharpness, they cannot *hallucinate* higher-frequency texture or details. To do this, a training set of sample images can be used to find plausible mappings between low-frequency originals and the missing higher frequencies. Inspired by some of the example-based texture synthesis algorithms we discuss in Section 10.5, the *example-based super-resolution* algorithm developed by Freeman, Jones, and Pasztor (2002) uses training images to *learn* the mapping between local texture patches and missing higher-frequency details. To ensure that overlapping patches are similar in appearance, a Markov random field is used and optimized using either belief propagation (Freeman, Pasztor, and Carmichael 2000) or a raster-scan deterministic variant (Freeman, Jones, and Pasztor 2002). Figure 10.30 shows the results of hallucinating missing details using this approach and compares these results to a more recent algorithm by Fattal (2007). This latter algorithm learns to predict oriented gradient magnitudes in the finer resolution image based on a pixel’s location relative to the nearest detected edge along with the corresponding edge statistics (magnitude and width). It is also possible to combine sparse (robust) derivative priors with example-based super-resolution, as shown by Tappen, Russell, and Freeman (2003).

An alternative (but closely related) form of hallucination is to *recognize* the parts of a training database of images to which a low-resolution pixel might correspond. In their work, Baker and Kanade (2002) use local derivative-of-Gaussian filter responses as features and then match *parent structure* vectors in a manner similar to De Bonet (1997).¹⁹ The high-frequency gradient at each recognized training image location is then used as a constraint on the super-resolved image, along with the usual reconstruction (prediction) Equation (10.26).

¹⁹For face super-resolution, where all the images are pre-aligned, only corresponding pixels in different images are examined.

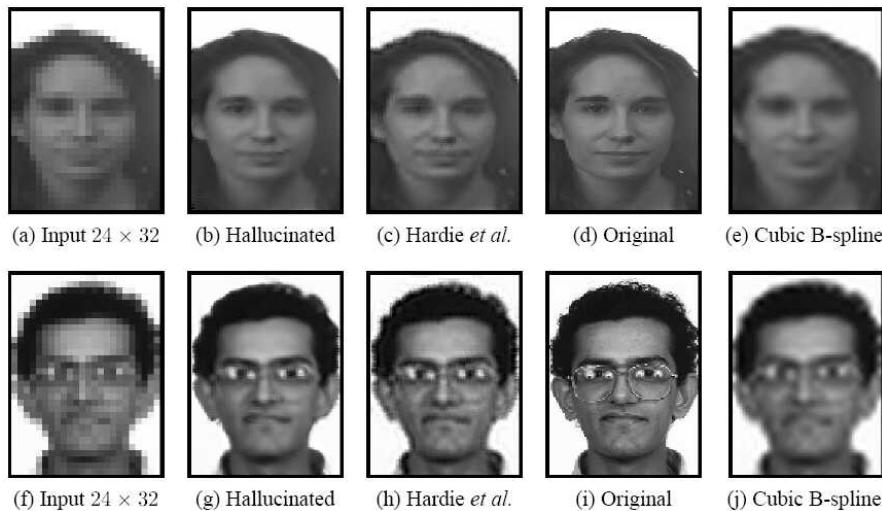


Figure 10.31 *Recognition-based super-resolution (Baker and Kanade 2002)* © 2002 IEEE. The Hallucinated column shows the results of the recognition-based algorithm compared to the regularization-based approach of Hardie, Barnard, and Armstrong (1997).

Figure 10.31 shows the result of hallucinating higher-resolution faces from lower-resolution inputs; Baker and Kanade (2002) also show examples of super-resolving known-font text. Exercise 10.7 gives more details on how to implement and test one or more of these super-resolution techniques.

The latest trend in super-resolution has been the use of deep neural networks to directly predict super-resolved images. This approach, which began with the seminal work of Dong, Loy *et al.* (2016), has generated dozens of different DNNs and architectures, including the Deep Learning Super Sampling hardware embedded in the latest NVIDIA graphics cards (Burnes 2020). The recent survey on single-image super-resolution by Anwar, Khan, and Barnes (2020) categorizes these algorithms into a taxonomy (Figure 10.32a), provides a pictorial summary network architectures (Figure 10.32b), and compares the super-resolution results both numerically and visually on noise-free known bicubic-kernel decimation image datasets. While the results shown in Figure 10.33 show dramatic differences between algorithms, it is not clear how well these algorithms generalize to real-world noisy input with unknown blur kernels. The RealSR real-world super-resolution dataset developed by (Cai, Zeng *et al.* 2019), shot using a zoom lens on a digital camera, provides a means to test (and train) algorithms on real imaging degradations. This dataset forms the basis for the NTIRE

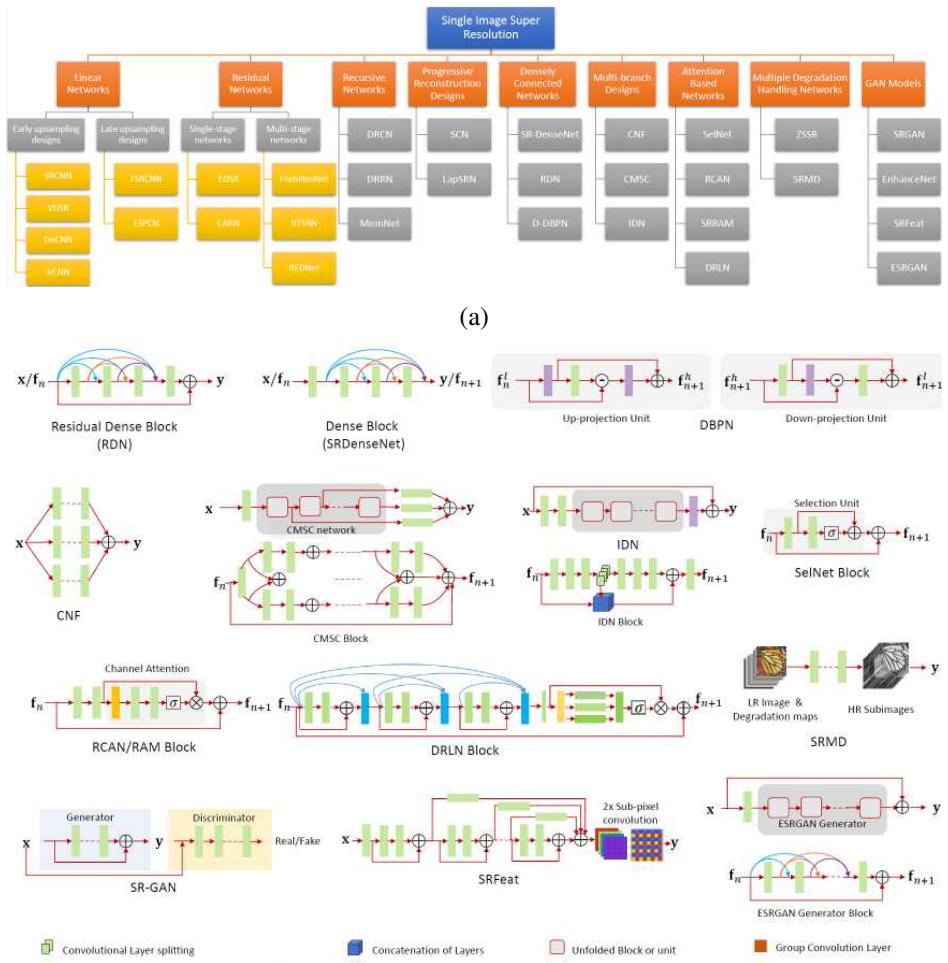


Figure 10.32 Recent deep neural network algorithms for single image super-resolution (Anwar, Khan, and Barnes 2020) © 2020 ACM: (a) a taxonomy of the algorithms based on their general approach; (b) schematic architectures for a subset of the algorithms.

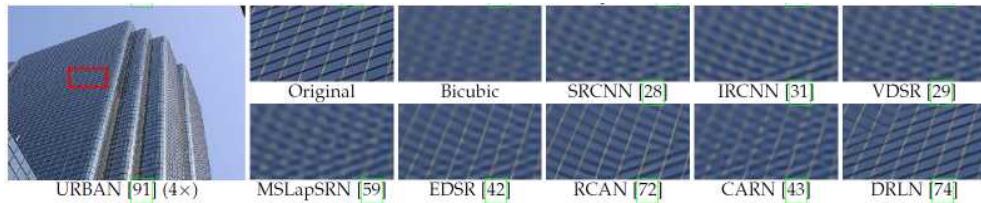


Figure 10.33 Visual comparison of some super-resolution algorithms (Anwar, Khan, and Barnes 2020) © 2020 ACM.

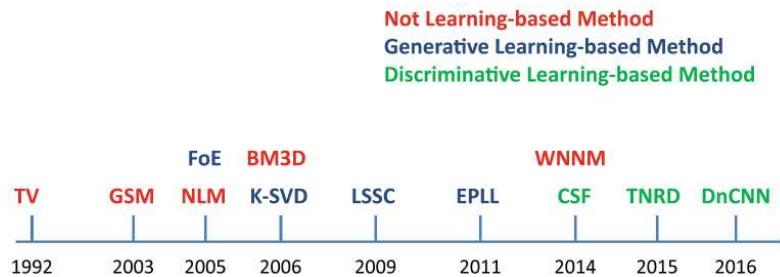


Figure 10.34 Timeline of denoising algorithms from Gu and Timofte (2019) © 2019 Springer.

challenges on real image super-resolution (Cai, Gu *et al.* 2019),²⁰ which provide empirical comparisons of recent deep network-based algorithms.

While single-image super-resolution is interesting, much more impressive (and practical) results can be obtained by building a multi-frame super-resolution algorithm directly into a smartphone camera, where the processing can be done jointly with the image demosaicing. We discuss recent work by Wronski, Garcia-Dorado *et al.* (2019) in Section 10.3.1 and Figure 10.38 on color image demosaicing. It is also possible to upsample videos temporally using frame interpolation (Section 9.4.1), spatially using video super-resolution (Liu and Sun 2013; Kappeler, Yoo *et al.* 2016; Shi, Caballero *et al.* 2016; Tao, Gao *et al.* 2017; Nah, Timofte *et al.* 2019; Isobe, Jia *et al.* 2020; Li, Tao *et al.* 2020), or simultaneously in both the spatial and temporal dimensions (Kang, Jo *et al.* 2020).

Single and multi-frame denoising

Image denoising is one of the classic problems in image processing and computer vision (Perona and Malik 1990b; Rudin, Osher, and Fatemi 1992; Buades, Coll, and Morel 2005b). Over the last four decades, hundreds of algorithms have been developed, and the field continues to be actively studied, with recent algorithms all being based on deep neural networks.

The latest benchmark for comparing image denoising algorithms, the NTIRE 2020 Challenge on Real Image Denoising (Abdelhamed, Afifi *et al.* 2020), is based on a smartphone image denoising dataset (SIDD) (Abdelhamed, Lin, and Brown 2018), where the noise-free ground truth images were obtained by averaging sets of 150 noisy images. This provides much more realistic and varied real-world noise and image processing models than the synthetically noised images used in most previous benchmarks (with the exception of (Plötz and Roth 2017)).

A recent (brief) survey on image denoising by Gu and Timofte (2019) includes the following seminal denoising papers²¹ (see Figure 10.34 for a timeline):

- total variation (TV) (Rudin, Osher, and Fatemi 1992; Chan, Osher, and Shen 2001; Chambolle 2004; Chan and Shen 2005),
- Gaussian scale mixtures (GSMs) (Lyu and Simoncelli 2009),
- Field of Experts (FoE) (Roth and Black 2009),
- non-local means (NLM) (Buades, Coll, and Morel 2005a,b),
- BM3D (Dabov, Foi *et al.* 2007),
- sparse overcomplete dictionaries (K-SVD) (Aharon, Elad, and Bruckstein 2006),
- expected patch log likelihood (EPLL) (Zoran and Weiss 2011),
- an MLP denoiser (Burger, Schuler, and Harmeling 2012),
- weighted nuclear norm minimization (WNNM) (Gu, Zhang *et al.* 2014),
- shrinkage fields (CSF) (Schmidt and Roth 2014),
- Trainable Nonlinear Reaction Diffusion (TNRD) (Chen and Pock 2016),
- a cross-channel noise model for color images (Nam, Hwang *et al.* 2016),

²⁰<https://data.vision.ee.ethz.ch/cvl/ntire20/>, <https://data.vision.ee.ethz.ch/cvl/aim20/>

²¹I have added a few more papers from the ICCV tutorial by Brown (2019) and a few additional recommendations from Abdelrahman Abdelhamed.

- a denoising residual CNN (DnCNN) (Zhang, Zuo *et al.* 2017), which is now considered the baseline for DNN denoising, and
- learning to see in the dark (Chen, Chen *et al.* 2018).

While these results show dramatic improvement over time, today’s imaging sensors for the most part produce relatively clean images, except in low-light situations, where the ISO camera gain must be increased and the read and photon noise become comparable to the signal strength. In this regime, it is preferable, if possible, to take a rapid burst of images at low ISO (gain) and then combine these to obtain a denoised image (Hasinoff, Kutulakos *et al.* 2009; Hasinoff, Durand, and Freeman 2010; Liu, Yuan *et al.* 2014). This approach was generalized and applied to low-light photography in the HDR+ system of Hasinoff, Sharlet *et al.* (2016). More recent work along these lines, some of which combines low-light photography, demosaicing, and in some cases super-resolution, includes papers by Godard, Matzen, and Uyttendaele (2018), Chen, Chen *et al.* (2018), Mildenhall, Barron *et al.* (2018), Wronski, Garcia-Dorado *et al.* (2019), and (Rong, Demandolx *et al.* 2020). Liba, Murthy *et al.* (2019) describe the technology that underlies Google’s *Night Sight* feature, which not only robustly aligns and merges different moving regions together under noisy conditions, but also introduces the concept of “motion metering” to determine the optimal number of frames and exposure times.

Blur removal

Under favorable conditions, super-resolution and related upsampling techniques can increase the resolution of a well-photographed image or image collection. When the input images are blurry to start with, the best one can often hope for is to reduce the amount of blur. This problem is closely related to super-resolution, with the biggest differences being that the blur kernel b is usually much larger (and unknown) and the downsampling factor D is unity.

A large literature on image deblurring exists; some publications with nice literature reviews include those by Fergus, Singh *et al.* (2006), Yuan, Sun *et al.* (2008), and Joshi, Zitnick *et al.* (2009). It is also possible to reduce blur by combining sharp (but noisy) images with blurrier (but cleaner) images (Yuan, Sun *et al.* 2007), take lots of quick exposures (Hasinoff and Kutulakos 2011; Hasinoff, Kutulakos *et al.* 2009; Hasinoff, Durand, and Freeman 2010), or use *coded aperture* techniques to simultaneously estimate depth and reduce blur (Levin, Fergus *et al.* 2007; Zhou, Lin, and Nayar 2009). When available, data from on-board IMUs (inertial measurement units) can be used for blur kernel determination (Joshi, Kang *et al.* 2010). It is also possible to use information from dual-pixel sensors to aid the deblurring of misfocused images (Abuolaim and Brown 2020).

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

(a)

rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

(b)

Figure 10.35 *Bayer RGB pattern: (a) color filter array layout; (b) interpolated pixel values, with unknown (guessed) values shown as lower case.*

The past decade has seen the introductions of a large number of new learning-based de-blurring algorithms (Sun, Cao *et al.* 2015; Schuler, Hirsch *et al.* 2016; Nah, Hyun Kim, and Mu Lee 2017; Kupyn, Budzan *et al.* 2018; Tao, Gao *et al.* 2018; Zhang, Dai *et al.* 2019; Kupyn, Martyniuk *et al.* 2019). There has also been some work on artificially re-introducing texture in deblurred images to better match the expected image statistics (Cho, Joshi *et al.* 2012), i.e., what is now commonly called *perceptual loss* (Section 5.3.4).

10.3.1 Color image demosaicing

A special case of super-resolution, which is used daily in most digital still cameras, is the process of *demoslicing* samples from a color filter array (CFA) into a full-color RGB image. Figure 10.35 shows the most commonly used CFA known as the *Bayer pattern*, which has twice as many green (G) sensors as red and blue sensors.

The process of going from the known CFA pixels values to the full RGB image is quite challenging. Unlike regular super-resolution, where small errors in guessing unknown values usually show up as blur or aliasing, demosaicing artifacts often produce spurious colors or high-frequency patterned *zippering*, which are quite visible to the eye (Figure 10.36b).

Over the years, a variety of techniques have been developed for image demosaicing (Kimmel 1999). Longere, Delahunt *et al.* (2002), Tappen, Russell, and Freeman (2003), and Li, Gunturk, and Zhang (2008) provide surveys of the field as well as comparisons of previously developed techniques using perceptually motivated metrics. To reduce the zipper effect, most techniques use the edge or gradient information from the green channel, which is more reliable because it is sampled more densely, to infer plausible values for the red and blue channels, which are more sparsely sampled.

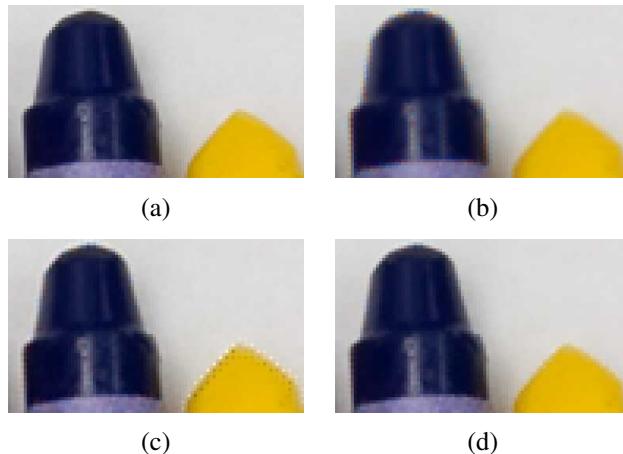


Figure 10.36 CFA demosaicing results (Bennett, Uyttendaele et al. 2006) © 2006 Springer: (a) original full-resolution image (a color subsampled version is used as the input to the algorithms); (b) bilinear interpolation results, showing color fringing near the tip of the blue crayon and zippering near its left (vertical) edge; (c) the high-quality linear interpolation results of Malvar, He, and Cutler (2004) (note the strong halo/checkerboard artifacts on the yellow crayon); (d) using the local two-color prior of Bennett, Uyttendaele et al. (2006).

To reduce color fringing, some techniques perform a color space analysis, e.g., using median filtering on color opponent channels (Longere, Delahunt et al. 2002). The approach of Bennett, Uyttendaele et al. (2006) computes local two-color models from an initial demosaicing result, using a moving 5×5 window to find the two dominant colors (Figure 10.37).²²

Once the local color model has been estimated at each pixel, a Bayesian approach is then used to encourage pixel values to lie along each color line and to cluster around the dominant color values, which reduces halos (Figure 10.36d). The Bayesian approach also supports the simultaneous application of demosaicing, denoising, and super-resolution, i.e., multiple CFA inputs can be merged into a higher-quality full-color image. More recent work that combines demosaicing and denoising includes papers by Chatterjee, Joshi et al. (2011) and Gharbi, Chaurasia et al. (2016). The NTIRE 2020 Challenge on Real Image Denoising (Abdelhamed, Afifi et al. 2020) includes a track on denoising RAW (i.e., color filter array) images. There's also an interesting paper by Jin, Facciolo, and Morel (2020) studying whether

²²Previous work on locally linear color models (Klinker, Shafer, and Kanade 1990; Omer and Werman 2004) focuses on color and illumination variation within a single material, whereas Bennett, Uyttendaele et al. (2006) use the two-color model to describe variations across color (material) edges.

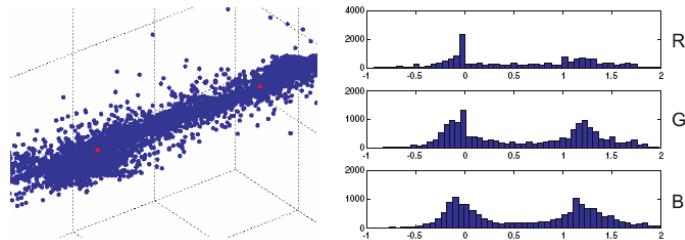


Figure 10.37 Two-color model computed from a collection of local 5×5 neighborhoods (Bennett, Uyttendaele et al. 2006) © 2006 Springer. After two-means clustering and reprojection along the line joining the two dominant colors (red dots), the majority of the pixels fall near the fitted line. The distribution along the line, projected along the RGB axes, is peaked at 0 and 1, the two dominant colors.

denoising should be applied before or after demosaicing.

As we mentioned before, burst photography (Cohen and Szeliski 2006; Hasinoff, Kutulakos et al. 2009; Hasinoff and Kutulakos 2011), i.e., the combination of rapidly acquired sequences of images, is becoming ubiquitous in smartphone cameras. A wonderful example of a recent system that performs joint demosaicing and multi-frame super-resolutions, based on locally adapted kernel functions (Figure 10.38), is the paper by Wronski, Garcia-Dorado et al. (2019), which underlies the *Super Res Zoom* feature in Google’s Pixel smartphones.

10.3.2 Lens blur (bokeh)

The ability to create a shallow depth-of-field photograph using a large aperture (Section 2.2.3) has always been one of the advantages of large-format, e.g., single lens reflex (SLR), cameras. The desire to artificially simulate refocusable, shallow depth-of-field cameras was one of the driving impetuses behind computational photography (Levoy 2006) and led to the development of lightfield cameras (Ng, Levoy et al. 2005), which we discuss in Section 14.3.4. Although some commercial models, such as the Lytro, were produced, the ability to create such images with smartphone cameras has only recently become widespread.²³

The Apple iPhone 7 Plus with its dual (wide/telephoto) lens was the first smartphone to introduce this feature, which they called the *Portrait mode*. Although the technical details behind this feature have never been published, the algorithm that estimates the depth image (which can be read out of the metadata in the portrait images) probably uses some combi-

²³An earlier feature called Google Lens Blur, which required moving the camera in a pattern, <https://ai.googleblog.com/2014/04/lens-blur-in-new-google-camera-app.html>, was never widely used.

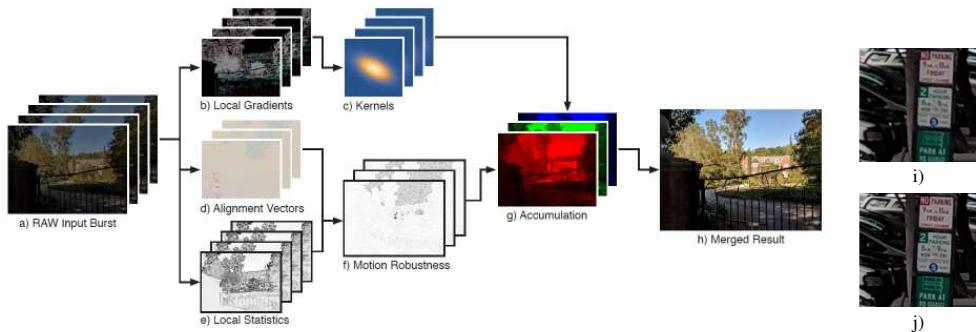


Figure 10.38 Hand-held multi-frame super-resolution (Wronski, Garcia-Dorado et al. 2019) © 2019 ACM. Processing pipeline, showing: (a) the captured burst of raw (Bayer CFA) images; (b) local gradients used to compute oriented kernels (c); (d) motion estimates, combined with local statistics (e) to compute blend weights (f). Results from (i) the previous method of Hasinoff, Sharlet et al. (2016) and (j) Wronski, Garcia-Dorado et al. (2019).

nation of stereo matching and deep learning. A little later, Google released its own Portrait Mode, which uses the dual pixels, originally designed for focusing the camera optics, along with person segmentation to compute a depth map, as described in the paper by Wadhwa, Garg *et al.* (2018). Once the depth map has been estimated, a fast approximation to a back-to-front blurred *over* compositing operator is used to correctly blur the background without including foreground colors. More recently Garg, Wadhwa *et al.* (2019) have improved the quality of the depth estimation using a deep network, and also used two lenses (along with dual pixels) to produce even higher-quality depth maps (Zhang, Wadhwa *et al.* 2020).

One final word on *bokeh*, which is the term photographers use to describe the shape of the glints or highlights that appear in an image. This shape is determined by the configuration of the *aperture blades* that control how much light enters the lens (on larger-format cameras). Traditionally, these were made with straight metal leaves, which resulted in polygonal apertures, but they were then mostly replaced by curved leaves to produce a more circular shape. When using computational photography, we can use whatever shape is pleasing to the photographer, but preferably *not* a Gaussian blur, which does not correspond to any real aperture and produces indistinct highlights. The paper by Wadhwa, Garg *et al.* (2018) uses a circular bokeh for their depth-of-field effect and a more recent version performs the computations in the HDR (radiance) space to produce more accurate highlights.²⁴

²⁴<https://ai.googleblog.com/2019/12/improvements-to-portrait-mode-on-google.html>

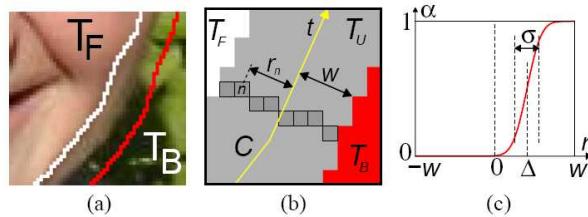


Figure 10.39 Softening a hard segmentation boundary (border matting) (Rother, Kolmogorov, and Blake 2004) © 2004 ACM: (a) the region surrounding a segmentation boundary where pixels of mixed foreground and background colors are visible; (b) pixel values along the boundary are used to compute a soft alpha matte; (c) at each point along the curve t , a displacement Δ and a width σ are estimated.

10.4 Image matting and compositing

Image matting and compositing is the process of cutting a foreground object out of one image and pasting it against a new background (Smith and Blinn 1996; Wang and Cohen 2009). It is commonly used in television and film production to composite a live actor in front of computer-generated imagery such as weather maps or 3D virtual characters and scenery (Wright 2006; Brinkmann 2008), and it has recently become a popular feature in video conferencing systems.

We have already seen a number of tools for interactively segmenting objects in an image, including snakes (Section 7.3.1), scissors (Section 7.3.1), and GrabCut segmentation (Section 4.3.2). While these techniques can generate reasonable pixel-accurate segmentations, they fail to capture the subtle interplay of foreground and background colors at *mixed pixels* along the boundary (Szeliski and Golland 1999) (Figure 10.39a).

To successfully copy a foreground object from one image to another without visible discretization artifacts, we need to *pull a matte*, i.e., to estimate a soft opacity channel α and the uncontaminated foreground colors F from the input composite image C . Recall from Section 3.1.3 (Figure 3.4) that the compositing equation (3.8) can be written as

$$C = (1 - \alpha)B + \alpha F. \quad (10.29)$$

This operator attenuates the influence of the background image B by a factor $(1 - \alpha)$ and then adds in the (partial) color values corresponding to the foreground element F .

While the compositing operation is easy to implement, the reverse *matting* operation of estimating F , α , and B given an input image C is much more challenging (Figure 10.40). To see why, observe that while the composite pixel color C provides three measurements,

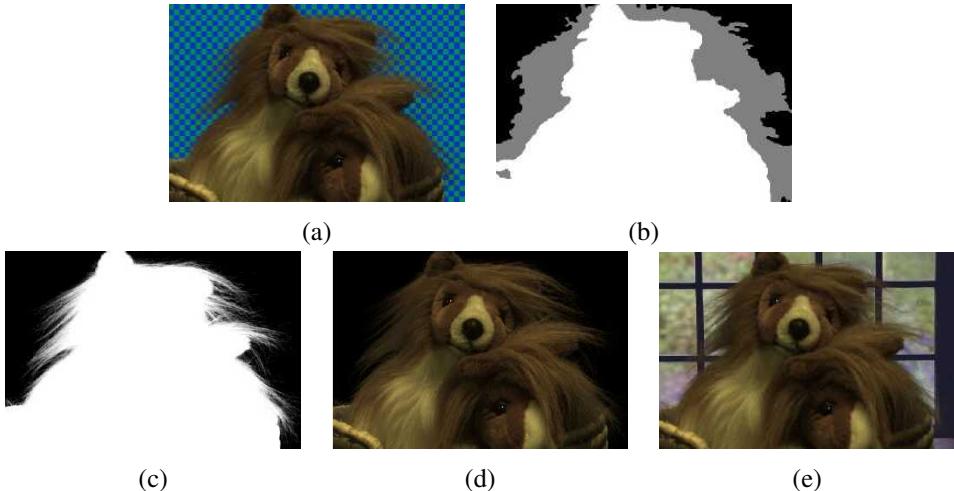


Figure 10.40 Natural image matting (Chuang, Curless et al. 2001) © 2001 IEEE: (a) input image with a “natural” (non-constant) background; (b) hand-drawn trimap—gray indicates unknown regions; (c) extracted alpha map; (d) extracted (premultiplied) foreground colors; (e) composite over a new background.

the F , α , and B unknowns have a total of seven degrees of freedom. Devising techniques to estimate these unknowns despite the underconstrained nature of the problem is the essence of image matting.

In this section, we review a number of image matting techniques. We begin with *blue screen matting*, which assumes that the background is a constant known color, and discuss its variants, two-screen matting (when multiple backgrounds can be used) and difference matting (where the known background is arbitrary). We then discuss local variants of *natural image matting*, where both the foreground and background are unknown. In these applications, it is usual to first specify a *trimap*, i.e., a three-way labeling of the image into foreground, background, and unknown regions (Figure 10.40b). Next, we present some global optimization approaches to natural image matting. Finally, we discuss variants on the matting problem, including shadow matting, flash matting, and environment matting.

10.4.1 Blue screen matting

Blue screen matting involves filming an actor (or object) in front of a constant colored background. While originally bright blue was the preferred color, bright green is now more commonly used (Wright 2006; Brinkmann 2008). Smith and Blinn (1996) discuss a number of

techniques for blue screen matting, which are mostly described in patents rather than in the open research literature. Early techniques used linear combinations of object color channels with user-tuned parameters to estimate the opacity α .

Chuang, Curless *et al.* (2001) describe a newer technique called Mishima's algorithm, which involves fitting two polyhedral surfaces (centered at the mean background color), separating the foreground and background color distributions, and then measuring the relative distance of a novel color to these surfaces to estimate α (Figure 10.41e). While this technique works well in many studio settings, it can still suffer from *blue spill*, where translucent pixels around the edges of an object acquire some of the background blue coloration.

Two-screen matting. In their paper, Smith and Blinn (1996) also introduce an algorithm called *triangulation matting* that uses more than one known background color to over-constrain the equations required to estimate the opacity α and foreground color F .

For example, consider in the compositing equation (10.29) setting the background color to black, i.e., $B = 0$. The resulting composite image C is therefore equal to αF . Replacing the background color with a different known non-zero value B now results in

$$C - \alpha F = (1 - \alpha)B, \quad (10.30)$$

which is an overconstrained set of (color) equations for estimating α . In practice, B should be chosen so as not to saturate C and, for best accuracy, several values of B should be used. It is also important that colors be linearized before processing, which is the case for *all* image matting algorithms. Papers that generate ground truth alpha mattes for evaluation purposes normally use these techniques to obtain accurate matte estimates (Chuang, Curless *et al.* 2001; Wang and Cohen 2007a; Levin, Acha, and Lischinski 2008; Rhemann, Rother *et al.* 2008, 2009).²⁵ Exercise 10.8 has you do this as well.

Difference matting. A related approach when the background is irregular but known is called *difference matting* (Wright 2006; Brinkmann 2008). It is most commonly used when the actor or object is filmed against a static background, e.g., for office video conferencing, person tracking applications (Toyama, Krumm *et al.* 1999), or to produce silhouettes for volumetric 3D reconstruction techniques (Section 12.7.3) (Szeliski 1993; Seitz and Dyer 1997; Seitz, Curless *et al.* 2006). It can also be used with a panning camera where the background is composited from frames where the foreground has been removed using a *garbage matte* (Section 10.4.5) (Chuang, Agarwala *et al.* 2002). Another application is the detection of vi-

²⁵See the alpha matting evaluation website at <http://alphamatting.com>.

sual continuity errors in films, i.e., differences in the background when a shot is re-taken at a later time (Pickup and Zisserman 2009).

In the case where the foreground and background motions can both be specified with parametric transforms, high-quality mattes can be extracted using a generalization of triangulation matting (Wexler, Fitzgibbon, and Zisserman 2002). When frames need to be processed independently, however, the results are often of poor quality (Figure 10.42). In such cases, using a pair of stereo cameras as input can dramatically improve the quality of the results (Criminisi, Cross *et al.* 2006; Yin, Criminisi *et al.* 2007).

10.4.2 Natural image matting

The most general version of image matting is when nothing is known about the background except, perhaps, for a rough segmentation of the scene into foreground, background, and unknown regions, which is known as the *trimap* (Figure 10.40b). Some techniques, however, relax this requirement and allow the user to just draw a few strokes or scribbles in the image: see Figures 10.45 and 10.46 (Wang and Cohen 2005; Wang, Agrawala, and Cohen 2007; Levin, Lischinski, and Weiss 2008; Rhemann, Rother *et al.* 2008; Rhemann, Rother, and Gelautz 2008). Fully automated single image matting results have also been reported (Levin, Acha, and Lischinski 2008; Singaraju, Rother, and Rhemann 2009). The survey paper by Wang and Cohen (2009) has detailed descriptions and comparisons of all of these techniques, a selection of which are described briefly below, while the website <http://alphamatting.com> has up-to-date lists and numerical comparisons of the most recent algorithms.

A relatively simple algorithm for performing natural image matting is Knockout, as described by Chuang, Curless *et al.* (2001) and illustrated in Figure 10.41f. In this algorithm, the nearest known foreground and background pixels (in image space) are determined and then blended with neighboring known pixels to produce a per-pixel foreground F and background B color estimate. The background color is then adjusted so that the measured color C lies on the line between F and B . Finally, opacity α is estimated on a per-channel basis, and the three estimates are combined based on per-channel color differences. (This is an approximation to the least squares solution for α .) Figure 10.42 shows that Knockout has problems when the background consists of more than one dominant local color.

More accurate matting results can be obtained if we treat the foreground and background colors as distributions sampled over some region (Figure 10.41g–h). Ruzon and Tomasi (2000) model local color distributions as mixtures of (uncorrelated) Gaussians and compute these models in strips. They then find the pairing of mixture components F and B that best describes the observed color C , compute the α as the relative distance between these means, and adjust the estimates of F and B so that they are collinear with C .

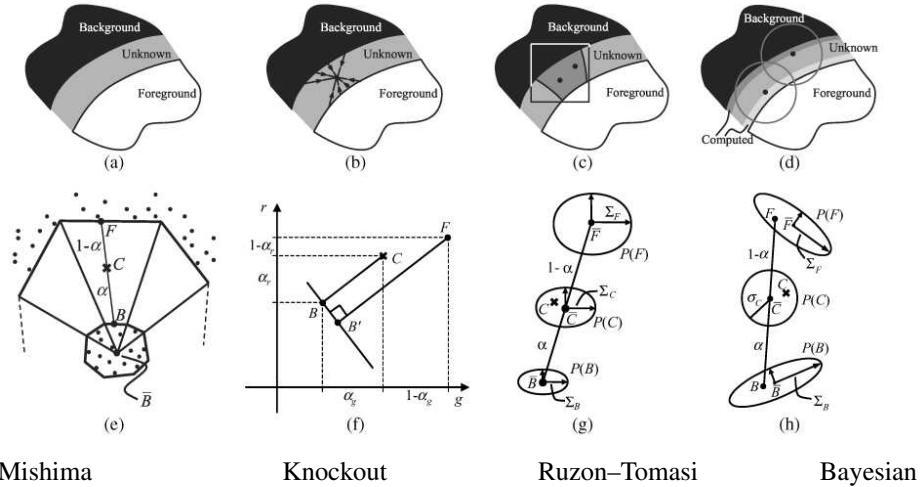


Figure 10.41 Image matting algorithms (Chuang, Curless et al. 2001) © 2001 IEEE. Mishima’s algorithm models global foreground and background color distribution as polyhedral surfaces centered around the mean background (blue) color. Knockout uses a local color estimate of foreground and background for each pixel and computes α along each color axis. Ruzon and Tomasi’s algorithm locally models foreground and background colors and variances. Chuang et al.’s Bayesian matting approach computes a MAP estimate of (fractional) foreground color and opacity given the local foreground and background distributions.

Chuang, Curless *et al.* (2001) and Hillman, Hannah, and Renshaw (2001) use full 3×3 color covariance matrices to model mixtures of correlated Gaussians, and compute estimates independently for each pixel. Matte extraction proceeds in strips starting from known color values growing into the unknown regions, so that recently computed F and B colors can be used in later stages.

To estimate the most likely value of an unknown pixel’s opacity and (unmixed) foreground and background colors, Chuang *et al.* use a fully Bayesian formulation that maximizes

$$P(F, B, \alpha | C) = P(C|F, B, \alpha)P(F)P(B)P(\alpha)/P(C). \quad (10.31)$$

This is equivalent to minimizing the negative log likelihood

$$L(F, B, \alpha | C) = L(C|F, B, \alpha) + L(F) + L(B) + L(\alpha) \quad (10.32)$$

(dropping the $L(C)$ term because it is constant).

Let us examine each of these terms in turn. The first, $L(C|F, B, \alpha)$, is the likelihood that pixel color C was observed given values for the unknowns (F, B, α) . If we assume Gaussian

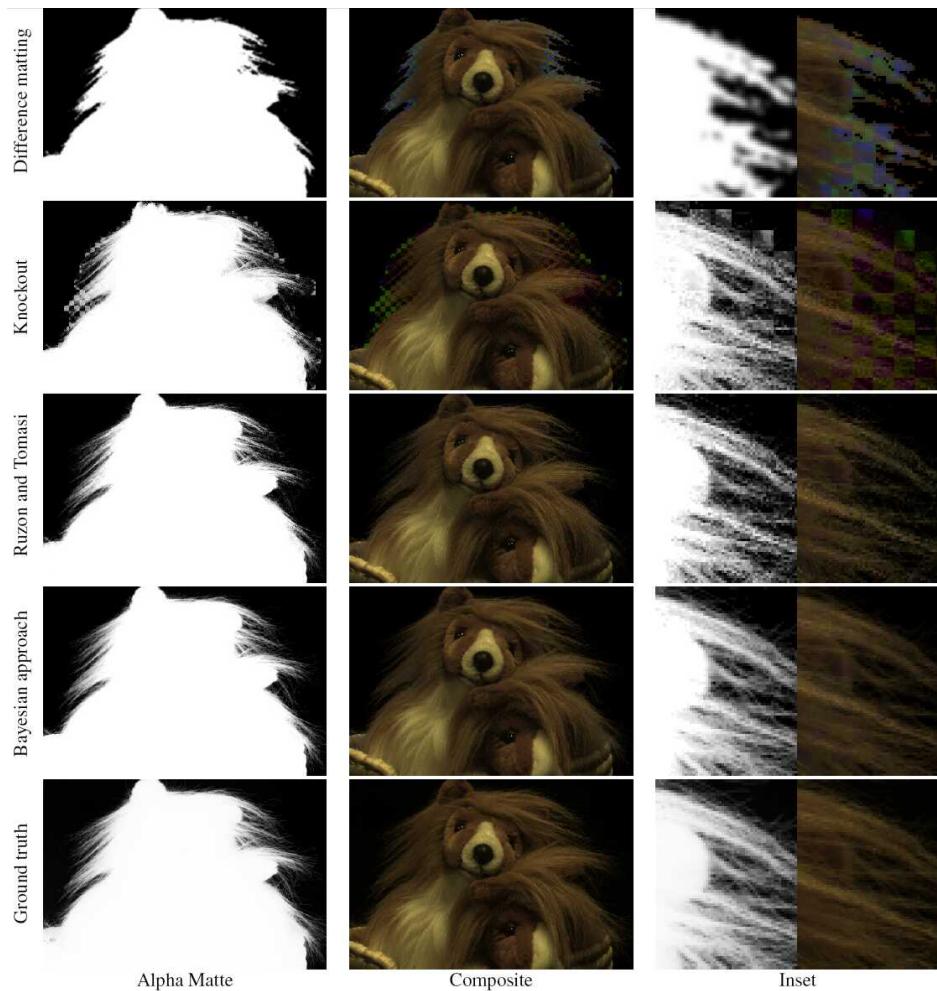


Figure 10.42 *Natural image matting results (Chuang, Curless et al. 2001) © 2001 IEEE. Difference matting and Knockout both perform poorly on this kind of background, while the newer natural image matting techniques perform well. Chuang et al.'s results are slightly smoother and closer to the ground truth.*

noise in our observation with variance σ_C^2 , this negative log likelihood (data term) is

$$L(C) = \frac{1}{2} \|C - [\alpha F + (1 - \alpha)B]\|^2 / \sigma_C^2, \quad (10.33)$$

as illustrated in Figure 10.41h.

The second term, $L(F)$, corresponds to the likelihood that a particular foreground color F comes from the Gaussian mixture model. After partitioning the sample foreground colors into clusters, a weighted mean \bar{F} and covariance Σ_F are computed, where the weights are proportional to a given foreground pixel's opacity and distance from the unknown pixel.²⁶ The negative log likelihood for each cluster is thus given by

$$L(F) = (F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}). \quad (10.34)$$

A similar method is used to estimate unknown background color distributions. If the background is already known, i.e., for blue screen or difference matting applications, its measured color value and variance are used instead.

An alternative to modeling the foreground and background color distributions as mixtures of Gaussians is to keep around the original color samples and to compute the most likely pairings that explain the observed color C (Wang and Cohen 2005, 2007a). These techniques are described in more detail in (Wang and Cohen 2009).

In their Bayesian matting paper, Chuang, Curless *et al.* (2001) assume a constant (non-informative) distribution for $L(\alpha)$. Follow-on papers assume this distribution to be more peaked around 0 and 1, or sometimes use Markov random fields (MRFs) to define a global correlated prior on $P(\alpha)$ (Wang and Cohen 2009).

To compute the most likely estimates for (F, B, α) , the Bayesian matting algorithm alternates between computing (F, B) and α , as each of these problems is quadratic and hence can be solved as a small linear system. When several color clusters are estimated, the most likely pairing of foreground and background color clusters is used.

Bayesian image matting produces results that improve on the original natural image matting algorithm by Ruzon and Tomasi (2000), as can be seen in Figure 10.42. However, compared to later techniques (Wang and Cohen 2009), its performance is not as good for complex backgrounds or inaccurate trimaps (Figure 10.44).

10.4.3 Optimization-based matting

An alternative to estimating each pixel's opacity and foreground color independently is to use global optimization to compute a matte that takes into account correlations between neigh-

²⁶Note that in this whole chapter, we mostly use upper-case italics to denote images or pixel values, even when they are color vectors. The covariance Σ_F is a 3×3 matrix for each foreground cluster.

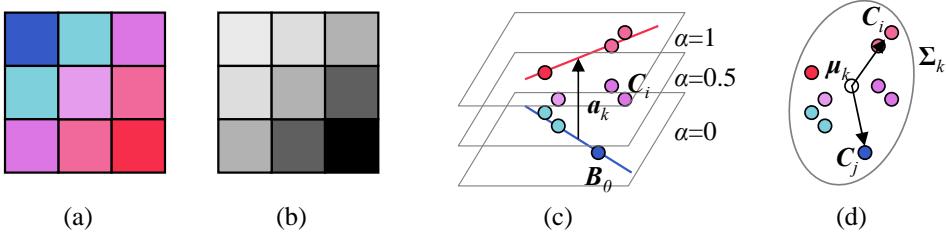


Figure 10.43 Color line matting (Levin, Lischinski, and Weiss 2008): (a) local 3×3 patch of colors; (b) potential assignment of α values; (c) foreground and background color lines, the vector \mathbf{a}_k joining their closest points of intersection, and the family of parallel planes of constant α values, $\alpha_i = \mathbf{a}_k \cdot (\mathbf{C}_i - \mathbf{B}_0)$; (d) a scatter plot of sample colors and the deviations from the mean μ_k for two sample colors \mathbf{C}_i and \mathbf{C}_j .

boring α values. Two examples of this are border matting in the GrabCut interactive segmentation system (Rother, Kolmogorov, and Blake 2004) and Poisson Matting (Sun, Jia *et al.* 2004).

Border matting first dilates the region around the binary segmentation produced by GrabCut (Section 4.3.2) and then solves for a sub-pixel boundary location Δ and a blur width σ for every point along the boundary (Figure 10.39). Smoothness in these parameters along the boundary is enforced using regularization and the optimization is performed using dynamic programming. While this technique can obtain good results for smooth boundaries, such as a person’s face, it has difficulty with fine details, such as hair.

Poisson matting (Sun, Jia *et al.* 2004) assumes a known foreground and background color for each pixel in the trimap (as with Bayesian matting). However, instead of independently estimating each α value, it assumes that the gradient of the alpha matte and the gradient of the color image are related by

$$\nabla \alpha = \frac{F - B}{\|F - B\|^2} \cdot \nabla C, \quad (10.35)$$

which can be derived by taking gradients of both sides of (10.29) and assuming that the foreground and background vary slowly. The per-pixel gradient estimates are then integrated into a continuous $\alpha(\mathbf{x})$ field using the regularization (least squares) technique first described in Section 4.2 (4.24) and subsequently used in Poisson blending (Section 8.4.4, Equation (8.75)) and gradient-based dynamic range compression mapping (Section 10.2.1, Equation (10.18)). This technique works well when good foreground and background color estimates are available and these colors vary slowly.

Instead of computing per-pixel foreground and background colors, Levin, Lischinski, and Weiss (2008) assume only that these color distributions can locally be well approximated as mixtures of two colors, which is known as the *color line model* (Figure 10.43a–c). Under this assumption, a closed-form estimate for α at each pixel i in a (say, 3×3) window W_k is given by

$$\alpha_i = \mathbf{a}_k \cdot (\mathbf{C}_i - \mathbf{B}_0) = \mathbf{a}_k \cdot \mathbf{C} + b_k, \quad (10.36)$$

where \mathbf{C}_i is the pixel color treated as a three-vector, \mathbf{B}_0 is any pixel along the background color line, and \mathbf{a}_k is the vector joining the two closest points on the foreground and background color lines, as shown in Figure 10.43c. (Note that the geometric derivation shown in this figure is an alternative to the algebraic derivation presented by Levin, Lischinski, and Weiss (2008).) Minimizing the deviations of the alpha values α_i from their respective color line models (10.36) over all overlapping windows W_k in the image gives rise to the cost

$$E_\alpha = \sum_k \left(\sum_{i \in W_k} (\alpha_i - \mathbf{a}_k \cdot \mathbf{C}_i - b_k)^2 + \epsilon \|\mathbf{a}_k\| \right), \quad (10.37)$$

where the ϵ term is used to regularize the value of \mathbf{a}_k in the case where the two color distributions overlap (i.e., in constant α regions).

Because this formula is quadratic in the unknowns $\{(\mathbf{a}_k, b_k)\}$, they can be eliminated inside each window W_k , leading to a final energy

$$E_\alpha = \alpha^T \mathbf{L} \alpha, \quad (10.38)$$

where the entries in the \mathbf{L} matrix are given by

$$L_{ij} = \sum_{k: i \in W_k \wedge j \in W_k} \left(\delta_{ij} - \frac{1}{M} \left(1 + (\mathbf{C}_i - \mu_k)^T \hat{\Sigma}_k^{-1} (\mathbf{C}_j - \mu_k) \right) \right), \quad (10.39)$$

where $M = |W_k|$ is the number of pixels in each (overlapping) window, μ_k is the mean color of the pixels in window W_k , and $\hat{\Sigma}_k$ is the 3×3 covariance of the pixel colors plus $\epsilon/M\mathbf{I}$.

Figure 10.43d shows the intuition behind the entries in this affinity matrix, which is called the *matting Laplacian*. Note how when two pixels \mathbf{C}_i and \mathbf{C}_j in W_k point in opposite directions away from the mean μ_k , their weighted dot product is close to -1 , and so their affinity becomes close to 0. Pixels close to each other in color space (and hence with similar expected α values) will have affinities close to $-2/M$.

Minimizing the quadratic energy (10.38) constrained by the known values of $\alpha = \{0, 1\}$ at scribbles only requires the solution of a sparse set of linear equations, which is why the authors call their technique a *closed-form solution* to natural image matting. Once α has

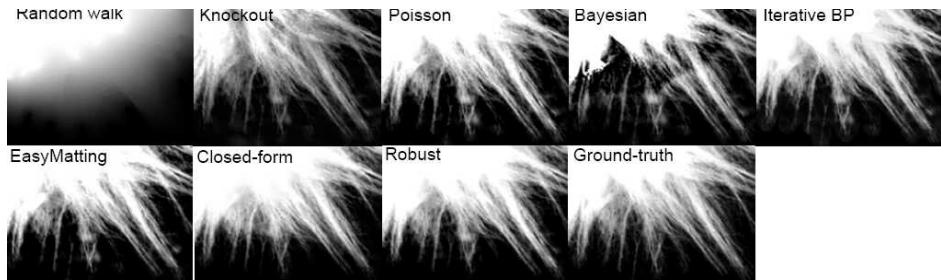


Figure 10.44 Comparative matting results for a medium accuracy trimap. Wang and Cohen (2009) describe the individual techniques being compared.

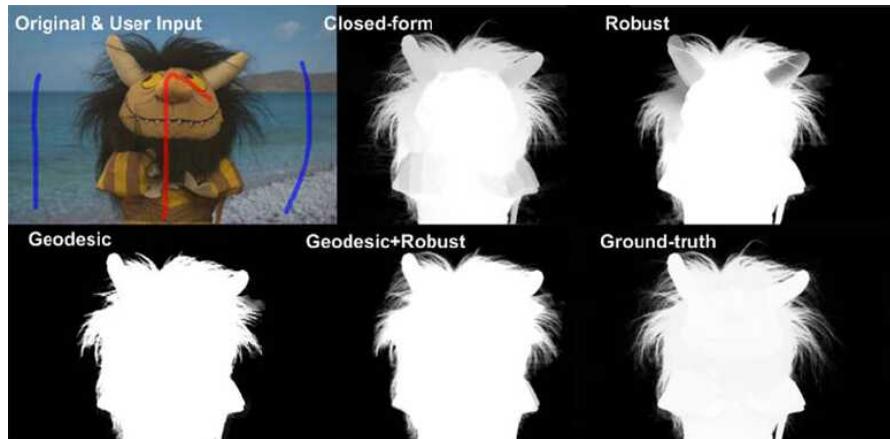


Figure 10.45 Comparative matting results with scribble-based inputs. Wang and Cohen (2009) describe the individual techniques being compared.

been computed, the foreground and background colors are estimated using a least squares minimization of the compositing equation (10.29) regularized with a spatially varying first-order smoothness,

$$E = \sum_i \|C_i - [\alpha + F_i + (1 - \alpha_i)B_i]\|^2 + \lambda |\nabla \alpha_i| (\|\nabla F_i\|^2 + \|\nabla B_i\|^2), \quad (10.40)$$

where the $|\nabla \alpha_i|$ weight is applied separately for the x and y components of the F and B derivatives (Levin, Lischinski, and Weiss 2008).

Laplacian (closed-form) matting is just one of many optimization-based techniques surveyed and compared by Wang and Cohen (2009). Some of these techniques use alternative formulations for the affinities or smoothness terms on the α matte, alternative estimation



Figure 10.46 *Stroke-based segmentation result (Rhemann, Rother et al. 2008) © 2008 IEEE.*

techniques such as belief propagation, or alternative representations (e.g., local histograms) for modeling local foreground and background color distributions (Wang and Cohen 2005, 2007a,b). Some of these techniques also provide real-time results as the user draws a contour line or sparse set of scribbles (Wang, Agrawala, and Cohen 2007; Rhemann, Rother *et al.* 2008) or even pre-segment the image into a small number of mattes that the user can select with simple clicks (Levin, Acha, and Lischinski 2008).

Figure 10.44 shows the results of running a number of the surveyed algorithms on a region of toy animal fur where a trimap has been specified, while Figure 10.45 shows results for techniques that can produce mattes with only a few scribbles as input. Figure 10.46 shows a result for an even more recent algorithm (Rhemann, Rother *et al.* 2008) that claims to outperform all of the techniques surveyed by Wang and Cohen (2009).

The latest results on natural image matting can be found on the <http://alphamatting.com> website created by Rhemann, Rother *et al.* (2009). It currently lists over 60 different algorithms, with most of the more recent algorithms using deep neural networks. The Deep Image Matting paper by Xu, Price *et al.* (2017) provides a larger database of 49,300 training images and 1,000 test images constructed by overlaying manually created color foreground mattes over a variety of backgrounds.²⁷

Pasting. Once a matte has been pulled from an image, it is usually composited directly over the new background, unless the seams between the cutout and background regions are to be hidden, in which case Poisson blending (Pérez, Gangnet, and Blake 2003) can be used (Section 8.4.4).

In the latter case, it is helpful if the matte boundary passes through regions that either have little texture or look similar in the old and new images. Papers by Jia, Sun *et al.* (2006) and Wang and Cohen (2007b) explain how to do this.

²⁷<https://sites.google.com/view/deepimagematting>

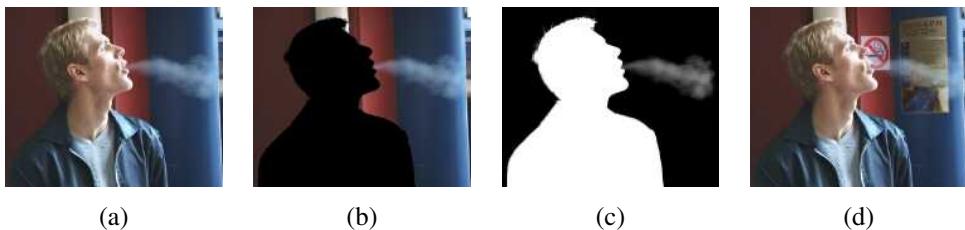


Figure 10.47 *Smoke matting* (Chuang, Agarwala et al. 2002) © 2002 ACM: (a) input video frame; (b) after removing the foreground object; (c) estimated alpha matte; (d) insertion of new objects into the background.

10.4.4 Smoke, shadow, and flash matting

In addition to matting out solid objects with fractional boundaries, it is also possible to matte out translucent media such as smoke (Chuang, Agarwala et al. 2002). Starting with a video sequence, each pixel is modeled as a linear combination of its (unknown) background color and a constant foreground (smoke) color that is common to all pixels. Voting in color space is used to estimate this foreground color and the distance along each color line is used to estimate the per-pixel temporally varying alpha (Figure 10.47).

Extracting and re-inserting shadows is also possible using a related technique (Chuang, Goldman et al. 2003; Wang, Curless, and Seitz 2020). Here, instead of assuming a constant foreground color, each pixel is assumed to vary between its fully lit and fully shadowed colors, which can be estimated by taking (robust) minimum and maximum values over time as a shadow passes over the scene (Exercise 10.9). The resulting fractional *shadow matte* can be used to re-project the shadow into a new scene. If the destination scene has a non-planar geometry, it can be scanned by waving a straight stick shadow across the scene. The new shadow matte can then be warped with the computed deformation field to have it drape correctly over the new scene (Figure 10.48). Shadows can also be extracted from video streams by extending video object segmentation algorithms (Section 9.4.3) to include shadows and other effects such as smoke (Lu, Cole et al. 2021). An example of useful shadow manipulation in photographs is the removal or softening of harsh shadows in people’s portraits (Sun, Barron et al. 2019; Zhou, Hadap et al. 2019; Zhang, Barron et al. 2020), which is available as the Portrait Light feature in Google Photos.²⁸

The quality and reliability of matting algorithms can also be enhanced using more sophisticated acquisition systems. For example, taking a flash and non-flash image pair supports the reliable extraction of foreground mattes, which show up as regions of large illumination

²⁸<https://blog.google/products/photos/new-helpful-editor>

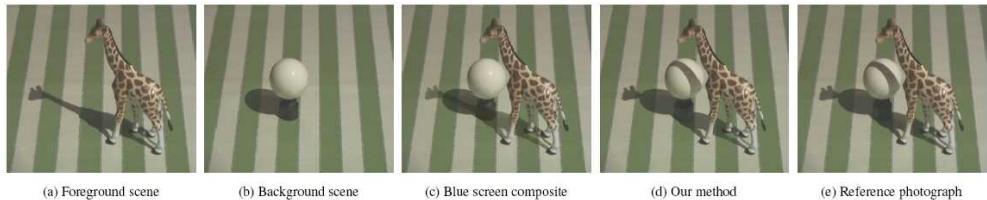


Figure 10.48 *Shadow matting* (Chuang, Goldman et al. 2003) © 2003 ACM. Instead of simply darkening the new scene with the shadow (c), shadow matting correctly dims the lit scene with the new shadow and drapes the shadow over 3D geometry (d).

change between the two images (Sun, Li et al. 2006). Taking simultaneous video streams focused at different distances (McGuire, Matusik et al. 2005) or using multi-camera arrays (Joshi, Matusik, and Avidan 2006) are also good approaches to producing high-quality mattes. These techniques are described in more detail in (Wang and Cohen 2009).

Lastly, photographing a refractive object in front of a number of patterned backgrounds allows the object to be placed in novel 3D environments. These environment matting techniques (Zongker, Werner et al. 1999; Chuang, Zongker et al. 2000) are discussed in Section 14.4.

10.4.5 Video matting

While regular single-frame matting techniques such as blue or green screen matting (Smith and Blinn 1996; Wright 2006; Brinkmann 2008) can be applied to video sequences, the presence of moving objects can sometimes make the matting process easier, as portions of the background may get revealed in preceding or subsequent frames.

Chuang, Agarwala et al. (2002) describe a nice approach to this *video matting* problem, where foreground objects are first removed using a conservative *garbage matte* and the resulting *background plates* are aligned and composited to yield a high-quality background estimate. They also describe how trimaps drawn at sparse keyframes can be interpolated to in-between frames using bi-direction optical flow. Alternative approaches to video matting, such as rotoscoping, which involves drawing curves or strokes in video sequence keyframes (Agarwala, Hertzmann et al. 2004; Wang, Bhat et al. 2005), are discussed in the matting survey paper by Wang and Cohen (2009). There is also a newer dataset of carefully matted stop-motion animation videos created by Erofeev, Gitman et al. (2015).²⁹

Since the original development of video matting techniques, improved algorithms have been developed for both interactive and fully automated *video object segmentation*, as dis-

²⁹<https://videomatting.com>

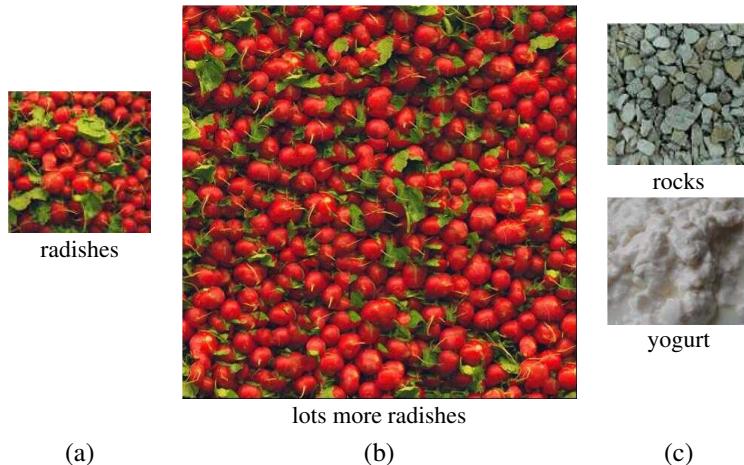


Figure 10.49 *Texture synthesis: (a) given a small patch of texture, the task is to synthesize (b) a similar-looking larger patch; (c) other semi-structured textures that are challenging to synthesize. (Images courtesy of Alyosha Efros.)*

cussed in Section 9.4.3. The paper by Sengupta, Jayaram *et al.* (2020) uses deep learning and adversarial loss, as well as a motion prior, to provide high-quality mattes from small-motion handheld videos where a *clean plate* of the background has also been captured. Wang, Curless, and Seitz (2020) describe a system where shadows and occlusions can be determined by observing people walking around a scene, enabling the insertion of new people at correct scales and lighting. In follow-up work Lin, Ryabtsev *et al.* (2021) describe a high-resolution real-time video matting system along with two new video and image matting datasets. Finally, Lu, Cole *et al.* (2021) describe how to extract shadows, reflections, and other effects associated with objects being tracked and segmented in videos.

10.5 Texture analysis and synthesis

While texture analysis and synthesis may not at first seem like computational photography techniques, they are, in fact, widely used to repair defects, such as small holes, in images or to create non-photorealistic painterly renderings from regular photographs.

The problem of texture synthesis can be formulated as follows: given a small sample of a “texture” (Figure 10.49a), generate a larger similar-looking image (Figure 10.49b). As you can imagine, for certain sample textures, this problem can be quite challenging.

Traditional approaches to texture analysis and synthesis try to match the spectrum of the

source image while generating shaped noise. Matching the frequency characteristics, which is equivalent to matching spatial correlations, is in itself not sufficient. The distributions of the responses at different frequencies must also match. Heeger and Bergen (1995) develop an algorithm that alternates between matching the histograms of multi-scale (steerable pyramid) responses and matching the final image histogram. Portilla and Simoncelli (2000) improve on this technique by also matching pairwise statistics across scale and orientations. De Bonet (1997) uses a coarse-to-fine strategy to find locations in the source texture with a similar *parent structure*, i.e., similar multi-scale oriented filter responses, and then randomly chooses one of these matching locations as the current sample value. Gatys, Ecker, and Bethge (2015) also use a pyramidal fine-to-coarse-to-fine algorithm, but using deep networks trained for object recognition. At each level in the deep network, they gather correlation statistics between various features. During generation, they iteratively update the random image until these more perceptually motivated statistic (Zhang, Isola *et al.* 2018) are matched. We give more details on this and other neural approaches to texture synthesis, such as Shaham, Dekel, and Michaeli (2019), in Section 10.5.3 on neural style transfer.

Exemplar-based texture synthesis algorithms sequentially generate texture pixels by looking for neighborhoods in the source texture that are similar to the currently synthesized image (Efros and Leung 1999). Consider the (as yet) unknown pixel p in the partially constructed texture on the left side of Figure 10.50. As some of its neighboring pixels have been already been synthesized, we can look for similar partial neighborhoods in the sample texture image on the right and randomly select one of these as the new value of p . This process can be repeated down the new image either in a raster fashion or by scanning around the periphery (“onion peeling”) when filling holes, as discussed in (Section 10.5.1). In their actual implementation, Efros and Leung (1999) find the most similar neighborhood and then include all other neighborhoods within a $d = (1 + \epsilon)$ distance, with $\epsilon = 0.1$. They also optionally weight the random pixel selections by the similarity metric d .

To accelerate this process and improve its visual quality, Wei and Levoy (2000) extend this technique using a coarse-to-fine generation process, where coarser levels of the pyramid, which have already been synthesized, are also considered during the matching (De Bonet 1997). To accelerate the nearest neighbor finding, tree-structured vector quantization is used. A much faster version of such nearest neighbor search is the widely used randomized Patch-Match iterative update algorithm developed by Barnes, Shechtman *et al.* (2009).

Efros and Freeman (2001) propose an alternative acceleration and visual quality improvement technique. Instead of synthesizing a single pixel at a time, overlapping square blocks are selected using similarity with previously synthesized regions (Figure 10.51). Once the appropriate blocks have been selected, the seam between newly overlapping blocks is determined

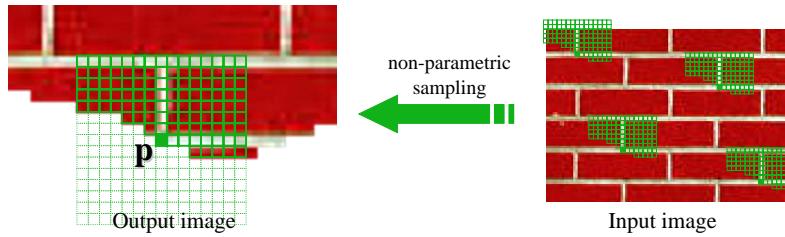


Figure 10.50 Texture synthesis using non-parametric sampling (Efros and Leung 1999). The value of the newest pixel p is randomly chosen from similar local (partial) patches in the source texture (input image). (Figure courtesy of Alyosha Efros.)

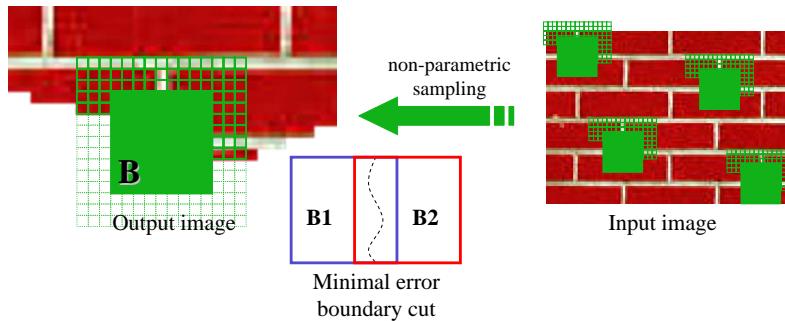


Figure 10.51 Texture synthesis by image quilting (Efros and Freeman 2001). Instead of generating a single pixel at a time, larger blocks are copied from the source texture. The transitions in the overlap regions between the selected blocks are then optimized using dynamic programming. (Figure courtesy of Alyosha Efros.)

using dynamic programming. (Full graph cut seam selection is not required, because only one seam location per row is needed for a vertical boundary.) Because this process involves selecting small patches and them stitching them together, Efros and Freeman (2001) call their system *image quilting*. Komodakis and Tziritas (2007) present an MRF-based version of this block synthesis algorithm that uses a new, efficient version of loopy belief propagation they call ‘Priority-BP’. Wei, Lefebvre *et al.* (2009) present a comprehensive survey of work in exemplar-based texture synthesis through 2009.

10.5.1 Application: Hole filling and inpainting

Filling holes left behind when objects or defects are excised from photographs, which is known as *inpainting*, is one of the most common applications of texture synthesis. Such

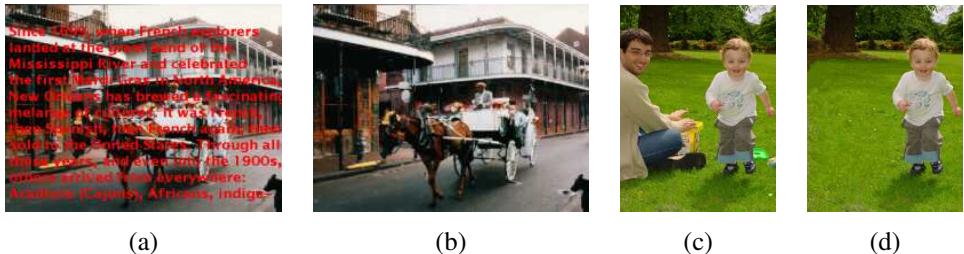


Figure 10.52 *Image inpainting (hole filling): (a–b) propagation along isophote directions (Bertalmio, Sapiro et al. 2000) © 2000 ACM; (c–d) exemplar-based inpainting with confidence-based filling order (Criminisi, Pérez, and Toyama 2004).*

techniques are used not only to remove unwanted people or interlopers from photographs (King 1997) but also to fix small defects in old photos and movies (*scratch removal*) or to remove wires holding props or actors in mid-air during filming (*wire removal*). Bertalmio, Sapiro *et al.* (2000) solve the problem by propagating pixel values along isophote (constant-value) directions interleaved with some anisotropic diffusion steps (Figure 10.52a–b). Telea (2004) develops a faster technique that uses the fast marching method from level sets (Section 7.3.2). However, these techniques will not hallucinate texture in the missing regions. Bertalmio, Vese *et al.* (2003) augment their earlier technique by adding synthetic texture to the infilled regions.

The example-based (non-parametric) texture generation techniques discussed in the previous section can also be used by filling the holes from the outside in (the “onion-peel” ordering). However, this approach may fail to propagate strong oriented structures. Criminisi, Pérez, and Toyama (2004) use exemplar-based texture synthesis where the order of synthesis is determined by the strength of the gradient along the region boundary (Figures 10.1d and 10.52c–d). Sun, Yuan *et al.* (2004) present a related approach where the user draws interactive lines to indicate where structures should be preferentially propagated. Additional techniques related to these approaches include those developed by Drori, Cohen-Or, and Yeshurun (2003), Kwatra, Schödl *et al.* (2003), Kwatra, Essa *et al.* (2005), Wilczkowiak, Brostow *et al.* (2005), Komodakis and Tziritas (2007), and Wexler, Shechtman, and Irani (2007).

Most hole filling algorithms borrow small pieces of the original image to fill in the holes. When a large database of source images is available, e.g., when images are taken from a photo sharing site or the internet, it is sometimes possible to copy a single contiguous image region to fill the hole. Hays and Efros (2007) present such a technique, which uses image context and boundary compatibility to select the source image, which is then blended with the original (holey) image using graph cuts and Poisson blending. This technique is discussed

in more detail in Section 6.4.4 and Figure 6.40.

As with other areas of image processing, deep neural networks are used in all of the latest techniques (Yang, Lu *et al.* 2017; Yu, Lin *et al.* 2018; Liu, Reda *et al.* 2018; Zeng, Fu *et al.* 2019; Yu, Lin *et al.* 2019; Chang, Liu *et al.* 2019; Nazeri, Ng *et al.* 2019; Ren, Yu *et al.* 2019; Shih, Su *et al.* 2020; Yi, Tang *et al.* 2020). Some of these papers have introduced interesting new extensions to neural network architectures, such as *partial convolutions* (Liu, Reda *et al.* 2018) and *partial convolutions* (Yu, Lin *et al.* 2019), the propagation of edge structures (Nazeri, Ng *et al.* 2019; Ren, Yu *et al.* 2019), multi-resolution attention and residuals (Yi, Tang *et al.* 2020), and iterative confidence feedback (Zeng, Lin *et al.* 2020). Inpainting has also been applied to video sequences (e.g., Gao, Saraf *et al.* 2020). Results on recent challenges on image inpainting can be found in the AIM 2020 Workshop and Challenges on this topic (Ntavelis, Romero *et al.* 2020a).

10.5.2 Application: Non-photorealistic rendering

Two more applications of the exemplar-based texture synthesis ideas are texture transfer (Efros and Freeman 2001) and image analogies (Hertzmann, Jacobs *et al.* 2001), which are both examples of non-photorealistic rendering (Gooch and Gooch 2001).

In addition to using a source texture image, texture transfer also takes a reference (or target) image, and tries to match certain characteristics of the target image with the newly synthesized image. For example, the new image being rendered in Figure 10.53c not only tries to satisfy the usual similarity constraints with the source texture in Figure 10.53b, but it also tries to match the luminance characteristics of the reference image. Efros and Freeman (2001) mention that blurred image intensities or local image orientation angles are alternative quantities that could be matched.

Hertzmann, Jacobs *et al.* (2001) formulate the following problem:

Given a pair of images A and A' (the unfiltered and filtered source images, respectively), along with some additional unfiltered target image B , synthesize a new filtered target image B' such that

$$A : A' :: B : B'.$$

Instead of having the user program a certain non-photorealistic rendering effect, it is sufficient to supply the system with examples of before and after images, and let the system synthesize the novel image using exemplar-based synthesis, as shown in Figure 10.54.

The algorithm used to solve image analogies proceeds in a manner analogous to the texture synthesis algorithms of Efros and Leung (1999) and Wei and Levoy (2000). Once Gaus-

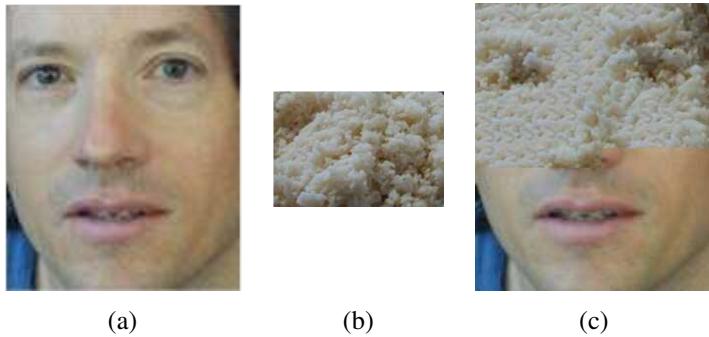


Figure 10.53 *Texture transfer* (Efros and Freeman 2001) © 2001 ACM: (a) reference (target) image; (b) source texture; (c) image (partially) rendered using the texture.



Figure 10.54 *Image analogies* (Hertzmann, Jacobs et al. 2001) © 2001 ACM. Given an example pair of a source image A and its rendered (filtered) version A' , generate the rendered version B' from another unfiltered source image B .

sian pyramids have been computed for all of the source and reference images, the algorithm looks for neighborhoods in the source filtered pyramids generated from A' that are similar to the partially constructed neighborhood in B' , while at the same time having similar multi-resolution appearances at corresponding locations in A and B . As with texture transfer, appearance characteristics can include not only (blurred) color or luminance values but also orientations.

This general framework allows image analogies to be applied to a variety of rendering tasks. In addition to exemplar-based non-photorealistic rendering, image analogies can be used for traditional texture synthesis, super-resolution, and texture transfer (using the same textured image for both A and A'). If only the filtered (rendered) image A' is available, as is the case with paintings, the missing reference image A can be hallucinated using a smart (edge preserving) blur operator. Finally, it is possible to train a system to perform *texture-by-numbers* by manually painting over a natural image with pseudocolors corresponding to pixels' semantic meanings, e.g., water, trees, and grass (Figure 10.55a–b). The resulting system

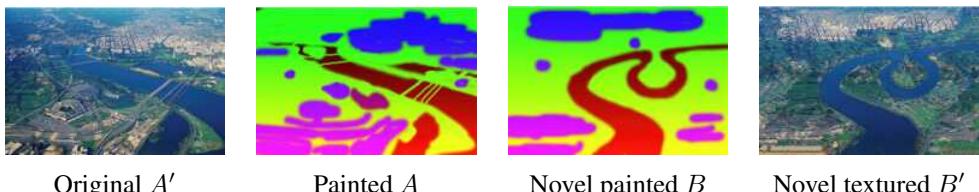


Figure 10.55 *Texture-by-numbers* (Hertzmann, Jacobs et al. 2001) © 2001 ACM. Given a textured image A' and a hand-labeled (painted) version A , synthesize a new image B' given just the painted version B .

can then convert a novel sketch into a fully rendered synthetic photograph (Figure 10.55c–d). In more recent work, Cheng, Vishwanathan, and Zhang (2008) add ideas from image quilting (Efros and Freeman 2001) and MRF inference (Komodakis, Tziritas, and Paragios 2008) to the basic image analogies algorithm, while Ramanarayanan and Bala (2007) recast this process as energy minimization, which means it can be viewed as a conditional random field (Section 4.3.1), and devise an efficient algorithm to find a good minimum.

More traditional filtering and feature detection techniques can also be used for non-photorealistic rendering.³⁰ For example, pen-and-ink illustration (Winkenbach and Salesin 1994) and painterly rendering techniques (Litwinowicz 1997) use local color, intensity, and orientation estimates as an input to their procedural rendering algorithms. Techniques for stylizing and simplifying photographs and video (DeCarlo and Santella 2002; Winnemöller, Olsen, and Gooch 2006; Farbman, Fattal et al. 2008), as in Figure 10.56, use combinations of edge-preserving blurring (Section 3.3.1) and edge detection and enhancement (Section 7.2.3).

10.5.3 Neural style transfer and semantic image synthesis

With the advent of deep learning, image-guided exemplar-based texture synthesis has mostly been replaced with statistics matching in deep networks (Gatys, Ecker, and Bethge 2015). Figure 10.57 illustrates the basic idea used in neural style transfer networks. In the original work of Gatys, Ecker, and Bethge (2016), a style image y_s and a content image y_c (see Figure 10.58 for examples) are input to a *loss network*, which compares features derived from the style and target images with those derived from the image \hat{y} being synthesized. These losses are normally a combination of a *perceptual loss*. The gradients of these losses are used to adjust the generated image \hat{y} in an iterative fashion, which makes this process

³⁰For a good selection of papers, see the Symposia on Non-Photorealistic Animation and Rendering (NPRA).



Figure 10.56 Non-photorealistic abstraction of photographs: (a) (DeCarlo and Santella 2002) © 2002 ACM and (b) (Farbman, Fattal et al. 2008) © 2008 ACM.

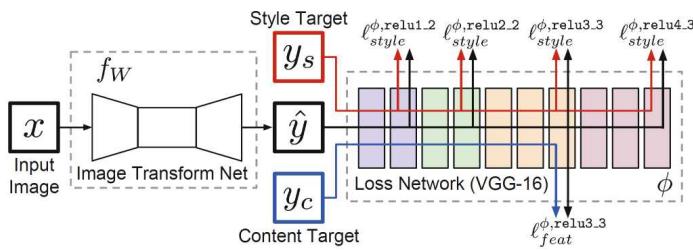


Figure 10.57 Network architecture for neural style transfer, which learns to transform images in one particular style (Johnson, Alahi, and Fei-Fei 2016) © 2016 Springer. During training, the content target image y_c is fed into the image transformation network as an input x , along with a style image y_s , and the network weights are updated so as to minimize the perceptual losses, i.e., the style reconstruction loss l_{style} and the feature reconstruction loss l_{feat} . The earlier network by Gatys, Ecker, and Bethge (2015) did not have an image transformation network, and instead used the losses to optimize the transformed image \hat{y} .

quite slow.

To accelerate this, Johnson, Alahi, and Fei-Fei (2016) train a feedforward *image transformation network* with a fixed style image and many different content targets, adjusting the network weights so that the stylized image \hat{y} resulting from a target y_c matches the desired statistics. When a new image x is presented to be stylized, it is simply run through the image transformation network. Figure 10.58a shows some comparisons between Gatys, Ecker, and Bethge (2016) and Johnson, Alahi, and Fei-Fei (2016).

Perceptual loss has now become a standard component of image synthesis systems (Dosovitskiy and Brox 2016), often as an additional component to the generative adversarial loss (Section 5.5.4). They are also sometimes used as an alternative to older image quality metrics such as SSIM (Zhang, Isola *et al.* 2018; Talebi and Milanfar 2018; Tariq, Tursun *et al.* 2020; Czolbe, Krause *et al.* 2020).

The basic architecture in Johnson, Alahi, and Fei-Fei (2016) was extended by Ulyanov, Vedaldi, and Lempitsky (2017), who show that using *instance normalization* instead of batch normalization significantly improves the results. Dumoulin, Shlens, and Kudlur (2017) and Huang and Belongie (2017) further extended these ideas to train one network to mimic different styles, using *conditional instance normalization* and *adaptive instance normalization* to select among the pre-trained styles (or in-between blends), as shown in Figure 10.58b.

Neural style transfer continues to be an actively studied area, with related approaches working on more generalized *image-to-image translation* (Isola, Zhu *et al.* 2017) and *semantic photo synthesis* (Chen and Koltun 2017; Park, Liu *et al.* 2019; Bau, Strobelt *et al.* 2019; Ntavelis, Romero *et al.* 2020b) applications—see Tewari, Fried *et al.* (2020, Section 6.1) for a recent survey. Most of the newer architectures use generative adversarial networks (GANs) (Kotovenko, Sanakoyeu *et al.* 2019; Shaham, Dekel, and Michaeli 2019; Yang, Wang *et al.* 2019; Svoboda, Anoosheh *et al.* 2020; Wang, Li *et al.* 2020; Xia, Zhang *et al.* 2020; Härkönen, Hertzmann *et al.* 2020), which we discussed in Section 5.5.4. There’s also a recent course on the more general topic of learning-based image synthesis (Zhu 2021).

10.6 Additional reading

Good overviews of the first decade of computational photography can be found in the book by Raskar and Tumblin (2010) and survey articles by Nayar (2006), Cohen and Szeliski (2006), Levoy (2006),Debevec (2006), and Hayes (2008), as well as two special journal issues edited by Bimber (2006) and Durand and Szeliski (2007). Notes from the courses on computational photography mentioned at the beginning of this chapter are another great source for more

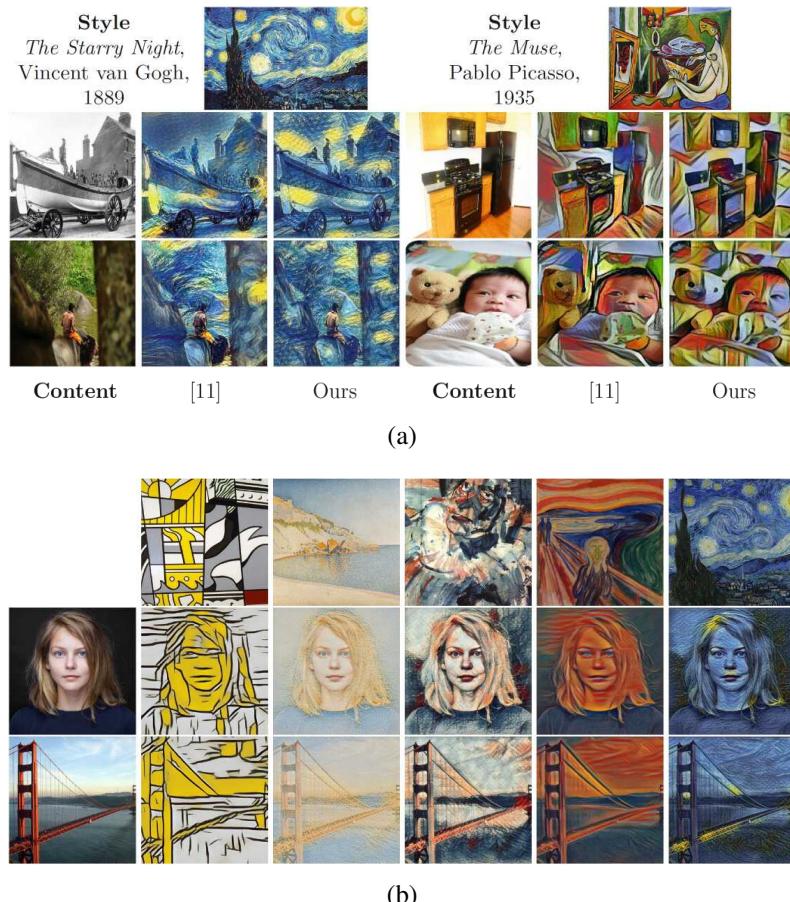


Figure 10.58 Two examples of neural style transfer: (a) the pre-trained network of Johnson, Alahi, and Fei-Fei (2016) © 2016 Springer (labeled “Ours”) vs. (Gatys, Ecker, and Bethge 2016) (labeled “[11]”); (b) a network that uses conditional instance normalization to mimic different styles (top row) applied to various content (left column) © (Dumoulin, Shlens, and Kudlur 2017).

recent material and references.³¹

The sub-field of high dynamic range imaging has its own book discussing research in this area (Reinhard, Heidrich *et al.* 2010), as well as some books describing related photographic techniques (Freeman 2008; Gulbins and Gulbins 2009). Algorithms for calibrating the radiometric response function of a camera can be found in articles by Mann and Picard (1995),Debevec and Malik (1997), and Mitsunaga and Nayar (1999).

The subject of tone mapping is treated extensively in (Reinhard, Heidrich *et al.* 2010). Representative papers from the large volume of literature on this topic include (Tumblin and Rushmeier 1993; Larson, Rushmeier, and Piatko 1997; Pattanaik, Ferwerda *et al.* 1998; Tumblin and Turk 1999; Durand and Dorsey 2002; Fattal, Lischinski, and Werman 2002; Reinhard, Stark *et al.* 2002; Lischinski, Farbman *et al.* 2006; Farbman, Fattal *et al.* 2008; Paris, Hasinoff, and Kautz 2011; Aubry, Paris *et al.* 2014).

The literature on super-resolution is quite extensive (Chaudhuri 2001; Park, Park, and Kang 2003; Capel and Zisserman 2003; Capel 2004; van Ouwerkerk 2006). The term super-resolution usually describes techniques for aligning and merging multiple images to produce higher-resolution composites (Keren, Peleg, and Brada 1988; Irani and Peleg 1991; Cheeseman, Kanefsky *et al.* 1993; Mann and Picard 1994; Chiang and Boult 1996; Basicle, Blake, and Zisserman 1996; Capel and Zisserman 1998; Smelyanskiy, Cheeseman *et al.* 2000; Capel and Zisserman 2000; Pickup, Capel *et al.* 2009; Gulbins and Gulbins 2009; Hasinoff, Sharlet *et al.* 2016; Wronski, Garcia-Dorado *et al.* 2019). However, single-image super-resolution techniques have also been developed (Freeman, Jones, and Pasztor 2002; Baker and Kanade 2002; Fattal 2007; Dong, Loy *et al.* 2016; Cai, Gu *et al.* 2019; Anwar, Khan, and Barnes 2020). Such techniques are closely related to denoising (Zhang, Zuo *et al.* 2017; Brown 2019; Liba, Murthy *et al.* 2019; Gu and Timofte 2019), deblurring and blind image deconvolution (Campisi and Egiazarian 2017; Zhang, Dai *et al.* 2019; Kupyn, Martyniuk *et al.* 2019), and demosaicing (Chatterjee, Joshi *et al.* 2011; Gharbi, Chaurasia *et al.* 2016; Abdelhamed, Afifi *et al.* 2020).

A good survey on image matting is given by Wang and Cohen (2009). Representative papers, which include extensive comparisons with previous work, include (Chuang, Curless *et al.* 2001; Wang and Cohen 2007a; Levin, Acha, and Lischinski 2008; Rhemann, Rother *et al.* 2008, 2009; Xu, Price *et al.* 2017). You can find pointers to recent papers and results on the <http://alphamatting.com> website created by Rhemann, Rother *et al.* (2009). Matting ideas can also be applied to manipulate shadows (Chuang, Goldman *et al.* 2003; Sun, Barron *et al.*

³¹CMU 15-463, http://graphics.cs.cmu.edu/courses/15-463/2008_fall, Berkeley CS194-26/294-26, <https://inst.eecs.berkeley.edu/~cs194-26/fa20>, MIT 6.815/6.865, <https://stellar.mit.edu/S/course/6/sp08/6.815/materials.html>, Stanford CS 448A, <https://graphics.stanford.edu/courses/cs448a-08-spring>, CMU 16-726, <https://learning-image-synthesis.github.io>, and SIGGRAPH courses, <https://web.media.mit.edu/~raskar/photo>.

2019; Zhou, Hadap *et al.* 2019; Zhang, Barron *et al.* 2020; Wang, Curless, and Seitz 2020) and videos (Chuang, Agarwala *et al.* 2002; Wang, Bhat *et al.* 2005; Erofeev, Gitman *et al.* 2015; Sengupta, Jayaram *et al.* 2020; Lin, Ryabtsev *et al.* 2021).

The literature on texture synthesis and hole filling includes traditional approaches to texture synthesis, which try to match image statistics between source and destination images (Heeger and Bergen 1995; De Bonet 1997; Portilla and Simoncelli 2000), as well as approaches that search for matching neighborhoods or patches inside the source sample (Efros and Leung 1999; Wei and Levoy 2000; Efros and Freeman 2001; Wei, Lefebvre *et al.* 2009) or use neural networks (Gatys, Ecker, and Bethge 2015; Shaham, Dekel, and Michaeli 2019). In a similar vein, traditional approaches to hole filling involve the solution of local variational (smooth continuation) problems (Bertalmio, Sapiro *et al.* 2000; Bertalmio, Vese *et al.* 2003; Telea 2004). The next wave of techniques use data-driven texture synthesis approaches (Drori, Cohen-Or, and Yeshurun 2003; Kwatra, Schödl *et al.* 2003; Criminisi, Pérez, and Toyama 2004; Sun, Yuan *et al.* 2004; Kwatra, Essa *et al.* 2005; Wilczkowiak, Brostow *et al.* 2005; Komodakis and Tziritas 2007; Wexler, Shechtman, and Irani 2007). The most recent algorithms for image and video inpainting use deep neural networks (Yang, Lu *et al.* 2017; Yu, Lin *et al.* 2018; Liu, Reda *et al.* 2018; Shih, Su *et al.* 2020; Yi, Tang *et al.* 2020; Gao, Saraf *et al.* 2020; Ntavelis, Romero *et al.* 2020a). In addition to generating isolated patches of texture or inpainting missing region, related techniques can also be used to transfer the style of an image or painting to another one (Efros and Freeman 2001; Hertzmann, Jacobs *et al.* 2001; Gatys, Ecker, and Bethge 2016; Johnson, Alahi, and Fei-Fei 2016; Dumoulin, Shlens, and Kudlur 2017; Huang and Belongie 2017; Shaham, Dekel, and Michaeli 2019).

10.7 Exercises

Ex 10.1: Radiometric calibration. Implement one of the multi-exposure radiometric calibration algorithms described in Section 10.2 (Debevec and Malik 1997; Mitsunaga and Nayar 1999; Reinhard, Heidrich *et al.* 2010). This calibration will be useful in a number of different applications, such as stitching images or stereo matching with different exposures and shape from shading.

1. Take a series of bracketed images with your camera on a tripod. If your camera has an automatic exposure bracketing (AEB) modes, taking three images may be sufficient to calibrate most of your camera’s dynamic range, especially if your scene has a lot of bright and dark regions. (Shooting outdoors or through a window on a sunny day is best.)

2. If your images are not taken on a tripod, first perform a global alignment.
3. Estimate the radiometric response function using one of the techniques cited above.
4. Estimate the high dynamic range radiance image by selecting or blending pixels from different exposures (Debevec and Malik 1997; Mitsunaga and Nayar 1999; Eden, Uyttendaele, and Szeliski 2006).
5. Repeat your calibration experiments under different conditions, e.g., indoors under incandescent light, to get a sense for the range of color balancing effects that your camera imposes.
6. If your camera supports RAW and JPEG mode, calibrate both sets of images simultaneously and to each other (the radiance at each pixel will correspond). See if you can come up with a model for what your camera does, e.g., whether it treats color balance as a diagonal or full 3×3 matrix multiply, whether it uses non-linearities in addition to gamma, whether it sharpens the image while “developing” the JPEG image, etc.
7. Develop an interactive viewer to change the exposure of an image based on the average exposure of a region around the mouse. (One variant is to show the adjusted image inside a window around the mouse. Another is to adjust the complete image based on the mouse position.)
8. Implement a tone mapping operator (Exercise 10.5) and use this to map your radiance image to a displayable gamut.

Ex 10.2: Noise level function. Determine your camera’s noise level function using either multiple shots or by analyzing smooth regions.

1. Set up your camera on a tripod looking at a calibration target or a static scene with a good variation in input levels and colors. (Check your camera’s histogram to ensure that all values are being sampled.)
2. Take repeated images of the same scene (ideally with a remote shutter release) and average them to compute the variance at each pixel. Discarding pixels near high gradients (which are affected by camera motion), plot for each color channel the standard deviation at each pixel as a function of its output value.
3. Fit a lower envelope to these measurements and use this as your noise level function. How much variation do you see in the noise as a function of input level? How much of this is significant, i.e., away from flat regions in your camera response function where you do not want to be sampling anyway?

4. (Optional) Using the same images, develop a technique that segments the image into near-constant regions (Liu, Szeliski *et al.* 2008). (This is easier if you are photographing a calibration chart.) Compute the deviations for each region from a *single* image and use them to estimate the NLF. How does this compare to the multi-image technique, and how stable are your estimates from image to image?

Ex 10.3: Vignetting. Estimate the amount of vignetting in some of your lenses using one of the following three techniques (or devise one of your choosing):

1. Take an image of a large uniform intensity region (well-illuminated wall or blue sky—but be careful of brightness gradients) and fit a radial polynomial curve to estimate the vignetting.
2. Construct a center-weighted panorama and compare these pixel values to the input image values to estimate the vignetting function. Weight pixels in slowly varying regions more highly, as small misalignments will give large errors at high gradients. Optionally estimate the radiometric response function as well (Litvinov and Schechner 2005; Goldman 2010).
3. Analyze the radial gradients (especially in low-gradient regions) and fit the robust means of these gradients to the derivative of the vignetting function, as described by Zheng, Yu *et al.* (2008).

For the parametric form of your vignetting function, you can either use a simple radial function, e.g.,

$$f(r) = 1 + \alpha_1 r + \alpha_2 r^2 + \dots \quad (10.41)$$

or one of the specialized equations developed by Kang and Weiss (2000) and Zheng, Lin, and Kang (2006).

In all of these cases, be sure that you are using linearized intensity measurements, by using either RAW images or images linearized through a radiometric response function, or at least images where the gamma curve has been removed.

(Optional) What happens if you forget to undo the gamma before fitting a (multiplicative) vignetting function?

Ex 10.4: Optical blur (PSF) estimation. Compute the optical PSF either using a known target (Figure 10.7) or by detecting and fitting step edges (Section 10.1.4) (Joshi, Szeliski, and Kriegman 2008; Cho, Paris *et al.* 2011).

1. Detect strong edges to sub-pixel precision.

2. Fit a local profile to each oriented edge and fill these pixels into an ideal target image, either at image resolution or at a higher resolution (Figure 10.9c–d).
3. Use least squares (10.1) at valid pixels to estimate the PSF kernel K , either globally or in locally overlapping sub-regions of the image.
4. Visualize the recovered PSFs and use them to remove chromatic aberration or deblur the image.

Ex 10.5: Tone mapping. Implement one of the tone mapping algorithms discussed in Section 10.2.1 (Durand and Dorsey 2002; Fattal, Lischinski, and Werman 2002; Reinhard, Stark *et al.* 2002; Lischinski, Farbman *et al.* 2006) or any of the numerous additional algorithms discussed by Reinhard, Heidrich *et al.* (2010) and <https://stellar.mit.edu/S/course/6/sp08/6.815/materials.html>.

(Optional) Compare your algorithm to local histogram equalization (Section 3.1.4).

Ex 10.6: Flash enhancement. Develop an algorithm to combine flash and non-flash photographs to best effect. You can use ideas from Eisemann and Durand (2004) and Petschnigg, Agrawala *et al.* (2004) or anything else you think might work well.

Ex 10.7: Super-resolution. Implement one or more super-resolution algorithms and compare their performance.

1. Take a set of photographs of the same scene using a hand-held camera (to ensure that there is some jitter between the photographs).
2. Determine the PSF for the images you are trying to super-resolve using one of the techniques in Exercise 10.4.
3. Alternatively, simulate a collection of lower-resolution images by taking a high-quality photograph (avoid those with compression artifacts) and applying your own prefilter kernel and downsampling.
4. Estimate the relative motion between the images using a parametric translation and rotation motion estimation algorithm (Sections 8.1.3 or 9.2).
5. Implement a basic least squares super-resolution algorithm by minimizing the difference between the observed and downsampled images (10.26–10.27).
6. Add in a gradient image prior, either as another least squares term or as a robust term that can be minimized using iteratively reweighted least squares (Appendix A.3).

7. (Optional) Implement one of the example-based super-resolution techniques, where matching against a set of exemplar images is used either to infer higher-frequency information to be added to the reconstruction (Freeman, Jones, and Pasztor 2002) or higher-frequency gradients to be matched in the super-resolved image (Baker and Kanade 2002).
8. (Optional) Use local edge statistic information to improve the quality of the super-resolved image (Fattal 2007).
9. (Optional) Try some of the newest DNN-based super-resolution algorithms.

Ex 10.8: Image matting. Develop an algorithm for pulling a foreground matte from natural images, as described in Section 10.4.

1. Make sure that the images you are taking are linearized (Exercise 10.1 and Section 10.1) and that your camera exposure is fixed (full manual mode), at least when taking multiple shots of the same scene.
2. To acquire ground truth data, place your object in front of a computer monitor and display a variety of solid background colors as well as some natural imagery.
3. Remove your object and re-display the same images to acquire known background colors.
4. Use triangulation matting (Smith and Blinn 1996) to estimate the ground truth opacities α and pre-multiplied foreground colors αF for your objects.
5. Implement one or more of the natural image matting algorithms described in Section 10.4 and compare your results to the ground truth values you computed. Alternatively, use the matting test images published on <http://alphamatting.com>.
6. (Optional) Run your algorithms on other images taken with the same calibrated camera (or other images you find interesting).

Ex 10.9: Smoke and shadow matting. Extract smoke or shadow mattes from one scene and insert them into another (Chuang, Agarwala *et al.* 2002; Chuang, Goldman *et al.* 2003).

1. Take a still or video sequence of images with and without some intermittent smoke and shadows. (Remember to linearize your images before proceeding with any computations.)
2. For each pixel, fit a line to the observed color values.

3. If performing smoke matting, robustly compute the intersection of these lines to obtain the smoke color estimate. Then, estimate the background color as the other extremum (unless you have already taken a smoke-free background image).
If performing shadow matting, compute robust shadow (minimum) and lit (maximum) values for each pixel.
4. Extract the smoke or shadow mattes from each frame as the fraction between these two values (background and smoke or shadowed and lit).
5. Scan a new (destination) scene or modify the original background with an image editor.
6. Re-insert the smoke or shadow matte, along with any other foreground objects you may have extracted.
7. (Optional) Using a series of cast stick shadows, estimate the deformation field for the destination scene to correctly warp (drape) the shadows across the new geometry. (This is related to the shadow scanning technique developed by Bouguet and Perona (1999) and implemented in Exercise 13.2.)
8. (Optional) Chuang, Goldman *et al.* (2003) only demonstrated their technique for planar source geometries. Can you extend their technique to capture shadows acquired from an irregular source geometry?
9. (Optional) Can you change the direction of the shadow, i.e., simulate the effect of changing the light source direction?
10. (Optional) Re-implement the facial shadow removal algorithm of Zhang, Barron *et al.* (2020) and try applying it to other domains.

Ex 10.10: Texture synthesis. Implement one of the texture synthesis or hole filling algorithms presented in Section 10.5. Here is one possible procedure:

1. Implement the basic Efros and Leung (1999) algorithm, i.e., starting from the outside (for hole filling) or in raster order (for texture synthesis), search for a similar neighborhood in the source texture image, and copy that pixel.
2. Add in the Wei and Levoy (2000) extension of generating the pixels in a coarse-to-fine fashion, i.e., generate a lower-resolution synthetic texture (or filled image), and use this as a guide for matching regions in the finer resolution version.
3. Add in the Criminisi, Pérez, and Toyama (2004) idea of prioritizing pixels to be filled by some function of the local structure (gradient or orientation strength).

4. Extend any of the above algorithms by selecting sub-blocks in the source texture and using optimization to determine the seam between the new block and the existing image that it overlaps (Efros and Freeman 2001).
5. (Optional) Implement one of the isophote (smooth continuation) inpainting algorithms (Bertalmio, Sapiro *et al.* 2000; Telea 2004).
6. (Optional) Add the ability to supply a target (reference) image (Efros and Freeman 2001) or to provide sample filtered or unfiltered (reference and rendered) images (Hertzmann, Jacobs *et al.* 2001), see Section 10.5.2.
7. (Optional) Try some of the newer DNN-based inpainting algorithms described at the end of Section 10.5.1.

Ex 10.11: Colorization. Implement the Levin, Lischinski, and Weiss (2004) colorization algorithm that is sketched out in Section 4.2.4 and Figure 4.10. If you prefer, you can implement this as a neural network (Zhang, Zhu *et al.* 2017). Find some historic monochrome photographs and some modern color ones. Write an interactive tool that lets you “pick” colors from a modern photo and paint over the old one. Tune the algorithm parameters to give you good results. Are you pleased with the results? Can you think of ways to make them look more “antique”, e.g., with softer (less saturated and edgy) colors?

(Alternative) Implement or test out one of the newer “automatic colorization” algorithms such as Zhang, Isola, and Efros (2016) or (Vondrick, Shrivastava *et al.* 2018).

Ex 10.12: Style transfer. Try some of the non-photorealistic rendering or style transfer algorithms from Sections 10.5.2–10.5.3 on your own images. Can you come up with surprising results? How about failure cases?

Chapter 11

Structure from motion and SLAM

11.1	Geometric intrinsic calibration	685
11.1.1	Vanishing points	687
11.1.2	<i>Application:</i> Single view metrology	688
11.1.3	Rotational motion	689
11.1.4	Radial distortion	691
11.2	Pose estimation	693
11.2.1	Linear algorithms	693
11.2.2	Iterative non-linear algorithms	695
11.2.3	<i>Application:</i> Location recognition	698
11.2.4	Triangulation	701
11.3	Two-frame structure from motion	703
11.3.1	Eight, seven, and five-point algorithms	703
11.3.2	Special motions and structures	708
11.3.3	Projective (uncalibrated) reconstruction	710
11.3.4	Self-calibration	712
11.3.5	<i>Application:</i> View morphing	714
11.4	Multi-frame structure from motion	715
11.4.1	Factorization	715
11.4.2	Bundle adjustment	717
11.4.3	Exploiting sparsity	719
11.4.4	<i>Application:</i> Match move	723
11.4.5	Uncertainty and ambiguities	723
11.4.6	<i>Application:</i> Reconstruction from internet photos	725
11.4.7	Global structure from motion	728
11.4.8	Constrained structure and motion	731
11.5	Simultaneous localization and mapping (SLAM)	734
11.5.1	<i>Application:</i> Autonomous navigation	737
11.5.2	<i>Application:</i> Smartphone augmented reality	739

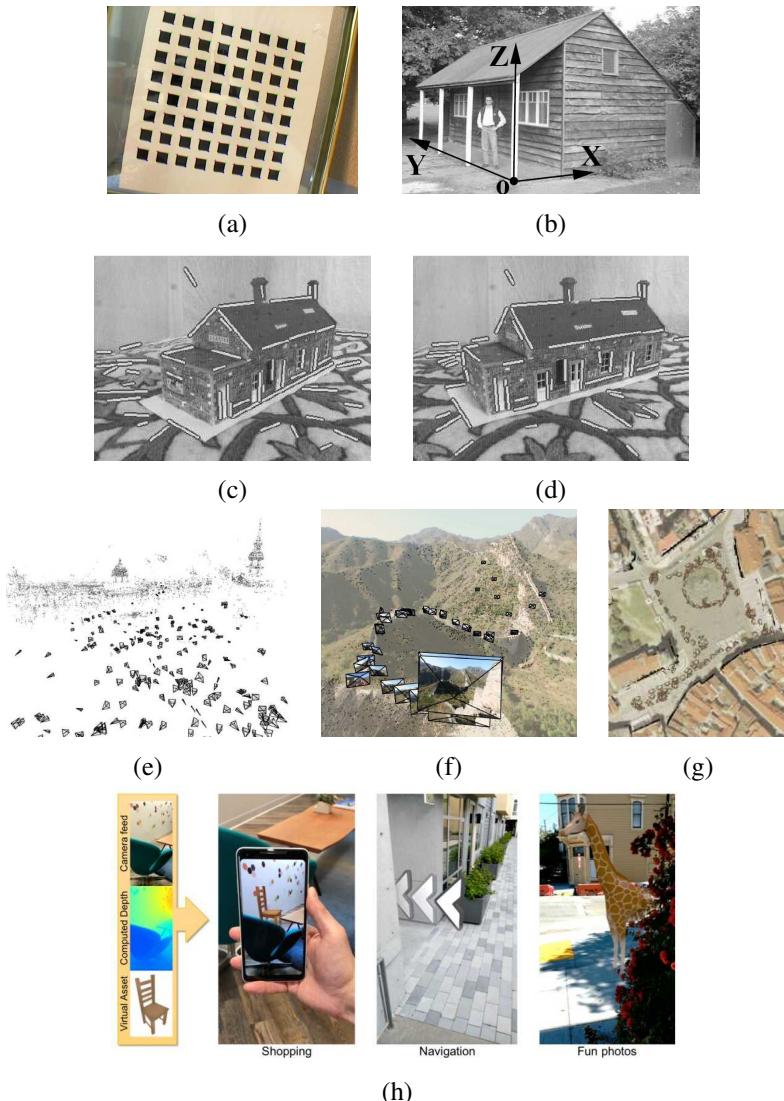


Figure 11.1 Structure from motion examples: (a) a two-dimensional calibration target (Zhang 2000) © 2000 IEEE; (b) single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer. (c–d) line matching (Schmid and Zisserman 1997) © 1997 IEEE; (e–g) 3D reconstructions of Trafalgar Square, Great Wall of China, and Prague Old Town Square (Snavely, Seitz, and Szeliski 2006) © 2006 ACM; (h) smartphone augmented reality showing real-time depth occlusion effects (Valentin, Kowdle et al. 2018) © 2018 ACM.

11.6 Additional reading	740
11.7 Exercises	743

The reconstruction of 3D models from images has been one of the central topics in computer vision since its inception (Figure 1.7). In fact, it was then believed that the construction of 3D models was a prerequisite for scene understanding and recognition (Marr 1982), although work in the last few decades has proven otherwise. However, 3D modeling has also proven to be immensely useful in applications such as virtual tourism (Section 11.4.6), autonomous navigation (Section 11.5.1), and augmented reality (Section 11.5.2).

In the last three chapters, we focused on techniques for establishing correspondences between 2D images and using these in a variety of applications such as image stitching, video enhancement, and computational photography. In this chapter, we turn to the topic of using such correspondences to build sparse 3D models of a scene and to re-localize cameras with respect to such models. While this process often involves simultaneously estimating both 3D geometry (structure) and camera pose (motion), it is commonly known (for historical reasons) as *structure from motion* (Ullman 1979).

The topics of projective geometry and structure from motion are extremely rich and some excellent textbooks and surveys have been written on them (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010; Ma, Soatto *et al.* 2012). This chapter skips over a lot of the richer material available in these books, such as the trifocal tensor and algebraic techniques for full self-calibration, and concentrates instead on the basics that we have found useful in large-scale, image-based reconstruction problems (Snavely, Seitz, and Szeliski 2006).

We begin this chapter in Section 11.1 with a review of commonly used techniques for calibrating the camera *intrinsics*, e.g., the focal length and radial distortion parameters we introduced in Sections 2.1.4–2.1.5. Next, we discuss how to estimate the *extrinsic pose* of a camera from 3D to 2D point correspondences (Section 11.2) as well as how to *triangulate* a set of 2D correspondences to estimate a point’s 3D location. We then look at the two-frame structure from motion problem (Section 11.3), which involves the determination of the *epipolar geometry* between two cameras and which can also be used to recover certain information about the camera intrinsics using self-calibration (Section 11.3.4). Section 11.4.1 looks at *factorization* approaches to simultaneously estimating structure and motion from large numbers of point tracks using orthographic approximations to the projection model. We then develop a more general and useful approach to structure from motion, namely the simultaneous *bundle adjustment* of all the camera and 3D structure parameters (Section 11.4.2). We also look at special cases that arise when there are higher-level structures, such as lines and planes, in the scene (Section 11.4.8). In the last part of this chapter (Section 11.5), we look at real-time systems for simultaneous localization and mapping (SLAM), which reconstruct a 3D world model while moving through an environment, and can be applied to both visual

navigation and augmented reality.

11.1 Geometric intrinsic calibration

As we discuss in the next section (Equations (11.14–11.15)), the computation of the internal (intrinsic) camera calibration parameters can occur simultaneously with the estimation of the (extrinsic) pose of the camera with respect to a known calibration target. This, indeed, is the “classic” approach to camera calibration used in both the photogrammetry (Slama 1980) and the computer vision (Tsai 1987) communities. In this section, we look at simpler alternative formulations that may not involve the full solution of a non-linear regression problem, the use of alternative calibration targets, and the estimation of the non-linear part of camera optics such as radial distortion. In some applications, you can use the EXIF tags associated with a JPEG image to obtain a rough estimate of a camera’s focal length and hence to initialize iterative estimation algorithms; but this technique should be used with caution as the results are often inaccurate.

Calibration patterns

The use of a calibration pattern or set of markers is one of the more reliable ways to estimate a camera’s intrinsic parameters. In photogrammetry, it is common to set up a camera in a large field looking at distant calibration targets whose exact location has been precomputed using surveying equipment (Slama 1980; Atkinson 1996; Kraus 1997). In this case, the translational component of the pose becomes irrelevant and only the camera rotation and intrinsic parameters need to be recovered.

If a smaller calibration rig needs to be used, e.g., for indoor robotics applications or for mobile robots that carry their own calibration target, it is best if the calibration object can span as much of the workspace as possible (Figure 11.2a), as planar targets often fail to accurately predict the components of the pose that lie far away from the plane. A good way to determine if the calibration has been successfully performed is to estimate the covariance in the parameters (Section 8.1.4) and then project 3D points from various points in the workspace into the image in order to estimate their 2D positional uncertainty.

If no calibration pattern is available, it is also possible to perform calibration simultaneously with structure and pose recovery (Sections 11.1.3 and 11.4.2), which is known as *self-calibration* (Faugeras, Luong, and Maybank 1992; Pollefeys, Koch, and Van Gool 1999; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). However, such an approach requires a large amount of imagery to be accurate.

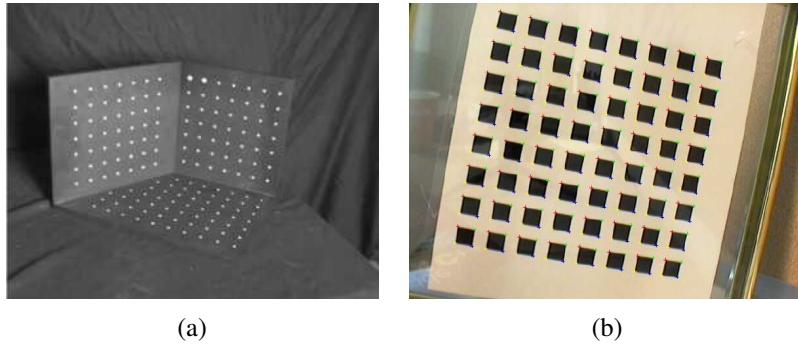


Figure 11.2 Calibration patterns: (a) a three-dimensional target (Quan and Lan 1999) © 1999 IEEE; (b) a two-dimensional target (Zhang 2000) © 2000 IEEE. Note that radial distortion needs to be removed from such images before the feature points can be used for calibration.

Planar calibration patterns

When a finite workspace is being used and accurate machining and motion control platforms are available, a good way to perform calibration is to move a planar calibration target through the workspace volume and use the known 3D point locations for calibration. This approach is sometimes called the *N-planes* calibration approach (Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée *et al.* 1992b; Grossberg and Nayar 2001) and has the advantage that each camera pixel can be mapped to a unique 3D ray in space, which takes care of both linear effects modeled by the calibration matrix \mathbf{K} and non-linear effects such as radial distortion (Section 11.1.4).

A less cumbersome but also less accurate calibration can be obtained by waving a planar calibration pattern in front of a camera (Figure 11.2b). In this case, the pattern's pose has to be recovered in conjunction with the intrinsics. In this technique, each input image is used to compute a separate homography (8.19–8.23) $\tilde{\mathbf{H}}$ mapping the plane's calibration points $(X_i, Y_i, 1)$ into image coordinates (x_i, y_i) ,

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \mathbf{K} \begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\mathbf{H}} \mathbf{p}_i, \quad (11.1)$$

where the \mathbf{r}_i are the first two columns of \mathbf{R} and \sim indicates equality up to scale. From these, Zhang (2000) shows how to form linear constraints on the nine entries in the $\mathbf{B} = \mathbf{K}^{-T}\mathbf{K}^{-1}$ matrix, from which the calibration matrix \mathbf{K} can be recovered using a matrix

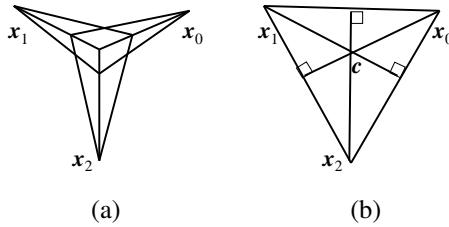


Figure 11.3 Calibration from vanishing points: (a) any pair of finite vanishing points (\hat{x}_i, \hat{x}_j) can be used to estimate the focal length; (b) the orthocenter of the vanishing point triangle gives the image center of the image c .

square root and inversion. The matrix \mathbf{B} is known as the *image of the absolute conic* (IAC) in projective geometry and is commonly used for camera calibration (Hartley and Zisserman 2004, Section 8.5). If only the focal length is being recovered, the even simpler approach of using vanishing points described below can be used instead.

11.1.1 Vanishing points

A common case for calibration that occurs often in practice is when the camera is looking at a manufactured or architectural scene with long extended rectangular patterns such as boxes or building walls. In this case, we can intersect the 2D lines corresponding to 3D parallel lines to compute their *vanishing points*, as described in Section 7.4.3, and use these to determine the intrinsic and extrinsic calibration parameters (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008).

Let us assume that we have detected two or more orthogonal vanishing points, all of which are *finite*, i.e., they are not obtained from lines that appear to be parallel in the image plane (Figure 11.3a). Let us also assume a simplified form for the calibration matrix \mathbf{K} where only the focal length is unknown (2.59). It is often safe for rough 3D modeling to assume that the optical center is at the center of the image, that the aspect ratio is 1, and that there is no skew. In this case, the projection equation for the vanishing points can be written as

$$\hat{\mathbf{x}}_i = \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ f \end{bmatrix} \sim \mathbf{R}\mathbf{p}_i = \mathbf{r}_i, \quad (11.2)$$

where \mathbf{p}_i corresponds to one of the cardinal directions $(1, 0, 0)$, $(0, 1, 0)$, or $(0, 0, 1)$, and \mathbf{r}_i

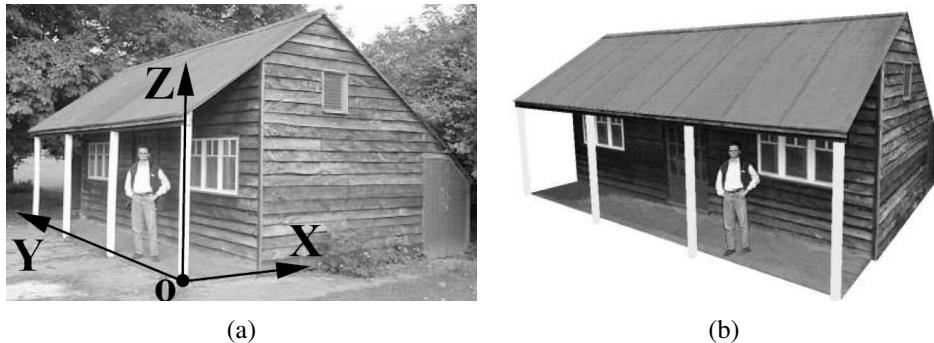


Figure 11.4 Single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer: (a) input image showing the three coordinate axes computed from the two horizontal vanishing points (which can be determined from the sidings on the shed); (b) a new view of the 3D reconstruction.

is the i th column of the rotation matrix \mathbf{R} .

From the orthogonality between columns of the rotation matrix, we have

$$\mathbf{r}_i \cdot \mathbf{r}_j \sim (x_i - c_x)(x_j - c_x) + (y_i - c_y)(y_j - c_y) + f^2 = 0, \quad i \neq j \quad (11.3)$$

from which we can obtain an estimate for f^2 . Note that the accuracy of this estimate increases as the vanishing points move closer to the center of the image. In other words, it is best to tilt the calibration pattern a decent amount around the 45° axis, as in Figure 11.3a. Once the focal length f has been determined, the individual columns of \mathbf{R} can be estimated by normalizing the left-hand side of (11.2) and taking cross products. Alternatively, the orthogonal Procrustes algorithm (8.32) can be used.

If all three vanishing points are visible and finite in the same image, it is also possible to estimate the image center as the orthocenter of the triangle formed by the three vanishing points (Caprile and Torre 1990; Hartley and Zisserman 2004, Section 8.6) (Figure 11.3b). In practice, however, it is more accurate to re-estimate any unknown intrinsic calibration parameters using non-linear least squares (11.14).

11.1.2 Application: Single view metrology

A fun application of vanishing point estimation and camera calibration is the *single view metrology* system developed by Criminisi, Reid, and Zisserman (2000). Their system allows people to interactively measure heights and other dimensions as well as to build piecewise-planar 3D models, as shown in Figure 11.4.

The first step in their system is to identify two orthogonal vanishing points on the ground plane and the vanishing point for the vertical direction, which can be done by drawing some parallel sets of lines in the image. Alternatively, automated techniques such as those discussed in Section 7.4.3 or by Schaffalitzky and Zisserman (2000) could be used. The user then marks a few dimensions in the image, such as the height of a reference object, and the system can automatically compute the height of another object. Walls and other planar impostors (geometry) can also be sketched and reconstructed.

In the formulation originally developed by Criminisi, Reid, and Zisserman (2000), the system produces an *affine* reconstruction, i.e., one that is only known up to a set of independent scaling factors along each axis. A potentially more useful system can be constructed by assuming that the camera is calibrated up to an unknown focal length, which can be recovered from orthogonal (finite) vanishing directions, as we have just described in Section 11.1.1. Once this is done, the user can indicate an origin on the ground plane and another point a known distance away. From this, points on the ground plane can be directly projected into 3D, and points above the ground plane, when paired with their ground plane projections, can also be recovered. A fully metric reconstruction of the scene then becomes possible.

Exercise 11.4 has you implement such a system and then use it to model some simple 3D scenes. Section 13.6.1 describes other, potentially multi-view, approaches to architectural reconstruction, including an interactive piecewise-planar modeling system that uses vanishing points to establish 3D line directions and plane normals (Sinha, Steedly *et al.* 2008).

11.1.3 Rotational motion

When no calibration targets or known structures are available but you can rotate the camera around its front nodal point (or, equivalently, work in a large open environment where all objects are distant), the camera can be calibrated from a set of overlapping images by assuming that it is undergoing pure rotational motion, as shown in Figure 11.5 (Stein 1995; Hartley 1997b; Hartley, Hayman *et al.* 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003). When a full 360° motion is used to perform this calibration, a very accurate estimate of the focal length f can be obtained, as the accuracy in this estimate is proportional to the total number of pixels in the resulting cylindrical panorama (Section 8.2.6) (Stein 1995; Shum and Szeliski 2000).

To use this technique, we first compute the homographies $\tilde{\mathbf{H}}_{ij}$ between all overlapping pairs of images, as explained in Equations (8.19–8.23). Then, we use the observation, first made in Equation (2.72) and explored in more detail in Equation (8.38), that each homography is related to the inter-camera rotation \mathbf{R}_{ij} through the (unknown) calibration matrices \mathbf{K}_i

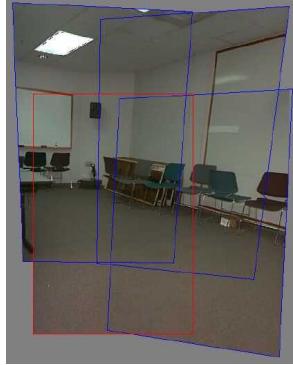


Figure 11.5 Four images taken with a hand-held camera registered using a 3D rotation motion model, which can be used to estimate the focal length of the camera (Szeliski and Shum 1997) © 2000 ACM.

and \mathbf{K}_j ,

$$\tilde{\mathbf{H}}_{ij} = \mathbf{K}_i \mathbf{R}_i \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} = \mathbf{K}_i \mathbf{R}_{ij} \mathbf{K}_j^{-1}. \quad (11.4)$$

The simplest way to obtain the calibration is to use the simplified form of the calibration matrix (2.59), where we assume that the pixels are square and the image center lies at the geometric center of the 2D pixel array, i.e., $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$. We subtract half the width and height from the original pixel coordinates so that the pixel $(x, y) = (0, 0)$ lies at the center of the image. We can then rewrite Equation (11.4) as

$$\mathbf{R}_{10} \sim \mathbf{K}_1^{-1} \tilde{\mathbf{H}}_{10} \mathbf{K}_0 \sim \begin{bmatrix} h_{00} & h_{01} & f_0^{-1} h_{02} \\ h_{10} & h_{11} & f_0^{-1} h_{12} \\ f_1 h_{20} & f_1 h_{21} & f_0^{-1} f_1 h_{22} \end{bmatrix}, \quad (11.5)$$

where h_{ij} are the elements of $\tilde{\mathbf{H}}_{10}$.

Using the orthonormality properties of the rotation matrix \mathbf{R}_{10} and the fact that the right-hand side of (11.5) is known only up to a scale, we obtain

$$h_{00}^2 + h_{01}^2 + f_0^{-2} h_{02}^2 = h_{10}^2 + h_{11}^2 + f_0^{-2} h_{12}^2 \quad (11.6)$$

and

$$h_{00} h_{10} + h_{01} h_{11} + f_0^{-2} h_{02} h_{12} = 0. \quad (11.7)$$

From this, we can compute estimates for f_0 of

$$f_0^2 = \frac{h_{12}^2 - h_{02}^2}{h_{00}^2 + h_{01}^2 - h_{10}^2 - h_{11}^2} \quad \text{if} \quad h_{00}^2 + h_{01}^2 \neq h_{10}^2 + h_{11}^2 \quad (11.8)$$

or

$$f_0^2 = -\frac{h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \quad \text{if } h_{00}h_{10} \neq -h_{01}h_{11}. \quad (11.9)$$

If neither of these conditions holds, we can also take the dot products between the first (or second) row and the third one. Similar results can be obtained for f_1 as well, by analyzing the columns of $\tilde{\mathbf{H}}_{10}$. If the focal length is the same for both images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, e.g., from different homographies, the median value can be used as the final estimate. A more general (upper-triangular) estimate of \mathbf{K} can be obtained in the case of a fixed-parameter camera $\mathbf{K}_i = \mathbf{K}$ using the technique of Hartley (1997b). Extensions to the cases of temporally varying calibration parameters and non-stationary cameras are discussed by Hartley, Hayman *et al.* (2000) and de Agapito, Hayman, and Reid (2001).

The quality of the intrinsic camera parameters can be greatly increased by constructing a full 360° panorama, as mis-estimating the focal length will result in a gap (or excessive overlap) when the first image in the sequence is stitched to itself (Figure 8.6). The resulting misalignment can be used to improve the estimate of the focal length and to re-adjust the rotation estimates, as described in Section 8.2.4. Rotating the camera by 90° around its optical axis and re-shooting the panorama is a good way to check for aspect ratio and skew pixel problems, as is generating a full hemi-spherical panorama when there is sufficient texture.

Ultimately, however, the most accurate estimate of the calibration parameters (including radial distortion) can be obtained using a full simultaneous non-linear minimization of the intrinsic and extrinsic (rotation) parameters, as described in Section 11.2.2.

11.1.4 Radial distortion

When images are taken with wide-angle lenses, it is often necessary to model *lens distortions* such as *radial distortion*. As discussed in Section 2.1.5, the radial distortion model says that coordinates in the observed images are displaced towards (*barrel* distortion) or away (*pincushion* distortion) from the image center by an amount proportional to their radial distance (Figure 2.13a–b). The simplest radial distortion models use low-order polynomials (c.f. Equation (2.78)),

$$\begin{aligned}\hat{x} &= x(1 + \kappa_1 r^2 + \kappa_2 r^4) \\ \hat{y} &= y(1 + \kappa_1 r^2 + \kappa_2 r^4),\end{aligned}\quad (11.10)$$

where $(x, y) = (0, 0)$ at the radial distortion center (2.77), $r^2 = x^2 + y^2$, and κ_1 and κ_2 are called the *radial distortion parameters* (Brown 1971; Slama 1980).¹

¹Sometimes the relationship between x and \hat{x} is expressed the other way around, i.e., using primed (final) coordinates on the right-hand side, $x = \hat{x}(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)$. This is convenient if we map image pixels into (warped)

A variety of techniques can be used to estimate the radial distortion parameters for a given lens, if the digital camera has not already done this in its capture software. One of the simplest and most useful is to take an image of a scene with a lot of straight lines, especially lines aligned with and near the edges of the image. The radial distortion parameters can then be adjusted until all of the lines in the image are straight, which is commonly called the *plumb-line method* (Brown 1971; Kang 2001; El-Melegy and Farag 2003). Exercise 11.5 gives some more details on how to implement such a technique.

Another approach is to use several overlapping images and to combine the estimation of the radial distortion parameters with the image alignment process, i.e., by extending the pipeline used for stitching in Section 8.3.1. Sawhney and Kumar (1999) use a hierarchy of motion models (translation, affine, projective) in a coarse-to-fine strategy coupled with a quadratic radial distortion correction term. They use direct (intensity-based) minimization to compute the alignment. Stein (1997) uses a feature-based approach combined with a general 3D motion model (and quadratic radial distortion), which requires more matches than a parallax-free rotational panorama but is potentially more general. More recent approaches sometimes simultaneously compute both the unknown intrinsic parameters and the radial distortion coefficients, which may include higher-order terms or more complex rational or non-parametric forms (Claus and Fitzgibbon 2005; Sturm 2005; Thirthala and Pollefeys 2005; Barreto and Daniilidis 2005; Hartley and Kang 2005; Steele and Jaynes 2006; Tardif, Sturm *et al.* 2009).

When a known calibration target is being used (Figure 11.2), the radial distortion estimation can be folded into the estimation of the other intrinsic and extrinsic parameters (Zhang 2000; Hartley and Kang 2007; Tardif, Sturm *et al.* 2009). This can be viewed as adding another stage to the general non-linear minimization pipeline shown in Figure 11.7 between the intrinsic parameter multiplication box f_C and the perspective division box f_P . (See Exercise 11.6 on more details for the case of a planar calibration target.)

Of course, as discussed in Section 2.1.5, more general models of lens distortion, such as fisheye and non-central projection, may sometimes be required. While the parameterization of such lenses may be more complicated (Section 2.1.5), the general approach of either using calibration rigs with known 3D positions or self-calibration through the use of multiple overlapping images of a scene can both be used (Hartley and Kang 2007; Tardif, Sturm, and Roy 2007). The same techniques used to calibrate for radial distortion can also be used to reduce the amount of chromatic aberration by separately calibrating each color channel and then warping the channels to put them back into alignment (Exercise 11.7).

rays and then undistort the rays to obtain 3D rays in space, i.e., if we are using inverse warping.

11.2 Pose estimation

A particular instance of feature-based alignment, which occurs very often, is estimating an object’s 3D pose from a set of 2D point projections. This *pose estimation* problem is also known as *extrinsic* calibration, as opposed to the *intrinsic* calibration of internal camera parameters such as focal length, which we discuss in Section 11.1. The problem of recovering pose from three correspondences, which is the minimal amount of information necessary, is known as the *perspective-3-point-problem* (P3P),² with extensions to larger numbers of points collectively known as PnP (Haralick, Lee *et al.* 1994; Quan and Lan 1999; Gao, Hou *et al.* 2003; Moreno-Noguer, Lepetit, and Fua 2007; Persson and Nordberg 2018).

In this section, we look at some of the techniques that have been developed to solve such problems, starting with the *direct linear transform* (DLT), which recovers a 3×4 camera matrix, followed by other “linear” algorithms, and then looking at statistically optimal iterative algorithms.

11.2.1 Linear algorithms

The simplest way to recover the pose of the camera is to form a set of rational linear equations analogous to those used for 2D motion estimation (8.19) from the camera matrix form of perspective projection (2.55–2.56),

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}} \quad (11.11)$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \quad (11.12)$$

where (x_i, y_i) are the measured 2D feature locations and (X_i, Y_i, Z_i) are the known 3D feature locations (Figure 11.6). As with (8.21), this system of equations can be solved in a linear fashion for the unknowns in the camera matrix \mathbf{P} by multiplying the denominator on both sides of the equation. Because \mathbf{P} is unknown up to a scale, we can either fix one of the entries, e.g., $p_{23} = 1$, or find the smallest singular vector of the set of linear equations. The resulting algorithm is called the *direct linear transform* (DLT) and is commonly attributed to Sutherland (1974). (For a more in-depth discussion, see Hartley and Zisserman (2004).) To compute the 12 (or 11) unknowns in \mathbf{P} , at least six correspondences between 3D and 2D locations must be known.

As with the case of estimating homographies (8.21–8.23), more accurate results for the entries in \mathbf{P} can be obtained by directly minimizing the set of Equations (11.11–11.12) using

²The “3-point” algorithms actually require a 4th point to resolve a 4-way ambiguity.

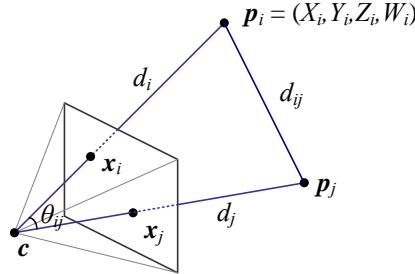


Figure 11.6 Pose estimation by the direct linear transform and by measuring visual angles and distances between pairs of points.

non-linear least squares with a small number of iterations. Note that instead of taking the ratios of the X/Z and Y/Z values as in (11.11–11.12), it is also possible to take a cross product of the 3-vector $(x_i, y_i, 1)$ image measurement and the 3-D ray (X, Y, Z) and set the three elements of this cross-product to 0. The resulting three equations, when interpreted as a set of least squares constraints, in effect compute the squared sine of the angle between the two rays.

Once the entries in \mathbf{P} have been recovered, it is possible to recover both the intrinsic calibration matrix \mathbf{K} and the rigid transformation (\mathbf{R}, \mathbf{t}) by observing from Equation (2.56) that

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \quad (11.13)$$

Because \mathbf{K} is upper-triangular (see the discussion in Section 2.1.4), both \mathbf{K} and \mathbf{R} can be obtained from the front 3×3 sub-matrix of \mathbf{P} using RQ factorization (Golub and Van Loan 1996).³

In most applications, however, we have some prior knowledge about the intrinsic calibration matrix \mathbf{K} , e.g., that the pixels are square, the skew is very small, and the image center is near the geometric center of the image (2.57–2.59). Such constraints can be incorporated into a non-linear minimization of the parameters in \mathbf{K} and (\mathbf{R}, \mathbf{t}) , as described in Section 11.2.2.

In the case where the camera is already calibrated, i.e., the matrix \mathbf{K} is known (Section 11.1), we can perform pose estimation using as few as three points (Fischler and Bolles 1981; Haralick, Lee *et al.* 1994; Quan and Lan 1999). The basic observation that these *linear PnP* (*perspective n-point*) algorithms employ is that the visual angle between any pair of 2D points \hat{x}_i and \hat{x}_j must be the same as the angle between their corresponding 3D points p_i and

³Note the unfortunate clash of terminologies: In matrix algebra textbooks, \mathbf{R} represents an upper-triangular matrix; in computer vision, \mathbf{R} is an orthogonal rotation.

\mathbf{p}_j (Figure 11.6).

A full derivation of this approach can be found in the first edition of this book (Szeliski 2010, Section 6.2.1) and also in (Quan and Lan 1999), where the authors provide accuracy results for this and other techniques, which use fewer points but require more complicated algebraic manipulations. The paper by Moreno-Noguer, Lepetit, and Fua (2007) reviews other alternatives and also gives a lower complexity algorithm that typically produces more accurate results. An even more recent paper by Terzakis and Lourakis (2020) reviews papers published in the last decade.

Unfortunately, because minimal PnP solutions can be quite noise sensitive and also suffer from *bas-relief ambiguities* (e.g., depth reversals) (Section 11.4.5), it is prudent to optimize the initial estimates from PnP using the iterative technique described in Section 11.2.2. An alternative pose estimation algorithm involves starting with a scaled orthographic projection model and then iteratively refining this initial estimate using a more accurate perspective projection model (DeMenthon and Davis 1995). The attraction of this model, as stated in the paper’s title, is that it can be implemented “in 25 lines of [Mathematica] code”.

CNN-based pose estimation

As with other areas on computer vision, deep neural networks have also been applied to pose estimation. Some representative papers include Xiang, Schmidt *et al.* (2018), Oberweger, Rad, and Lepetit (2018), Hu, Hugonot *et al.* (2019), Peng, Liu *et al.* (2019), and (Hu, Fua *et al.* 2020) for object pose estimation, and papers such as Kendall and Cipolla (2017) and Kim, Dunn, and Frahm (2017) discussed in Section 11.2.3 on location recognition. There is also a very active community around estimating pose from RGB-D images, with the most recent papers (Hagelskjær and Buch 2020; Labb  , Carpentier *et al.* 2020) evaluated on the BOP (Benchmark for 6DOF Object Pose) (Hod  , Michel *et al.* 2018).⁴

11.2.2 Iterative non-linear algorithms

The most accurate and flexible way to estimate pose is to directly minimize the squared (or robust) reprojection error for the 2D points as a function of the unknown pose parameters in (\mathbf{R}, \mathbf{t}) and optionally \mathbf{K} using non-linear least squares (Tsai 1987; Bogart 1991; Gleicher and Witkin 1992). We can write the projection equations as

$$\mathbf{x}_i = \mathbf{f}(\mathbf{p}_i; \mathbf{R}, \mathbf{t}, \mathbf{K}) \quad (11.14)$$

⁴<https://bop.felk.cvut.cz/challenges/bop-challenge-2020>, https://cmp.felk.cvut.cz/sixd/workshop_2020

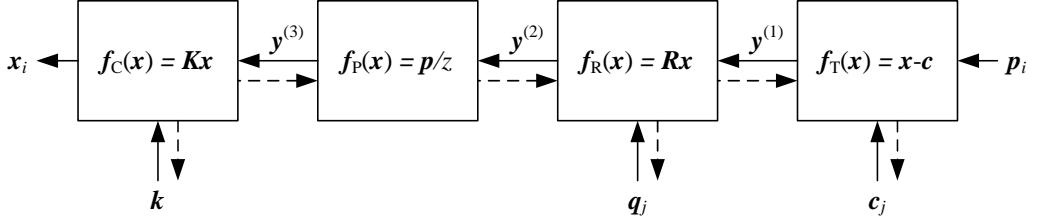


Figure 11.7 A set of chained transforms for projecting a 3D point \mathbf{p}_i to a 2D measurement \mathbf{x}_i through a series of transformations $\mathbf{f}^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

and iteratively minimize the robustified linearized reprojection errors

$$E_{\text{NLP}} = \sum_i \rho \left(\frac{\partial \mathbf{f}}{\partial \mathbf{R}} \Delta \mathbf{R} + \frac{\partial \mathbf{f}}{\partial \mathbf{t}} \Delta \mathbf{t} + \frac{\partial \mathbf{f}}{\partial \mathbf{K}} \Delta \mathbf{K} - \mathbf{r}_i \right), \quad (11.15)$$

where $\mathbf{r}_i = \tilde{\mathbf{x}}_i - \hat{\mathbf{x}}_i$ is the current residual vector (2D error in predicted position) and the partial derivatives are with respect to the unknown pose parameters (rotation, translation, and optionally calibration). The *robust loss function* ρ , which we first introduced in (4.15) in Section 4.1.3, is used to reduce the influence of outlier correspondences. Note that if full 2D covariance estimates are available for the 2D feature locations, the above squared norm can be weighted by the inverse point covariance matrix, as in Equation (8.11).

An easier to understand (and implement) version of the above non-linear regression problem can be constructed by re-writing the projection equations as a concatenation of simpler steps, each of which transforms a 4D homogeneous coordinate \mathbf{p}_i by a simple transformation such as translation, rotation, or perspective division (Figure 11.7). The resulting projection equations can be written as

$$\mathbf{y}^{(1)} = \mathbf{f}_T(\mathbf{p}_i; \mathbf{c}_j) = \mathbf{p}_i - \mathbf{c}_j, \quad (11.16)$$

$$\mathbf{y}^{(2)} = \mathbf{f}_R(\mathbf{y}^{(1)}; \mathbf{q}_j) = \mathbf{R}(\mathbf{q}_j) \mathbf{y}^{(1)}, \quad (11.17)$$

$$\mathbf{y}^{(3)} = \mathbf{f}_P(\mathbf{y}^{(2)}) = \frac{\mathbf{y}^{(2)}}{z^{(2)}}, \quad (11.18)$$

$$\mathbf{x}_i = \mathbf{f}_C(\mathbf{y}^{(3)}; \mathbf{k}) = \mathbf{K}(\mathbf{k}) \mathbf{y}^{(3)}. \quad (11.19)$$

Note that in these equations, we have indexed the camera centers \mathbf{c}_j and camera rotation quaternions \mathbf{q}_j by an index j , in case more than one pose of the calibration object is being used (see also Section 11.4.2.) We are also using the camera center \mathbf{c}_j instead of the world translation \mathbf{t}_j , as this is a more natural parameter to estimate.

The advantage of this chained set of transformations is that each one has a simple partial derivative with respect both to its parameters and to its input. Thus, once the predicted value of $\tilde{\mathbf{x}}_i$ has been computed based on the 3D point location \mathbf{p}_i and the current values of the pose parameters $(\mathbf{c}_j, \mathbf{q}_j, \mathbf{k})$, we can obtain all of the required partial derivatives using the chain rule

$$\frac{\partial \mathbf{r}_i}{\partial \mathbf{p}^{(k)}} = \frac{\partial \mathbf{r}_i}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{p}^{(k)}}, \quad (11.20)$$

where $\mathbf{p}^{(k)}$ indicates one of the parameter vectors that is being optimized. (This same “trick” is used in neural networks as part of *backpropagation*, which we presented in Figure 5.31.)

The one special case in this formulation that can be considerably simplified is the computation of the rotation update. Instead of directly computing the derivatives of the 3×3 rotation matrix $\mathbf{R}(\mathbf{q})$ as a function of the unit quaternion entries, you can prepend the incremental rotation matrix $\Delta \mathbf{R}(\boldsymbol{\omega})$ given in Equation (2.35) to the current rotation matrix and compute the partial derivative of the transform with respect to these parameters, which results in a simple cross product of the backward chaining partial derivative and the outgoing 3D vector, as explained in Equation (2.36).

Target-based augmented reality

A widely used application of pose estimation is *augmented reality*, where virtual 3D images or annotations are superimposed on top of a live video feed, either through the use of see-through glasses (a head-mounted display) or on a regular computer or mobile device screen (Azuma, Baillot *et al.* 2001; Haller, Billinghurst, and Thomas 2007; Billinghurst, Clark, and Lee 2015). In some applications, a special pattern printed on cards or in a book is tracked to perform the augmentation (Kato, Billinghurst *et al.* 2000; Billinghurst, Kato, and Poupyrev 2001). For a desktop application, a grid of dots printed on a mouse pad can be tracked by a camera embedded in an augmented mouse to give the user control of a full six degrees of freedom over their position and orientation in a 3D space (Hinckley, Sinclair *et al.* 1999). Today, tracking known targets such as movie posters is used in some phone-based augmented reality systems such as Facebook’s Spark AR.⁵

Sometimes, the scene itself provides a convenient object to track, such as the rectangle defining a desktop used in *through-the-lens camera control* (Gleicher and Witkin 1992). In outdoor locations, such as film sets, it is more common to place special markers such as brightly colored balls in the scene to make it easier to find and track them (Bogart 1991). In older applications, surveying techniques were used to determine the locations of these balls

⁵<https://sparkar.facebook.com/ar-studio>

before filming. Today, it is more common to apply structure-from-motion directly to the film footage itself (Section 11.5.2).

Exercise 8.4 has you implement a tracking and pose estimation system for augmented-reality applications.

11.2.3 *Application: Location recognition*

One of the most exciting applications of pose estimation is in the area of location recognition, which can be used both in desktop applications (“*Where did I take this holiday snap?*”) and in mobile smartphone applications. The latter case includes not only finding out your current location based on a cell-phone image, but also providing you with navigation directions or annotating your images with useful information, such as building names and restaurant reviews (i.e., a pocketable form of *augmented reality*). This problem is also often called *visual (or image-based) localization* (Se, Lowe, and Little 2002; Zhang and Kosecka 2006; Janai, Güney *et al.* 2020, Section 13.3) or *visual place recognition* (Lowry, Sünderhauf *et al.* 2015).

Some approaches to location recognition assume that the photos consist of architectural scenes for which vanishing directions can be used to pre-rectify the images for easier matching (Robertson and Cipolla 2004). Other approaches use general affine covariant interest points to perform *wide baseline matching* (Schaffalitzky and Zisserman 2002), with the winning entry on the ICCV 2005 Computer Vision Contest (Szeliski 2005) using this approach (Zhang and Kosecka 2006). The Photo Tourism system of Snavely, Seitz, and Szeliski (2006) (Section 14.1.2) was the first to apply these kinds of ideas to large-scale image matching and (implicit) location recognition from internet photo collections taken under a wide variety of viewing conditions.

The main difficulty in location recognition is in dealing with the extremely large community (user-generated) photo collections on websites such as Flickr (Philbin, Chum *et al.* 2007; Chum, Philbin *et al.* 2007; Philbin, Chum *et al.* 2008; Irschara, Zach *et al.* 2009; Turcot and Lowe 2009; Sattler, Leibe, and Kobbelt 2011, 2017) or commercially captured databases (Schindler, Brown, and Szeliski 2007; Klingner, Martin, and Roseborough 2013; Torii, Arandjelović *et al.* 2018). The prevalence of commonly appearing elements such as foliage, signs, and common architectural elements further complicates the task (Schindler, Brown, and Szeliski 2007; Jegou, Douze, and Schmid 2009; Chum and Matas 2010b; Knopp, Sivic, and Pajdla 2010; Torii, Sivic *et al.* 2013; Sattler, Havlena *et al.* 2016). Figure 7.26 shows some results on location recognition from community photo collections, while Figure 11.8 shows sample results from denser commercially acquired datasets. In the latter case, the overlap between adjacent database images can be used to verify and prune potential matches using “temporal” filtering, i.e., requiring the query image to match nearby overlap-

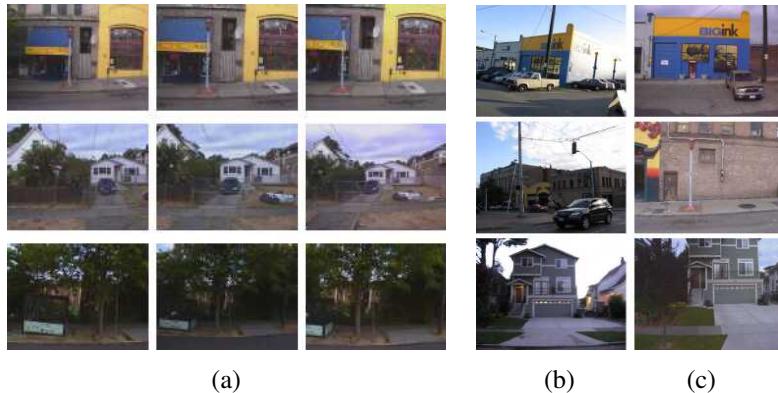


Figure 11.8 *Feature-based location recognition (Schindler, Brown, and Szeliski 2007) © 2007 IEEE: (a) three typical series of overlapping street photos; (b) handheld camera shots and (c) their corresponding database photos.*

ping database images before accepting the match. Similar ideas have been used to improve location recognition from panoramic video sequences (Levin and Szeliski 2004; Samano, Zhou, and Calway 2020) and to combine local SLAM reconstructions from image sequences with matching against a precomputed map for higher reliability (Stenborg, Sattler, and Hammarstrand 2020). Recognizing indoor locations inside buildings and shopping malls poses its own set of challenges, including textureless areas and repeated elements (Levin and Szeliski 2004; Wang, Fidler, and Urtasun 2015; Sun, Xie *et al.* 2017; Taira, Okutomi *et al.* 2018; Taira, Rocco *et al.* 2019; Lee, Ryu *et al.* 2021). The matching of ground-level to aerial images has also been studied (Kaminsky, Snavely *et al.* 2009; Shan, Wu *et al.* 2014).

Some of the initial research on location recognition was organized around the Oxford 5k and Paris 6k datasets (Philbin, Chum *et al.* 2007, 2008; Radenović, Iscen *et al.* 2018), as well as the Vienna (Irschara, Zach *et al.* 2009) and Photo Tourism (Li, Snavely, and Huttenlocher 2010) datasets, and later around the 7 scenes indoor RGB-D dataset (Shotton, Glocker *et al.* 2013) and Cambridge Landmarks (Kendall, Grimes, and Cipolla 2015). The NetVLAD paper (Arandjelovic, Gronat *et al.* 2016) was tested on Google Street View Time Machine data. Currently, the most widely used visual localization datasets are collected at the Long-Term Visual Localization Benchmark⁶ and include such datasets as Aachen Day-Night (Sattler, Maddern *et al.* 2018) and InLoc (Taira, Okutomi *et al.* 2018). And while most localization systems work from collections of ground-level images, it is also possible to re-localize based on textured digital elevation (terrain) models for outdoor (non-city) applications (Baatz, Saurer *et*

⁶<https://www.visuallocalization.net>

al. 2012; Brejcha, Lukáč *et al.* 2020).

Some of the most recent approaches to localization use deep networks to generate feature descriptors (Arandjelovic, Gronat *et al.* 2016; Kim, Dunn, and Frahm 2017; Torii, Arandjelović *et al.* 2018; Radenović, Tolias, and Chum 2019; Yang, Kien Nguyen *et al.* 2019; Sarlin, Unagar *et al.* 2021), perform large-scale instance retrieval (Radenović, Tolias, and Chum 2019; Cao, Araujo, and Sim 2020; Ng, Balntas *et al.* 2020; Tolias, Jenicek, and Chum 2020; Pion, Humenberger *et al.* 2020 and Section 6.2.3), map images to 3D scene coordinates (Brachmann and Rother 2018), or perform end-to-end scene coordinate regression (Shotton, Glocker *et al.* 2013), absolute pose regression (APR) (Kendall, Grimes, and Cipolla 2015; Kendall and Cipolla 2017), or relative pose regression (RPR) (Melekhov, Ylioinas *et al.* 2017; Balntas, Li, and Prisacariu 2018). Recent evaluations of these techniques have shown that classical approaches based on feature matching followed by geometric pose optimization typically outperform pose regression approaches in terms of accuracy and generalization (Sattler, Zhou *et al.* 2019; Zhou, Sattler *et al.* 2019; Ding, Wang *et al.* 2019; Lee, Ryu *et al.* 2021; Sarlin, Unagar *et al.* 2021).

The Long-Term Visual Localization benchmark has a leaderboard listing the best-performing localization systems. In the CVPR 2020 workshop and challenge, some of the winning entries were based on recent detectors, descriptors, and matchers such as SuperGlue (Sarlin, DeTone *et al.* 2020), ASLFeat (Luo, Zhou *et al.* 2020), and R2D2 (Revaud, Weinzaepfel *et al.* 2019). Other systems that did well include HF-Net (Sarlin, Cadena *et al.* 2019), ONavi (Fan, Zhou *et al.* 2020), and D2-Net (Dusmanu, Rocco *et al.* 2019). An even more recent trend is to use DNNs or transformers to establish dense coarse-to-fine matches (Jiang, Trulls *et al.* 2021; Sun, Shen *et al.* 2021).

Another variant on location recognition is the automatic discovery of *landmarks*, i.e., frequently photographed objects and locations. Simon, Snavely, and Seitz (2007) show how these kinds of objects can be discovered simply by analyzing the matching graph constructed as part of the 3D modeling process in Photo Tourism. More recent work has extended this approach to larger datasets using efficient clustering techniques (Philbin and Zisserman 2008; Li, Wu *et al.* 2008; Chum, Philbin, and Zisserman 2008; Chum and Matas 2010a; Arandjelović and Zisserman 2012), combining meta-data such as GPS and textual tags with visual search (Quack, Leibe, and Van Gool 2008; Crandall, Backstrom *et al.* 2009; Li, Snavely *et al.* 2012), and using multiple descriptors to obtain real-time performance in micro aerial vehicle navigation (Lim, Sinha *et al.* 2012). It is now even possible to automatically associate object tags with images based on their co-occurrence in multiple loosely tagged images (Simon and Seitz 2008; Gammeter, Bossard *et al.* 2009).

The concept of organizing the world’s photo collections by location has even been re-

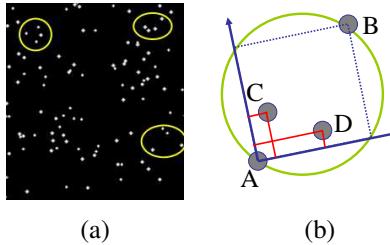


Figure 11.9 Locating star fields using astrometry, <https://astrometry.net>. (a) Input star field and some selected star quads. (b) The 2D coordinates of stars C and D are encoded relative to the unit square defined by A and B.

cently extended to organizing all of the universe’s (astronomical) photos in an application called *astrometry*.⁷ The technique used to match any two star fields is to take quadruplets of nearby stars (a pair of stars and another pair inside their diameter) to form a 30-bit *geometric hash* by encoding the relative positions of the second pair of points using the inscribed square as the reference frame, as shown in Figure 11.9. Traditional information retrieval techniques (k-d trees built for different parts of a sky atlas) are then used to find matching quads as potential star field location hypotheses, which can then be verified using a similarity transform.

11.2.4 Triangulation

The problem of determining a point’s 3D position from a set of corresponding image locations and known camera positions is known as *triangulation*. This problem is the converse of the pose estimation problem we studied in Section 11.2.

One of the simplest ways to solve this problem is to find the 3D point \mathbf{p} that lies closest to all of the 3D rays corresponding to the 2D matching feature locations $\{\mathbf{x}_j\}$ observed by cameras $\{\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]\}$, where $\mathbf{t}_j = -\mathbf{R}_j\mathbf{c}_j$ and \mathbf{c}_j is the j th camera center (2.55–2.56). As you can see in Figure 11.10, these rays originate at \mathbf{c}_j in a direction $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^{-1}\mathbf{K}_j^{-1}\mathbf{x}_j)$, where $\mathcal{N}(\mathbf{v})$ normalizes a vector \mathbf{v} to unit length. The nearest point to \mathbf{p} on this ray, which we denote as $\mathbf{q}_j = \mathbf{c}_j + d_j\hat{\mathbf{v}}_j$, minimizes the distance

$$\|\mathbf{q}_j - \mathbf{p}\|^2 = \|\mathbf{c}_j + d_j\hat{\mathbf{v}}_j - \mathbf{p}\|^2, \quad (11.21)$$

which has a minimum at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$. Hence,

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_\parallel, \quad (11.22)$$

⁷<https://astrometry.net>

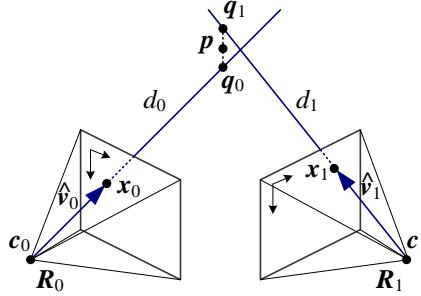


Figure 11.10 3D point triangulation by finding the point \mathbf{p} that lies nearest to all of the optical rays $\mathbf{c}_j + d_j \hat{\mathbf{v}}_j$.

in the notation of Equation (2.29), and the squared distance between \mathbf{p} and \mathbf{q}_j is

$$r_j^2 = \|(\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)\|^2 = \|(\mathbf{p} - \mathbf{c}_j)_\perp\|^2. \quad (11.23)$$

The optimal value for \mathbf{p} , which lies closest to all of the rays, can be computed as a regular least squares problem by summing over all the r_j^2 and finding the optimal value of \mathbf{p} ,

$$\mathbf{p} = \left[\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T) \right]^{-1} \left[\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T) \mathbf{c}_j \right]. \quad (11.24)$$

An alternative formulation, which is more statistically optimal and which can produce significantly better estimates if some of the cameras are closer to the 3D point than others, is to minimize the residual in the measurement equations

$$x_j = \frac{p_{00}^{(j)} X + p_{01}^{(j)} Y + p_{02}^{(j)} Z + p_{03}^{(j)} W}{p_{20}^{(j)} X + p_{21}^{(j)} Y + p_{22}^{(j)} Z + p_{23}^{(j)} W} \quad (11.25)$$

$$y_j = \frac{p_{10}^{(j)} X + p_{11}^{(j)} Y + p_{12}^{(j)} Z + p_{13}^{(j)} W}{p_{20}^{(j)} X + p_{21}^{(j)} Y + p_{22}^{(j)} Z + p_{23}^{(j)} W}, \quad (11.26)$$

where (x_j, y_j) are the measured 2D feature locations and $\{p_{00}^{(j)} \dots p_{23}^{(j)}\}$ are the known entries in camera matrix \mathbf{P}_j (Sutherland 1974).

As with Equations (8.21, 11.11, and 11.12), this set of non-linear equations can be converted into a linear least squares problem by multiplying both sides of the denominator, again resulting in the direct linear transform (DLT) formulation. Note that if we use homogeneous coordinates $\mathbf{p} = (X, Y, Z, W)$, the resulting set of equations is homogeneous and is

best solved as a singular value decomposition (SVD) or eigenvalue problem (looking for the smallest singular vector or eigenvector). If we set $W = 1$, we can use regular linear least squares, but the resulting system may be singular or poorly conditioned, i.e., if all of the viewing rays are parallel, as occurs for points far away from the camera.

For this reason, it is generally preferable to parameterize 3D points using homogeneous coordinates, especially if we know that there are likely to be points at greatly varying distances from the cameras. Of course, minimizing the set of observations (11.25–11.26) using non-linear least squares, as described in (8.14 and 8.23), is preferable to using linear least squares, regardless of the representation chosen.

For the case of two observations, it turns out that the location of the point \mathbf{p} that exactly minimizes the true reprojection error (11.25–11.26) can be computed using the solution of degree six polynomial equations (Hartley and Sturm 1997). Another problem to watch out for with triangulation is the issue of *cheirality*, i.e., ensuring that the reconstructed points lie in front of all the cameras (Hartley 1998). While this cannot always be guaranteed, a useful heuristic is to take the points that lie behind the cameras because their rays are diverging (imagine Figure 11.10 where the rays were pointing *away* from each other) and to place them on the plane at infinity by setting their W values to 0.

11.3 Two-frame structure from motion

So far in our study of 3D reconstruction, we have always assumed that either the 3D point positions or the 3D camera poses are known in advance. In this section, we take our first look at *structure from motion*, which is the simultaneous recovery of 3D structure and pose from image correspondences. In particular, we examine techniques that operate on just two frames with point correspondences. We divide this section into the study of classic “ n -point” algorithms, special (degenerate) cases, projective (uncalibrated) reconstruction, and self-calibration for cameras whose intrinsic calibrations are unknown.

11.3.1 Eight, seven, and five-point algorithms

Consider Figure 11.11, which shows a 3D point \mathbf{p} being viewed from two cameras whose relative position can be encoded by a rotation \mathbf{R} and a translation \mathbf{t} . As we do not know anything about the camera positions, without loss of generality, we can set the first camera at the origin $\mathbf{c}_0 = \mathbf{0}$ and at a canonical orientation $\mathbf{R}_0 = \mathbf{I}$.

The 3D point $\mathbf{p}_0 = d_0 \hat{\mathbf{x}}_0$ observed in the first image at location $\hat{\mathbf{x}}_0$ and at a z distance of

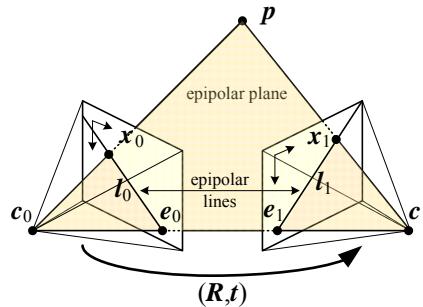


Figure 11.11 Epipolar geometry: The vectors $\mathbf{t} = \mathbf{c}_1 - \mathbf{c}_0$, $\mathbf{p} - \mathbf{c}_0$ and $\mathbf{p} - \mathbf{c}_1$ are co-planar and define the basic epipolar constraint expressed in terms of the pixel measurements \mathbf{x}_0 and \mathbf{x}_1 .

d_0 is mapped into the second image by the transformation

$$d_1 \hat{\mathbf{x}}_1 = \mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t} = \mathbf{R}(d_0 \hat{\mathbf{x}}_0) + \mathbf{t}, \quad (11.27)$$

where $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1} \mathbf{x}_j$ are the (local) ray direction vectors. Taking the cross product of the two (interchanged) sides with \mathbf{t} in order to annihilate it on the right-hand side yields⁸

$$d_1 [\mathbf{t}]_{\times} \hat{\mathbf{x}}_1 = d_0 [\mathbf{t}]_{\times} \mathbf{R} \hat{\mathbf{x}}_0. \quad (11.28)$$

Taking the dot product of both sides with $\hat{\mathbf{x}}_1$ yields

$$d_0 \hat{\mathbf{x}}_1^T ([\mathbf{t}]_{\times} \mathbf{R}) \hat{\mathbf{x}}_0 = d_1 \hat{\mathbf{x}}_1^T [\mathbf{t}]_{\times} \hat{\mathbf{x}}_1 = 0, \quad (11.29)$$

because the right-hand side is a triple product with two identical entries. (Another way to say this is that the cross product matrix $[\mathbf{t}]_{\times}$ is skew symmetric and returns 0 when pre- and post-multiplied by the same vector.)

We therefore arrive at the basic *epipolar constraint*

$$\hat{\mathbf{x}}_1^T \mathbf{E} \hat{\mathbf{x}}_0 = 0, \quad (11.30)$$

where

$$\mathbf{E} = [\mathbf{t}]_{\times} \mathbf{R} \quad (11.31)$$

is called the *essential matrix* (Longuet-Higgins 1981).

⁸The cross-product operator $[\cdot]_{\times}$ was introduced in (2.32).

An alternative way to derive the epipolar constraint is to notice that, for the cameras to be oriented so that the rays $\hat{\mathbf{x}}_0$ and $\hat{\mathbf{x}}_1$ intersect in 3D at point \mathbf{p} , the vectors connecting the two camera centers $\mathbf{c}_1 - \mathbf{c}_0 = -\mathbf{R}_1^{-1}\mathbf{t}$ and the rays corresponding to pixels \mathbf{x}_0 and \mathbf{x}_1 , namely $\mathbf{R}_j^{-1}\hat{\mathbf{x}}_j$, must be co-planar. This requires that the triple product

$$(\hat{\mathbf{x}}_0, \mathbf{R}^{-1}\hat{\mathbf{x}}_1, -\mathbf{R}^{-1}\mathbf{t}) = (\mathbf{R}\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, -\mathbf{t}) = \hat{\mathbf{x}}_1 \cdot (\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0) = \hat{\mathbf{x}}_1^T([\mathbf{t}]_{\times} \mathbf{R})\hat{\mathbf{x}}_0 = 0. \quad (11.32)$$

Notice that the essential matrix \mathbf{E} maps a point $\hat{\mathbf{x}}_0$ in image 0 into a line $\mathbf{l}_1 = \mathbf{E}\hat{\mathbf{x}}_0 = [\mathbf{t}]_{\times} \mathbf{R}\hat{\mathbf{x}}_0$ in image 1, because $\hat{\mathbf{x}}_1^T \mathbf{l}_1 = 0$ (Figure 11.11). All such lines must pass through the second *epipole* \mathbf{e}_1 , which is therefore defined as the left singular vector of \mathbf{E} with a 0 singular value, or, equivalently, the projection of the vector \mathbf{t} into image 1. The dual (transpose) of these relationships gives us the epipolar line in the first image as $\mathbf{l}_0 = \mathbf{E}^T \hat{\mathbf{x}}_1$ and \mathbf{e}_0 as the zero-value right singular vector of \mathbf{E} .

Eight-point algorithm. Given this fundamental relationship (11.30), how can we use it to recover the camera motion encoded in the essential matrix \mathbf{E} ? If we have N corresponding measurements $\{(\mathbf{x}_{i0}, \mathbf{x}_{i1})\}$, we can form N homogeneous equations in the nine elements of $\mathbf{E} = \{e_{00} \dots e_{22}\}$,

$$\begin{array}{lllll} x_{i0}x_{i1}e_{00} & +y_{i0}x_{i1}e_{01} & +x_{i1}e_{02} & + \\ x_{i0}y_{i1}e_{00} & +y_{i0}y_{i1}e_{11} & +y_{i1}e_{12} & + \\ x_{i0}e_{20} & +y_{i0}e_{21} & +e_{22} & = 0 \end{array} \quad (11.33)$$

where $\mathbf{x}_{ij} = (x_{ij}, y_{ij}, 1)$. This can be written more compactly as

$$[\mathbf{x}_{i1} \ \mathbf{x}_{i0}^T] \otimes \mathbf{E} = \mathbf{Z}_i \otimes \mathbf{E} = \mathbf{z}_i \cdot \mathbf{f} = 0, \quad (11.34)$$

where \otimes indicates an element-wise multiplication and summation of matrix elements, and \mathbf{z}_i and \mathbf{f} are the vectorized forms of the $\mathbf{Z}_i = \hat{\mathbf{x}}_{i1}\hat{\mathbf{x}}_{i0}^T$ and \mathbf{E} matrices.⁹ Given $N \geq 8$ such equations, we can compute an estimate (up to scale) for the entries in \mathbf{E} using an SVD.

In the presence of noisy measurements, how close is this estimate to being statistically optimal? If you look at the entries in (11.33), you can see that some entries are the products of image measurements such as $x_{i0}y_{i1}$ and others are direct image measurements (or even the identity). If the measurements have comparable noise, the terms that are products of measurements have their noise amplified by the other element in the product, which can lead to very poor scaling, e.g., an inordinately large influence of points with large coordinates (far away from the image center).

⁹We use \mathbf{f} instead of \mathbf{e} to denote the vectorized form of \mathbf{E} to avoid confusion with the epipoles \mathbf{e}_j .

To counteract this trend, Hartley (1997a) suggests that the point coordinates should be translated and scaled so that their centroid lies at the origin and their variance is unity, i.e.,

$$\tilde{x}_i = s(x_i - \mu_x) \quad (11.35)$$

$$\tilde{y}_i = s(y_i - \mu_y) \quad (11.36)$$

such that $\sum_i \tilde{x}_i = \sum_i \tilde{y}_i = 0$ and $\sum_i \tilde{x}_i^2 + \sum_i \tilde{y}_i^2 = 2n$, where n is the number of points.¹⁰

Once the essential matrix $\tilde{\mathbf{E}}$ has been computed from the transformed coordinates $\{(\tilde{\mathbf{x}}_{i0}, \tilde{\mathbf{x}}_{i1})\}$, where $\tilde{\mathbf{x}}_{ij} = \mathbf{T}_j \hat{\mathbf{x}}_{ij}$ and \mathbf{T}_j is the 3×3 matrix that implements the shift and scale operations in (11.35–11.36), the original essential matrix \mathbf{E} can be recovered as

$$\mathbf{E} = \mathbf{T}_1^T \tilde{\mathbf{E}} \mathbf{T}_0. \quad (11.37)$$

In his paper, Hartley (1997a) compares the improvement due to his re-normalization strategy to alternative distance measures proposed by others such as Zhang (1998a,b) and concludes that his simple re-normalization in most cases is as effective as (or better than) alternative techniques. Torr and Fitzgibbon (2004) recommend a variant on this algorithm where the norm of the upper 2×2 sub-matrix of \mathbf{E} is set to 1 and show that it has even better stability with respect to 2D coordinate transformations.

7-point algorithm. Because \mathbf{E} is rank-deficient, it turns out that we actually only need seven correspondences of the form of Equation (11.34) instead of eight to estimate this matrix (Hartley 1994a; Torr and Murray 1997; Hartley and Zisserman 2004). The advantage of using fewer correspondences inside a RANSAC robust fitting stage is that fewer random samples need to be generated. From this set of seven homogeneous equations (which we can stack into a 7×9 matrix for SVD analysis), we can find two independent vectors, say \mathbf{f}_0 and \mathbf{f}_1 such that $\mathbf{z}_i \cdot \mathbf{f}_j = 0$. These two vectors can be converted back into 3×3 matrices \mathbf{E}_0 and \mathbf{E}_1 , which span the solution space for

$$\mathbf{E} = \alpha \mathbf{E}_0 + (1 - \alpha) \mathbf{E}_1. \quad (11.38)$$

To find the correct value of α , we observe that \mathbf{E} has a zero determinant, as it is rank deficient, and hence

$$|\alpha \mathbf{E}_0 + (1 - \alpha) \mathbf{E}_1| = 0. \quad (11.39)$$

¹⁰More precisely, Hartley (1997a) suggests scaling the points “so that the average distance from the origin is equal to $\sqrt{2}$ ” but the heuristic of unit variance is faster to compute (does not require per-point square roots) and should yield comparable improvements.

This gives us a cubic equation in α , which has either one or three solutions (roots). Substituting these values into (11.38) to obtain \mathbf{E} , we can test this essential matrix against other unused feature correspondences to select the correct one.

The normalized “eight-point algorithm” (Hartley 1997a) and seven-point algorithm described above are not the only way to estimate the camera motion from correspondences. Additional variants include a five-point algorithm that requires finding the roots of a 10th degree polynomial (Nistér 2004) as well as variants that handle special (restricted) motions or scene structures, as discussed later on in this section. Because such algorithms use fewer points to compute their estimates, they are less sensitive to outliers when used as part of a random sampling (RANSAC) strategy.¹¹

Recovering \mathbf{t} and \mathbf{R} . Once an estimate for the essential matrix \mathbf{E} has been recovered, the direction of the translation vector \mathbf{t} can be estimated. Note that the absolute distance between the two cameras can never be recovered from pure image measurements alone, regardless of how many cameras or points are used. Knowledge about absolute camera and point positions or distances, often called *ground control points* in photogrammetry, is always required to establish the final scale, position, and orientation.

To estimate this direction $\hat{\mathbf{t}}$, observe that under ideal noise-free conditions, the essential matrix \mathbf{E} is singular, i.e., $\hat{\mathbf{t}}^T \mathbf{E} = 0$. This singularity shows up as a singular value of 0 when an SVD of \mathbf{E} is performed,

$$\mathbf{E} = [\hat{\mathbf{t}}]_{\times} \mathbf{R} = \mathbf{U} \Sigma \mathbf{V}^T = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \hat{\mathbf{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix}. \quad (11.40)$$

When \mathbf{E} is computed from noisy measurements, the singular vector associated with the smallest singular value gives us $\hat{\mathbf{t}}$. (The other two singular values should be similar but are not, in general, equal to 1 because \mathbf{E} is only computed up to an unknown scale.)

Once $\hat{\mathbf{t}}$ has been recovered, how can we estimate the corresponding rotation matrix \mathbf{R} ? Recall that the cross-product operator $[\hat{\mathbf{t}}]_{\times}$ (2.32) projects a vector onto a set of orthogonal basis vectors that include $\hat{\mathbf{t}}$, zeros out the $\hat{\mathbf{t}}$ component, and rotates the other two by 90°,

$$[\hat{\mathbf{t}}]_{\times} = \mathbf{S} \mathbf{Z} \mathbf{R}_{90^\circ} \mathbf{S}^T = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \hat{\mathbf{t}}^T \end{bmatrix}, \quad (11.41)$$

¹¹You can find an experimental comparison of a number of RANSAC variants at <https://opencv.org/evaluating-opencvs-new-ransacs/>.

where $\hat{\mathbf{t}} = \mathbf{s}_0 \times \mathbf{s}_1$. From Equations (11.40 and 11.41), we get

$$\mathbf{E} = [\hat{\mathbf{t}}]_{\times} \mathbf{R} = \mathbf{S} \mathbf{Z} \mathbf{R}_{90^\circ} \mathbf{S}^T \mathbf{R} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T, \quad (11.42)$$

from which we can conclude that $\mathbf{S} = \mathbf{U}$. Recall that for a noise-free essential matrix, ($\boldsymbol{\Sigma} = \mathbf{Z}$), and hence

$$\mathbf{R}_{90^\circ} \mathbf{U}^T \mathbf{R} = \mathbf{V}^T \quad (11.43)$$

and

$$\mathbf{R} = \mathbf{U} \mathbf{R}_{90^\circ}^T \mathbf{V}^T. \quad (11.44)$$

Unfortunately, we only know both \mathbf{E} and $\hat{\mathbf{t}}$ up to a sign. Furthermore, the matrices \mathbf{U} and \mathbf{V} are not guaranteed to be rotations (you can flip both their signs and still get a valid SVD). For this reason, we have to generate all four possible rotation matrices

$$\mathbf{R} = \pm \mathbf{U} \mathbf{R}_{\pm 90^\circ}^T \mathbf{V}^T \quad (11.45)$$

and keep the two whose determinant $|\mathbf{R}| = 1$. To disambiguate between the remaining pair of potential rotations, which form a *twisted pair* (Hartley and Zisserman 2004, p. 259), we need to pair them with both possible signs of the translation direction $\pm \hat{\mathbf{t}}$ and select the combination for which the largest number of points is seen in front of both cameras.¹²

The property that points must lie in front of the camera, i.e., at a positive distance along the viewing rays emanating from the camera, is known as *cheirality* (Hartley 1998). In addition to determining the signs of the rotation and translation, as described above, the cheirality (sign of the distances) of the points in a reconstruction can be used inside a RANSAC procedure (along with the reprojection errors) to distinguish between likely and unlikely configurations.¹³ cheirality can also be used to transform projective reconstructions (Sections 11.3.3 and 11.3.4) into *quasi-affine* reconstructions (Hartley 1998).

11.3.2 Special motions and structures

In certain situations, specially tailored algorithms can take advantage of known (or guessed) camera arrangements or 3D structures.

¹²In the noise-free case, a single point suffices. It is safer, however, to test all or a sufficient subset of points, downweighting the ones that lie close to the plane at infinity, for which it is easy to get depth reversals.

¹³Note that as points get further away from a camera, i.e., closer toward the plane at infinity, errors in cheirality become more likely.

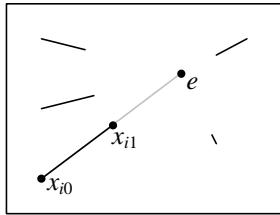


Figure 11.12 Pure translational camera motion results in visual motion where all the points move towards (or away from) a common focus of expansion (FOE) e . They therefore satisfy the triple product condition $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{e}) = \mathbf{e} \cdot (\mathbf{x}_0 \times \mathbf{x}_1) = 0$.

Pure translation (known rotation). In the case where we know the rotation, we can pre-rotate the points in the second image to match the viewing direction of the first. The resulting set of 3D points all move towards (or away from) the *focus of expansion* (FOE), as shown in Figure 11.12.¹⁴ The resulting essential matrix \mathbf{E} is (in the noise-free case) skew symmetric and so can be estimated more directly by setting $e_{ij} = -e_{ji}$ and $e_{ii} = 0$ in (11.33). Two points with non-zero parallax now suffice to estimate the FOE.

A more direct derivation of the FOE estimate can be obtained by minimizing the triple product

$$\sum_i (\mathbf{x}_{i0}, \mathbf{x}_{i1}, \mathbf{e})^2 = \sum_i ((\mathbf{x}_{i0} \times \mathbf{x}_{i1}) \cdot \mathbf{e})^2, \quad (11.46)$$

which is equivalent to finding the null space for the set of equations

$$(y_{i0} - y_{i1})e_0 + (x_{i1} - x_{i0})e_1 + (x_{i0}y_{i1} - y_{i0}x_{i1})e_2 = 0. \quad (11.47)$$

Note that, as in the eight-point algorithm, it is advisable to normalize the 2D points to have unit variance before computing this estimate.

In situations where a large number of points at infinity are available, e.g., when shooting outdoor scenes or when the camera motion is small compared to distant objects, this suggests an alternative RANSAC strategy for estimating the camera motion. First, pick a pair of points to estimate a rotation, hoping that both of the points lie at infinity (very far from the camera). Then, compute the FOE and check whether the residual error is small (indicating agreement with this rotation hypothesis) and whether the motions towards or away from the epipole (FOE) are all in the same direction (ignoring very small motions, which may be noise-contaminated).

¹⁴Fans of *Star Trek* and *Star Wars* will recognize this as the “jump to hyperdrive” visual effect.

Pure rotation. The case of pure rotation results in a degenerate estimate of the essential matrix \mathbf{E} and of the translation direction $\hat{\mathbf{t}}$. Consider first the case of the rotation matrix being known. The estimates for the FOE will be degenerate, because $\mathbf{x}_{i0} \approx \mathbf{x}_{i1}$, and hence (11.47), is degenerate. A similar argument shows that the equations for the essential matrix (11.33) are also rank-deficient.

This suggests that it might be prudent before computing a full essential matrix to first compute a rotation estimate \mathbf{R} using (8.32), potentially with just a small number of points, and then compute the residuals after rotating the points before proceeding with a full \mathbf{E} computation.

Dominant planar structure. When a dominant plane is present in the scene, DEGENSAC, which tests whether too many correspondences are co-planar, can be used to recover the fundamental matrix more reliably than the seven-point algorithm (Chum, Werner, and Matas 2005).

As you can tell from the previous special cases, there exist many different specialized cases of two-frame structure-from-motion as well as many alternative appropriate techniques. The OpenGV library developed by Kneip and Furgale (2014) contains open-source implementations of many of these algorithms.¹⁵

11.3.3 Projective (uncalibrated) reconstruction

In many cases, such as when trying to build a 3D model from internet or legacy photos taken by unknown cameras without any EXIF tags, we do not know ahead of time the intrinsic calibration parameters associated with the input images. In such situations, we can still estimate a two-frame reconstruction, although the true metric structure may not be available, e.g., orthogonal lines or planes in the world may not end up being reconstructed as orthogonal.

Consider the derivations we used to estimate the essential matrix \mathbf{E} (11.30–11.32). In the uncalibrated case, we do not know the calibration matrices \mathbf{K}_j , so we cannot use the normalized ray directions $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$. Instead, we have access only to the image coordinates \mathbf{x}_j , and so the essential matrix equation (11.30) becomes

$$\hat{\mathbf{x}}_1^T \mathbf{E} \hat{\mathbf{x}}_1 = \mathbf{x}_1^T \mathbf{K}_1^{-T} \mathbf{E} \mathbf{K}_0^{-1} \mathbf{x}_0 = \mathbf{x}_1^T \mathbf{F} \mathbf{x}_0 = 0, \quad (11.48)$$

where

$$\mathbf{F} = \mathbf{K}_1^{-T} \mathbf{E} \mathbf{K}_0^{-1} \quad (11.49)$$

¹⁵<https://laurentkneip.github.io/opengv>

is called the *fundamental matrix* (Faugeras 1992; Hartley, Gupta, and Chang 1992; Hartley and Zisserman 2004).

Like the essential matrix, the fundamental matrix is (in principle) rank two,

$$\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \mathbf{e}_1 \end{bmatrix} \begin{bmatrix} \sigma_0 & & \\ & \sigma_1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{e}_0^T \end{bmatrix}. \quad (11.50)$$

Its smallest left singular vector indicates the epipole \mathbf{e}_1 in the image 1 and its smallest right singular vector is \mathbf{e}_0 (Figure 11.11). The fundamental matrix can be factored into a skew-symmetric cross product matrix $[\mathbf{e}]_\times$ and a homography $\tilde{\mathbf{H}}$,

$$\mathbf{F} = [\mathbf{e}]_\times \tilde{\mathbf{H}}. \quad (11.51)$$

The homography $\tilde{\mathbf{H}}$, which in principle from (11.49) should equal

$$\tilde{\mathbf{H}} = \mathbf{K}_1^{-T} \mathbf{R} \mathbf{K}_0^{-1}, \quad (11.52)$$

cannot be uniquely recovered from \mathbf{F} , as any homography of the form $\tilde{\mathbf{H}}' = \tilde{\mathbf{H}} + \mathbf{e}\mathbf{v}^T$ results in the same \mathbf{F} matrix. (Note that $[\mathbf{e}]_\times$ annihilates any multiple of \mathbf{e} .)

Any one of these valid homographies $\tilde{\mathbf{H}}$ maps some plane in the scene from one image to the other. It is not possible to tell in advance which one it is without either selecting four or more co-planar correspondences to compute $\tilde{\mathbf{H}}$ as part of the \mathbf{F} estimation process (in a manner analogous to guessing a rotation for \mathbf{E}) or mapping all points in one image through $\tilde{\mathbf{H}}$ and seeing which ones line up with their corresponding locations in the other. The resulting representation is often referred to as *plane plus parallax* (Kumar, Anandan, and Hanna 1994; Sawhney 1994) and is described in more detail in Section 2.1.4.

To create a *projective* reconstruction of the scene, we can pick any valid homography $\tilde{\mathbf{H}}$ that satisfies Equation (11.49). For example, following a technique analogous to Equations (11.40–11.44), we get

$$\mathbf{F} = [\mathbf{e}]_\times \tilde{\mathbf{H}} = \mathbf{S} \mathbf{Z} \mathbf{R}_{90^\circ} \mathbf{S}^T \tilde{\mathbf{H}} = \mathbf{U} \Sigma \mathbf{V}^T \quad (11.53)$$

and hence

$$\tilde{\mathbf{H}} = \mathbf{U} \mathbf{R}_{90^\circ}^T \hat{\Sigma} \mathbf{V}^T, \quad (11.54)$$

where $\hat{\Sigma}$ is the singular value matrix with the smallest value replaced by a reasonable alternative (say, the middle value).¹⁶ We can then form a pair of camera matrices

$$\mathbf{P}_0 = [\mathbf{I} | \mathbf{0}] \quad \text{and} \quad \mathbf{P}_0 = [\tilde{\mathbf{H}} | \mathbf{e}], \quad (11.55)$$

¹⁶Hartley and Zisserman (2004, p. 256) recommend using $\tilde{\mathbf{H}} = [\mathbf{e}]_\times \mathbf{F}$ (Luong and Viéville 1996), which places the camera on the plane at infinity.

from which a projective reconstruction of the scene can be computed using triangulation (Section 11.2.4).

While the projective reconstruction may not be useful on its own, it can often be *upgraded* to an affine or metric reconstruction, as described below. Even without this step, however, the fundamental matrix \mathbf{F} can be very useful in finding additional correspondences, as they must all lie on corresponding epipolar lines, i.e., any feature \mathbf{x}_0 in image 0 must have its correspondence lying on the associated epipolar line $\mathbf{l}_1 = \mathbf{F}\mathbf{x}_0$ in image 1, assuming that the point motions are due to a rigid transformation.

11.3.4 Self-calibration

The results of structure from motion computation are much more useful if a *metric* reconstruction is obtained, i.e., one in which parallel lines are parallel, orthogonal walls are at right angles, and the reconstructed model is a scaled version of reality. Over the years, a large number of *self-calibration* (or *auto-calibration*) techniques have been developed for converting a projective reconstruction into a metric one, which is equivalent to recovering the unknown calibration matrices \mathbf{K}_j associated with each image (Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010).

In situations where additional information is known about the scene, different methods may be employed. For example, if there are parallel lines in the scene, three or more vanishing points, which are the images of points at infinity, can be used to establish the homography for the plane at infinity, from which focal lengths and rotations can be recovered. If two or more finite *orthogonal* vanishing points have been observed, the single-image calibration method based on vanishing points (Section 11.1.1) can be used instead.

In the absence of such external information, it is not possible to recover a fully parameterized independent calibration matrix \mathbf{K}_j for each image from correspondences alone. To see this, consider the set of all camera matrices $\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]$ projecting world coordinates $\mathbf{p}_i = (X_i, Y_i, Z_i, W_i)$ into screen coordinates $\mathbf{x}_{ij} \sim \mathbf{P}_j \mathbf{p}_i$. Now consider transforming the 3D scene $\{\mathbf{p}_i\}$ through an arbitrary 4×4 projective transformation $\tilde{\mathbf{H}}$, yielding a new model consisting of points $\mathbf{p}'_i = \tilde{\mathbf{H}} \mathbf{p}_i$. Post-multiplying each \mathbf{P}_j matrix by $\tilde{\mathbf{H}}^{-1}$ still produces the same screen coordinates and a new set calibration matrices can be computed by applying RQ decomposition to the new camera matrix $\mathbf{P}'_j = \mathbf{P}_j \tilde{\mathbf{H}}^{-1}$.

For this reason, all self-calibration methods assume some restricted form of the calibration matrix, either by setting or equating some of their elements or by assuming that they do not vary over time. While most of the techniques discussed by Hartley and Zisserman (2004); Moons, Van Gool, and Vergauwen (2010) require three or more frames, in this section we present a simple technique that can recover the focal lengths (f_0, f_1) of both images from the

fundamental matrix \mathbf{F} in a two-frame reconstruction (Hartley and Zisserman 2004, p. 472).

To accomplish this, we assume that the camera has zero skew, a known aspect ratio (usually set to 1), and a known image center, as in Equation (2.59). How reasonable is this assumption in practice? The answer, as with many questions, is “it depends”.

If absolute metric accuracy is required, as in photogrammetry applications, it is imperative to pre-calibrate the cameras using one of the techniques from Section 11.1 and to use ground control points to pin down the reconstruction. If instead, we simply wish to reconstruct the world for visualization or image-based rendering applications, as in the Photo Tourism system of Snavely, Seitz, and Szeliski (2006), this assumption is quite reasonable in practice.

Most cameras today have square pixels and an image center near the middle of the image, and are much more likely to deviate from a simple camera model due to radial distortion (Section 11.1.4), which should be compensated for whenever possible. The biggest problems occur when images have been cropped off-center, in which case the image center will no longer be in the middle, or when perspective pictures have been taken of a different picture, in which case a general camera matrix becomes necessary.¹⁷

Given these caveats, the two-frame focal length estimation algorithm based on the Kruppa equations developed by Hartley and Zisserman (2004, p. 456) proceeds as follows. Take the left and right singular vectors $\{\mathbf{u}_0, \mathbf{u}_1, \mathbf{v}_0, \mathbf{v}_1\}$ of the fundamental matrix \mathbf{F} (11.50) and their associated singular values $\{\sigma_0, \sigma_1\}$ and form the following set of equations:

$$\frac{\mathbf{u}_1^T \mathbf{D}_0 \mathbf{u}_1}{\sigma_0^2 \mathbf{v}_0^T \mathbf{D}_1 \mathbf{v}_0} = -\frac{\mathbf{u}_0^T \mathbf{D}_0 \mathbf{u}_1}{\sigma_0 \sigma_1 \mathbf{v}_0^T \mathbf{D}_1 \mathbf{v}_1} = \frac{\mathbf{u}_0^T \mathbf{D}_0 \mathbf{u}_0}{\sigma_1^2 \mathbf{v}_1^T \mathbf{D}_1 \mathbf{v}_1}, \quad (11.56)$$

where the two matrices

$$\mathbf{D}_j = \mathbf{K}_j \mathbf{K}_j^T = \text{diag}(f_j^2, f_j^2, 1) = \begin{bmatrix} f_j^2 & & \\ & f_j^2 & \\ & & 1 \end{bmatrix} \quad (11.57)$$

encode the unknown focal lengths. For simplicity, let us rewrite each of the numerators and denominators in (11.56) as

$$e_{ij0}(f_0^2) = \mathbf{u}_i^T \mathbf{D}_0 \mathbf{u}_j = a_{ij} + b_{ij} f_0^2, \quad (11.58)$$

$$e_{ij1}(f_1^2) = \sigma_i \sigma_j \mathbf{v}_i^T \mathbf{D}_1 \mathbf{v}_j = c_{ij} + d_{ij} f_1^2. \quad (11.59)$$

Notice that each of these is affine (linear plus constant) in either f_0^2 or f_1^2 . Hence, we can cross-multiply these equations to obtain quadratic equations in f_j^2 , which can readily be

¹⁷In Photo Tourism, our system registered photographs of an information sign outside Notre Dame with real pictures of the cathedral.

solved. (See also the work by Bougnoux (1998) and Kanatani and Matsunaga (2000) for some alternative formulations.)

An alternative solution technique is to observe that we have a set of three equations related by an unknown scalar λ , i.e.,

$$e_{ij0}(f_0^2) = \lambda e_{ij1}(f_1^2) \quad (11.60)$$

(Richard Hartley, personal communication, July 2009). These can readily be solved to yield $(f_0^2, \lambda f_1^2, \lambda)$ and hence (f_0, f_1) .

How well does this approach work in practice? There are certain degenerate configurations, such as when there is no rotation or when the optical axes intersect, when it does not work at all. (In such a situation, you can vary the focal lengths of the cameras and obtain a deeper or shallower reconstruction, which is an example of a *bas-relief ambiguity* (Section 11.4.5).) Hartley and Zisserman (2004) recommend using techniques based on three or more frames. However, if you find two images for which the estimates of $(f_0^2, \lambda f_1^2, \lambda)$ are well conditioned, they can be used to initialize a more complete bundle adjustment of all the parameters (Section 11.4.2). An alternative, which is often used in systems such as Photo Tourism, is to use camera EXIF tags or generic default values to initialize focal length estimates and refine them as part of bundle adjustment.

11.3.5 Application: View morphing

An interesting application of basic two-frame structure from motion is *view morphing* (also known as *view interpolation*, see Section 14.1), which can be used to generate a smooth 3D animation from one view of a 3D scene to another (Chen and Williams 1993; Seitz and Dyer 1996).

To create such a transition, you must first smoothly interpolate the camera matrices, i.e., the camera positions, orientations, and focal lengths. While simple linear interpolation can be used (representing rotations as quaternions (Section 2.1.3)), a more pleasing effect is obtained by *easing in* and *easing out* the camera parameters, e.g., using a raised cosine, as well as moving the camera along a more circular trajectory (Snavely, Seitz, and Szeliski 2006).

To generate in-between frames, either a full set of 3D correspondences needs to be established (Section 12.3) or 3D models (proxies) must be created for each reference view. Section 14.1 describes several widely used approaches to this problem. One of the simplest is to just triangulate the set of matched feature points in each image, e.g., using Delaunay triangulation. As the 3D points are re-projected into their intermediate views, pixels can be mapped from their original source images to their new views using affine or projective mapping (Szeliski and Shum 1997). The final image is then composited using a linear blend of

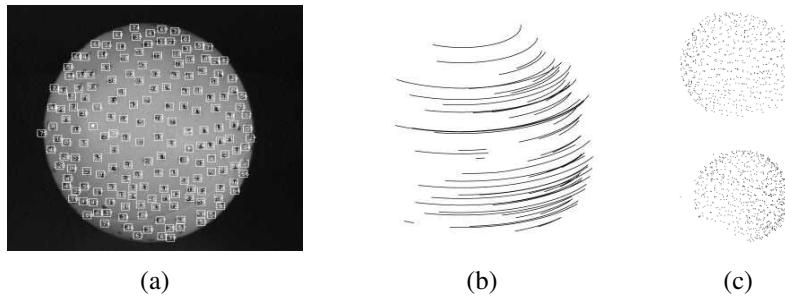


Figure 11.13 3D reconstruction of a rotating ping pong ball using factorization (Tomasi and Kanade 1992) © 1992 Springer: (a) sample image with tracked features overlaid; (b) subsampled feature motion stream; (c) two views of the reconstructed 3D model.

the two reference images, as with usual morphing (Section 3.6.3).

11.4 Multi-frame structure from motion

While two-frame techniques are useful for reconstructing sparse geometry from stereo image pairs and for initializing larger-scale 3D reconstructions, most applications can benefit from the much larger number of images that are usually available in photo collections and videos of scenes.

In this section, we briefly review an older technique called *factorization*, which can provide useful solutions for short video sequences, and then turn to the more commonly used *bundle adjustment* approach, which uses non-linear least squares to obtain optimal solutions under general camera configurations.

11.4.1 Factorization

When processing video sequences, we often get extended *feature tracks* (Section 7.1.5) from which it is possible to recover the structure and motion using a process called *factorization*. Consider the tracks generated by a rotating ping pong ball, which has been marked with dots to make its shape and motion more discernable (Figure 11.13). We can readily see from the shape of the tracks that the moving object must be a sphere, but how can we infer this mathematically?

It turns out that, under orthography or related models we discuss below, the shape and motion can be recovered simultaneously using a singular value decomposition (Tomasi and

Kanade 1992). The details of how to do this are presented in the paper by Tomasi and Kanade (1992) and also in the first edition of this book (Szeliski 2010, Section 7.3).

Once the rotation matrices and 3D point locations have been recovered, there still exists a bas-relief ambiguity, i.e., we can never be sure if the object is rotating left to right or if its depth reversed version is moving the other way. (This can be seen in the classic rotating Necker Cube visual illusion.) Additional cues, such as the appearance and disappearance of points, or perspective effects, both of which are discussed below, can be used to remove this ambiguity.

For motion models other than pure orthography, e.g., for scaled orthography or paraperspective, the approach above must be extended in the appropriate manner. Such techniques are relatively straightforward to derive from first principles; more details can be found in papers that extend the basic factorization approach to these more flexible models (Poelman and Kanade 1997). Additional extensions of the original factorization algorithm include multi-body rigid motion (Costeira and Kanade 1995), sequential updates to the factorization (Morita and Kanade 1997), the addition of lines and planes (Morris and Kanade 1998), and re-scaling the measurements to incorporate individual location uncertainties (Anandan and Irani 2002).

A disadvantage of factorization approaches is that they require a complete set of tracks, i.e., each point must be visible in each frame, for the factorization approach to work. Tomasi and Kanade (1992) deal with this problem by first applying factorization to smaller denser subsets and then using known camera (motion) or point (structure) estimates to *hallucinate* additional missing values, which allows them to incrementally incorporate more features and cameras. Huynh, Hartley, and Heyden (2003) extend this approach to view missing data as special cases of outliers. Buchanan and Fitzgibbon (2005) develop fast iterative algorithms for performing large matrix factorizations with missing data. The general topic of principal component analysis (PCA) with missing data also appears in other computer vision problems (Shum, Ikeuchi, and Reddy 1995; De la Torre and Black 2003; Gross, Matthews, and Baker 2006; Torresani, Hertzmann, and Bregler 2008; Vidal, Ma, and Sastry 2016).

Perspective and projective factorization

Another disadvantage of regular factorization is that it cannot deal with perspective cameras. One way to get around this problem is to perform an initial affine (e.g., orthographic) reconstruction and to then correct for the perspective effects in an iterative manner (Christy and Horaud 1996). This algorithm usually converges in three to five iterations, with the majority of the time spent in the SVD computation.

An alternative approach, which does not assume partially calibrated cameras (known im-

age center, square pixels, and zero skew) is to perform a fully *projective* factorization (Sturm and Triggs 1996; Triggs 1996). In this case, the inclusion of the third row of the camera matrix in the measurement matrix is equivalent to multiplying each reconstructed measurement $\mathbf{x}_{ji} = \mathbf{M}_j \mathbf{p}_i$ by its inverse (projective) depth $\eta_{ji} = d_{ji}^{-1} = 1/(\mathbf{P}_{j2} \mathbf{p}_i)$ or, equivalently, multiplying each measured position by its projective depth d_{ji} . In the original paper by Sturm and Triggs (1996), the projective depths d_{ji} are obtained from two-frame reconstructions, while in later work (Triggs 1996; Oliensis and Hartley 2007), they are initialized to $d_{ji} = 1$ and updated after each iteration. Oliensis and Hartley (2007) present an update formula that is guaranteed to converge to a fixed point. None of these authors suggest actually estimating the third row of \mathbf{P}_j as part of the projective depth computations. In any case, it is unclear when a fully projective reconstruction would be preferable to a partially calibrated one, especially if they are being used to initialize a full bundle adjustment of all the parameters.

One of the attractions of factorization methods is that they provide a “closed form” (sometimes called a “linear”) method to initialize iterative techniques such as bundle adjustment. An alternative initialization technique is to estimate the homographies corresponding to some common plane seen by all the cameras (Rother and Carlsson 2002). In a calibrated camera setting, this can correspond to estimating consistent rotations for all of the cameras, for example, using matched vanishing points (Antone and Teller 2002). Once these have been recovered, the camera positions can then be obtained by solving a linear system (Antone and Teller 2002; Rother and Carlsson 2002; Rother 2003).

11.4.2 Bundle adjustment

As we have mentioned several times before, the most accurate way to recover structure and motion is to perform robust non-linear minimization of the measurement (re-projection) errors, which is commonly known in the photogrammetry (and now computer vision) communities as *bundle adjustment*.¹⁸ Triggs, McLauchlan *et al.* (1999) provide an excellent overview of this topic, including its historical development, pointers to the photogrammetry literature (Slama 1980; Atkinson 1996; Kraus 1997), and subtle issues with gauge ambiguities. The topic is also treated in depth in textbooks and surveys on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010).

We have already introduced the elements of bundle adjustment in our discussion on iterative pose estimation (Section 11.2.2), i.e., Equations (11.14–11.20) and Figure 11.7. The

¹⁸The term “bundle” refers to the bundles of rays connecting camera centers to 3D points and the term “adjustment” refers to the iterative minimization of re-projection error. Alternative terms for this in the vision community include *optimal motion estimation* (Weng, Ahuja, and Huang 1993) and *non-linear least squares* (Appendix A.3) (Taylor, Kriegman, and Anandan 1991; Szeliski and Kang 1994).

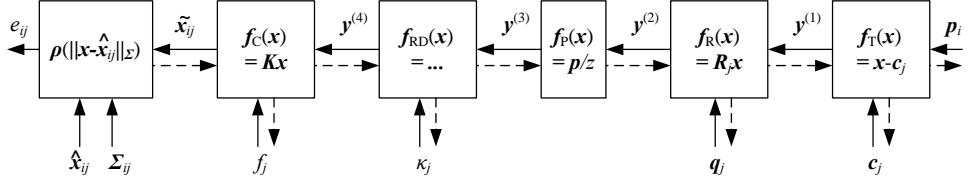


Figure 11.14 A set of chained transforms for projecting a 3D point \mathbf{p}_i into a 2D measurement \mathbf{x}_{ij} through a series of transformations $\mathbf{f}^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass. The formula for the radial distortion function is $\mathbf{f}_{RD}(\mathbf{x}) = (1 + \kappa_1 r^2 + \kappa_2 r^4)\mathbf{x}$.

biggest difference between these formulas and full bundle adjustment is that our feature location measurements \mathbf{x}_{ij} now depend not only on the point (track) index i but also on the camera pose index j ,

$$\mathbf{x}_{ij} = \mathbf{f}(\mathbf{p}_i, \mathbf{R}_j, \mathbf{c}_j, \mathbf{K}_j), \quad (11.61)$$

and that the 3D point positions \mathbf{p}_i are also being simultaneously updated. In addition, it is common to add a stage for radial distortion parameter estimation (2.78),

$$\mathbf{f}_{RD}(\mathbf{x}) = (1 + \kappa_1 r^2 + \kappa_2 r^4)\mathbf{x}, \quad (11.62)$$

if the cameras being used have not been pre-calibrated, as shown in Figure 11.14.

While most of the boxes (transforms) in Figure 11.14 have previously been explained (11.19), the leftmost box has not. This box performs a robust comparison of the predicted and measured 2D locations $\hat{\mathbf{x}}_{ij}$ and $\tilde{\mathbf{x}}_{ij}$ after re-scaling by the measurement noise covariance Σ_{ij} . In more detail, this operation can be written as

$$\mathbf{r}_{ij} = \tilde{\mathbf{x}}_{ij} - \hat{\mathbf{x}}_{ij}, \quad (11.63)$$

$$s_{ij}^2 = \mathbf{r}_{ij}^T \Sigma_{ij}^{-1} \mathbf{r}_{ij}, \quad (11.64)$$

$$e_{ij} = \hat{\rho}(s_{ij}^2), \quad (11.65)$$

where $\hat{\rho}(r^2) = \rho(r)$. The corresponding Jacobians (partial derivatives) can be written as

$$\frac{\partial e_{ij}}{\partial s_{ij}^2} = \hat{\rho}'(s_{ij}^2), \quad (11.66)$$

$$\frac{\partial s_{ij}^2}{\partial \tilde{\mathbf{x}}_{ij}} = \Sigma_{ij}^{-1} \mathbf{r}_{ij}. \quad (11.67)$$

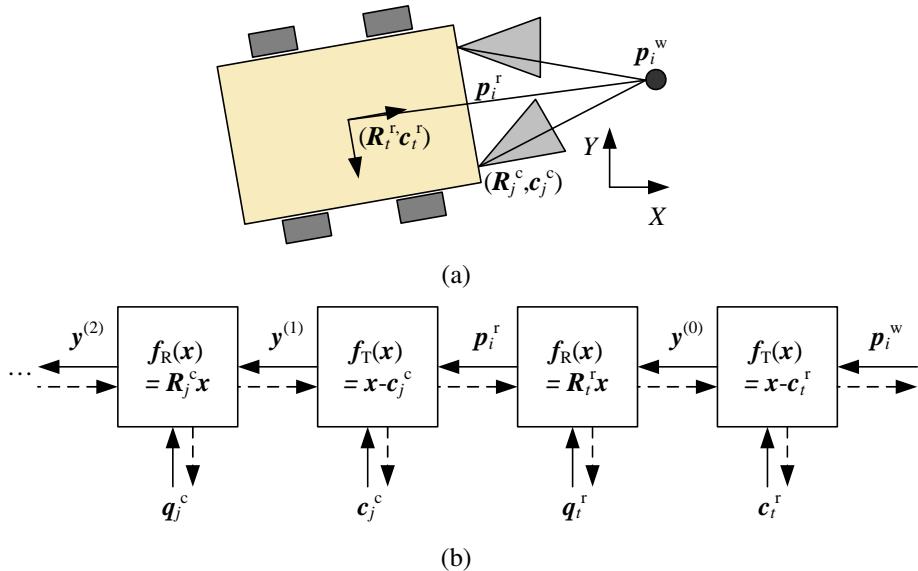


Figure 11.15 A camera rig and its associated transform chain. (a) As the mobile rig (robot) moves around in the world, its pose with respect to the world at time t is captured by $(\mathbf{R}_t^r, \mathbf{c}_t^r)$. Each camera’s pose with respect to the rig is captured by $(\mathbf{R}_j^c, \mathbf{c}_j^c)$. (b) A 3D point with world coordinates \mathbf{p}_i^w is first transformed into rig coordinates \mathbf{p}_i^r , and then through the rest of the camera-specific chain, as shown in Figure 11.14.

The advantage of the chained representation introduced above is that it not only makes the computations of the partial derivatives and Jacobians simpler but it can also be adapted to any camera configuration. Consider for example a pair of cameras mounted on a robot that is moving around in the world, as shown in Figure 11.15a. By replacing the rightmost two transformations in Figure 11.14 with the transformations shown in Figure 11.15b, we can simultaneously recover the position of the robot at each time and the calibration of each camera with respect to the rig, in addition to the 3D structure of the world.

11.4.3 Exploiting sparsity

Large bundle adjustment problems, such as those involving reconstructing 3D scenes from thousands of internet photographs (Snavely, Seitz, and Szeliski 2008b; Agarwal, Furukawa *et al.* 2010, 2011; Snavely, Simon *et al.* 2010), can require solving non-linear least squares problems with millions of measurements (feature matches) and tens of thousands of unknown parameters (3D point positions and camera poses). Unless some care is taken, these kinds of

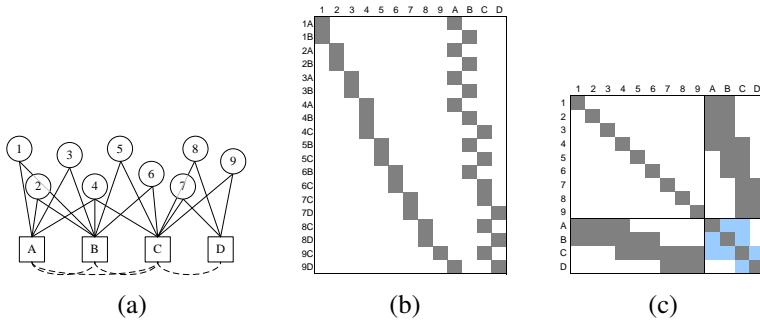


Figure 11.16 (a) Bipartite graph for a toy structure from motion problem and (b) its associated Jacobian \mathbf{J} and (c) Hessian \mathbf{A} . Numbers indicate 3D points and letters indicate cameras. The dashed arcs and light blue squares indicate the fill-in that occurs when the structure (point) variables are eliminated.

problem can become intractable, because the (direct) solution of dense least squares problems is cubic in the number of unknowns.

Fortunately, structure from motion is a *bipartite* problem in structure and motion. Each feature point \mathbf{x}_{ij} in a given image depends on one 3D point position \mathbf{p}_i and one 3D camera pose ($\mathbf{R}_j, \mathbf{c}_j$). This is illustrated in Figure 11.16a, where each circle (1–9) indicates a 3D point, each square (A–D) indicates a camera, and lines (edges) indicate which points are visible in which cameras (2D features). If the values for all the points are known or fixed, the equations for all the cameras become independent, and vice versa.

If we order the structure variables before the motion variables in the Hessian matrix \mathbf{A} (and hence also the right-hand side vector \mathbf{b}), we obtain a structure for the Hessian shown in Figure 11.16c.¹⁹ When such a system is solved using sparse Cholesky factorization (see Appendix A.4) (Björck 1996; Golub and Van Loan 1996), the *fill-in* occurs in the smaller motion Hessian A_{cc} (Szeliski and Kang 1994; Triggs, McLauchlan *et al.* 1999; Hartley and Zisserman 2004; Lourakis and Argyros 2009; Engels, Stewénius, and Nistér 2006). More recent papers (Byr öd and Åström 2009; Jeong, Nistér *et al.* 2010; Agarwal, Snavely *et al.* 2010; Jeong, Nistér *et al.* 2012) explore the use of iterative (conjugate gradient) techniques for the solution of bundle adjustment problems. Other papers explore the use of parallel multicore algorithms (Wu, Agarwal *et al.* 2011).

¹⁹This ordering is preferable when there are fewer cameras than 3D points, which is the usual case. The exception is when we are tracking a small number of points through many video frames, in which case this ordering should be reversed.

In more detail, the *reduced* motion Hessian is computed using the *Schur complement*,

$$\mathbf{A}'_{CC} = \mathbf{A}_{CC} - \mathbf{A}_{PC}^T \mathbf{A}_{PP}^{-1} \mathbf{A}_{PC}, \quad (11.68)$$

where \mathbf{A}_{PP} is the point (structure) Hessian (the top left block of Figure 11.16c), \mathbf{A}_{PC} is the point-camera Hessian (the top right block), and \mathbf{A}_{CC} and \mathbf{A}'_{CC} are the motion Hessians before and after the point variable elimination (the bottom right block of Figure 11.16c). Notice that \mathbf{A}'_{CC} has a non-zero entry between two cameras if they see any 3D point in common. This is indicated with dashed arcs in Figure 11.16a and light blue squares in Figure 11.16c.

Whenever there are global parameters present in the reconstruction algorithm, such as camera intrinsics that are common to all of the cameras, or camera rig calibration parameters such as those shown in Figure 11.15, they should be ordered last (placed along the right and bottom edges of \mathbf{A}) to reduce fill-in.

Engels, Stewénius, and Nistér (2006) provide a nice recipe for sparse bundle adjustment, including all the steps needed to initialize the iterations, as well as typical computation times for a system that uses a fixed number of backward-looking frames in a real-time setting. They also recommend using homogeneous coordinates for the structure parameters \mathbf{p}_i , which is a good idea, as it avoids numerical instabilities for points near infinity.

Bundle adjustment is now the standard method of choice for most structure-from-motion problems and is commonly applied to problems with hundreds of weakly calibrated images and tens of thousands of points. (Much larger problems are commonly solved in photogrammetry and aerial imagery, but these are usually carefully calibrated and make use of surveyed ground control points.) However, as the problems become larger, it becomes impractical to re-solve full bundle adjustment problems at each iteration.

One approach to dealing with this problem is to use an incremental algorithm, where new cameras are added over time. (This makes particular sense if the data is being acquired from a video camera or moving vehicle (Nistér, Naroditsky, and Bergen 2006; Pollefeys, Nistér *et al.* 2008).) A Kalman filter can be used to incrementally update estimates as new information is acquired. Unfortunately, such sequential updating is only statistically optimal for linear least squares problems.

For non-linear problems such as structure from motion, an extended Kalman filter, which linearizes measurement and update equations around the current estimate, needs to be used (Gelb 1974; Viéville and Faugeras 1990). To overcome this limitation, several passes can be made through the data (Azarbayejani and Pentland 1995). Because points disappear from view (and old cameras become irrelevant), a *variable state dimension filter* (VSDF) can be used to adjust the set of state variables over time, for example, by keeping only cameras and point tracks seen in the last k frames (McLauchlan 2000). A more flexible approach to using

a fixed number of frames is to propagate corrections backwards through points and cameras until the changes on parameters are below a threshold (Steedly and Essa 2001). Variants of these techniques, including methods that use a fixed window for bundle adjustment (Engels, Stewénius, and Nistér 2006) or select keyframes for doing full bundle adjustment (Klein and Murray 2008) are now commonly used in simultaneous localization and mapping (SLAM) and augmented-reality applications, as discussed in Section 11.5.

When maximum accuracy is required, it is still preferable to perform a full bundle adjustment over all the frames. To control the resulting computational complexity, one approach is to lock together subsets of frames into locally rigid configurations and to optimize the relative positions of these cluster (Steedly, Essa, and Dellaert 2003). A different approach is to select a smaller number of frames to form a *skeletal set* that still spans the whole dataset and produces reconstructions of comparable accuracy (Snavely, Seitz, and Szeliski 2008b). We describe this latter technique in more detail in Section 11.4.6, where we discuss applications of structure from motion to large image sets. Additional techniques for efficiently solving large structure from motion and SLAM systems can be found in the survey by Dellaert and Kaess (2017); Dellaert (2021).

While bundle adjustment and other robust non-linear least squares techniques are the methods of choice for most structure-from-motion problems, they suffer from initialization problems, i.e., they can get stuck in local energy minima if not started sufficiently close to the global optimum. Many systems try to mitigate this by being conservative in what reconstruction they perform early on and which cameras and points they add to the solution (Section 11.4.6). An alternative, however, is to re-formulate the problem using a norm that supports the computation of global optima.

Kahl and Hartley (2008) describe techniques for using L_∞ norms in geometric reconstruction problems. The advantage of such norms is that globally optimal solutions can be efficiently computed using second-order cone programming (SOCP). The disadvantage is that L_∞ norms are particularly sensitive to outliers and so must be combined with good outlier rejection techniques before they can be used.

A large number of high-quality open source bundle adjustment packages have been developed, including the Ceres Solver,²⁰ Multicore Bundle Adjustment (Wu, Agarwal *et al.* 2011),²¹ the Sparse Levenberg-Marquardt based non-linear least squares optimizer and bundle adjuster,²² and OpenSfM.²³ You can find more pointers to open-source software in Appendix Appendix C.2 and reviews of open-source and commercial photogrammetry soft-

²⁰<http://ceres-solver.org>

²¹<https://grail.cs.washington.edu/projects/mcba>

²²<https://github.com/chzach/SSBA>

²³<https://www.opensfm.org>

ware²⁴ as well as examples of their application²⁵ on the web.

11.4.4 Application: Match move

One of the neatest applications of structure from motion is to estimate the 3D motion of a video or film camera, along with the geometry of a 3D scene, in order to superimpose 3D graphics or computer-generated images (CGI) on the scene. In the visual effects industry, this is known as the *match move* problem (Roble 1999), as the motion of the synthetic 3D camera used to render the graphics must be *matched* to that of the real-world camera. For very small motions, or motions involving pure camera rotations, one or two tracked points can suffice to compute the necessary visual motion. For planar surfaces moving in 3D, four points are needed to compute the homography, which can then be used to insert planar overlays, e.g., to replace the contents of advertising billboards during sporting events.

The general version of this problem requires the estimation of the full 3D camera pose along with the focal length (zoom) of the lens and potentially its radial distortion parameters (Roble 1999). When the 3D structure of the scene is known ahead of time, pose estimation techniques such as *view correlation* (Bogart 1991) or *through-the-lens camera control* (Gleicher and Witkin 1992) can be used, as described in Section 11.4.4.

For more complex scenes, it is usually preferable to recover the 3D structure simultaneously with the camera motion using structure-from-motion techniques. The trick with using such techniques is that to prevent any visible jitter between the synthetic graphics and the actual scene, features must be tracked to very high accuracy and ample feature tracks must be available in the vicinity of the insertion location. Some of today's best known match move software packages, such as the *boujou* package from 2d3, which won an Emmy award in 2002, originated in structure-from-motion research in the computer vision community (Fitzgibbon and Zisserman 1998).

11.4.5 Uncertainty and ambiguities

Because structure from motion involves the estimation of so many highly coupled parameters, often with no known “ground truth” components, the estimates produced by structure from motion algorithms can often exhibit large amounts of uncertainty (Szeliski and Kang 1997; Wilson and Wehrwein 2020). An example of this is the classic *bas-relief ambiguity*, which

²⁴<https://peterfalkingham.com/2020/07/10/free-and-commercial-photogrammetry-software-review-2020>

²⁵<https://beforesandafters.com/2020/07/06/tales-from-on-set-lidar-scanning-for-joker-and-john-wick-3/>, <https://rd.nytimes.com/projects/reconstructing-journalistic-scenes-in-3d>

makes it hard to simultaneously estimate the 3D depth of a scene and the amount of camera motion (Oliensis 2005).²⁶

As mentioned before, a unique coordinate frame and scale for a reconstructed scene cannot be recovered from monocular visual measurements alone. (When a stereo rig is used, the scale can be recovered if we know the distance (baseline) between the cameras.) This seven-degree-of-freedom (coordinate frame and scale) *gauge ambiguity* makes it tricky to compute the covariance matrix associated with a 3D reconstruction (Triggs, McLauchlan *et al.* 1999; Kanatani and Morris 2001). A simple way to compute a covariance matrix that ignores the gauge freedom (indeterminacy) is to throw away the seven smallest eigenvalues of the information matrix (inverse covariance), whose values are equivalent to the problem Hessian \mathbf{A} up to noise scaling (see Section 8.1.4 and Appendix B.6). After we do this, the resulting matrix can be inverted to obtain an estimate of the parameter covariance.

Szeliski and Kang (1997) use this approach to visualize the largest directions of variation in typical structure from motion problems. Not surprisingly, they find that, ignoring the gauge freedoms, the greatest uncertainties for problems such as observing an object from a small number of nearby viewpoints are in the depths of the 3D structure relative to the extent of the camera motion.²⁷

It is also possible to estimate *local* or *marginal* uncertainties for individual parameters, which corresponds simply to taking block sub-matrices from the full covariance matrix. Under certain conditions, such as when the camera poses are relatively certain compared to 3D point locations, such uncertainty estimates can be meaningful. However, in many cases, individual uncertainty measures can mask the extent to which reconstruction errors are correlated, which is why looking at the first few modes of greatest joint variation can be helpful.

The other way in which gauge ambiguities affect structure from motion and, in particular, bundle adjustment is that they make the system Hessian matrix \mathbf{A} rank-deficient and hence impossible to invert. A number of techniques have been proposed to mitigate this problem (Triggs, McLauchlan *et al.* 1999; Bartoli 2003). In practice, however, it appears that simply adding a small amount of the Hessian diagonal $\lambda \text{diag}(\mathbf{A})$ to the Hessian \mathbf{A} itself, as is done in the Levenberg–Marquardt non-linear least squares algorithm (Appendix A.3), usually works well.

²⁶Bas-relief refers to a kind of sculpture in which objects, often on ornamental friezes, are sculpted with less depth than they actually occupy. When lit from above by sunlight, they appear to have true 3D depth because of the ambiguity between relative depth and the angle of the illuminant (Section 13.1.1).

²⁷A good way to minimize the amount of such ambiguities is to use wide field of view cameras (Antone and Teller 2002; Levin and Szeliski 2006).



Figure 11.17 *Incremental structure from motion (Snavely, Seitz, and Szeliski 2006) © 2006 ACM. Starting with an initial two-frame reconstruction of Trevi Fountain, batches of images are added using pose estimation, and their positions (along with the 3D model) are refined using bundle adjustment.*

11.4.6 Application: Reconstruction from internet photos

The most widely used application of structure from motion is in the reconstruction of 3D objects and scenes from video sequences and collections of images (Pollefeys and Van Gool 2002). The last two decades have seen an explosion of techniques for performing this task automatically without the need for any manual correspondence or pre-surveyed ground control points. A lot of these techniques assume that the scene is taken with the same camera and hence the images all have the same intrinsics (Fitzgibbon and Zisserman 1998; Koch, Pollefeys, and Van Gool 2000; Schaffalitzky and Zisserman 2002; Tuytelaars and Van Gool 2004; Pollefeys, Nistér *et al.* 2008; Moons, Van Gool, and Vergauwen 2010). Many of these techniques take the results of the sparse feature matching and structure from motion computation and then compute dense 3D surface models using multi-view stereo techniques (Section 12.7) (Koch, Pollefeys, and Van Gool 2000; Pollefeys and Van Gool 2002; Pollefeys, Nistér *et al.* 2008; Moons, Van Gool, and Vergauwen 2010; Schönberger, Zheng *et al.* 2016).

An exciting innovation in this space has been the application of structure from motion and multi-view stereo techniques to thousands of images taken from the internet, where very little is known about the cameras taking the photographs (Snavely, Seitz, and Szeliski 2008a). Before the structure from motion computation can begin, it is first necessary to establish sparse correspondences between different pairs of images and to then link such correspondences into *feature tracks*, which associate individual 2D image features with global 3D points. Because the $O(N^2)$ comparison of all pairs of images can be very slow, a number of techniques have been developed in the recognition community to make this process faster (Section 7.1.4) (Nistér and Stewénius 2006; Philbin, Chum *et al.* 2008; Li, Wu *et al.* 2008; Chum, Philbin, and Zisserman 2008; Chum and Matas 2010a; Arandjelović and Zisserman 2012).

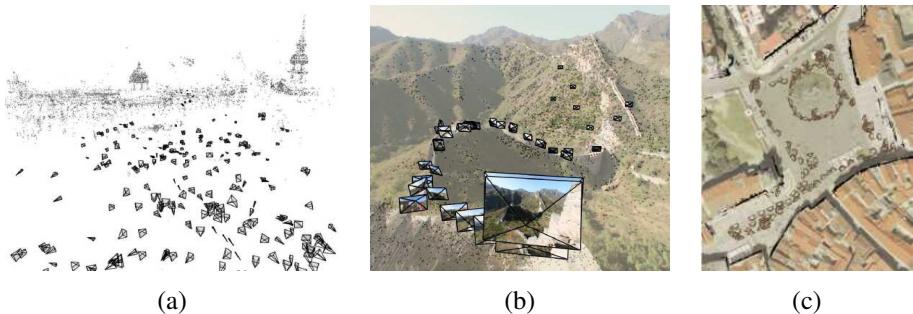


Figure 11.18 3D reconstructions produced by the incremental structure from motion algorithm developed by Snavely, Seitz, and Szeliski (2006) © 2006 ACM: (a) cameras and point cloud from Trafalgar Square; (b) cameras and points overlaid on an image from the Great Wall of China; (c) overhead view of a reconstruction of the Old Town Square in Prague registered to an aerial photograph.

To begin the reconstruction process, it is important to select a good pair of images, where there are both a large number of consistent matches (to lower the likelihood of incorrect correspondences) and a significant amount of out-of-plane parallax,²⁸ to ensure that a stable reconstruction can be obtained (Snavely, Seitz, and Szeliski 2006). The EXIF tags associated with the photographs can be used to get good initial estimates for camera focal lengths, although this is not always strictly necessary, because these parameters are re-adjusted as part of the bundle adjustment process.

Once an initial pair has been reconstructed, the pose of cameras that see a sufficient number of the resulting 3D points can be estimated (Section 11.2) and the complete set of cameras and feature correspondences can be used to perform another round of bundle adjustment. Figure 11.17 shows the progression of the incremental bundle adjustment algorithm, where sets of cameras are added after each successive round of bundle adjustment, while Figure 11.18 shows some additional results. An alternative to this kind of *seed and grow* approach is to first reconstruct triplets of images and then hierarchically merge them into larger collections (Fitzgibbon and Zisserman 1998).

Unfortunately, as the incremental structure from motion algorithm continues to add more cameras and points, it can become extremely slow. The direct solution of a dense system of $O(N)$ equations for the camera pose updates can take $O(N^3)$ time; while structure from motion problems are rarely dense, scenes such as city squares have a high percentage of

²⁸A simple way to compute this is to robustly fit a homography to the correspondences and measure reprojection errors.

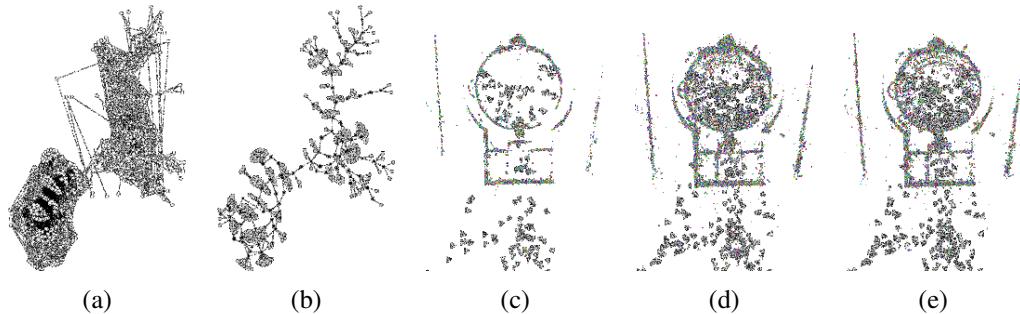


Figure 11.19 Large-scale structure from motion using skeletal sets (Snavely, Seitz, and Szeliski 2008b) © 2008 IEEE: (a) original match graph for 784 images; (b) skeletal set containing 101 images; (c) top-down view of scene (Pantheon) reconstructed from the skeletal set; (d) reconstruction after adding in the remaining images using pose estimation; (e) final bundle adjusted reconstruction, which is almost identical.

cameras that see points in common. Re-running the bundle adjustment algorithm after every few camera additions results in a quartic scaling of the run time with the number of images in the dataset. One approach to solving this problem is to select a smaller number of images for the original scene reconstruction and to fold in the remaining images at the very end.

Snavely, Seitz, and Szeliski (2008b) develop an algorithm for computing such a *skeletal set* of images, which is guaranteed to produce a reconstruction whose error is within a bounded factor of the optimal reconstruction accuracy. Their algorithm first evaluates all pairwise uncertainties (position covariances) between overlapping images and then chains them together to estimate a lower bound for the relative uncertainty of any distant pair. The skeletal set is constructed so that the maximal uncertainty between any pair grows by no more than a constant factor. Figure 11.19 shows an example of the skeletal set computed for 784 images of the Pantheon in Rome. As you can see, even though the skeletal set contains just a fraction of the original images, the shapes of the skeletal set and full bundle adjusted reconstructions are virtually indistinguishable.

Since the initial publication on large-scale internet photo reconstruction by Snavely, Seitz, and Szeliski (2008a,b), there have been a large number of follow-on papers exploring even larger datasets and more efficient algorithms (Agarwal, Furukawa *et al.* 2010, 2011; Frahm, Fite-Georgel *et al.* 2010; Wu 2013; Heinly, Schönberger *et al.* 2015; Schönberger and Frahm 2016). Among these, the COLMAP open source structure from motion and multi-view stereo system is currently one of the most widely used, as it can reconstruct extremely large scenes,

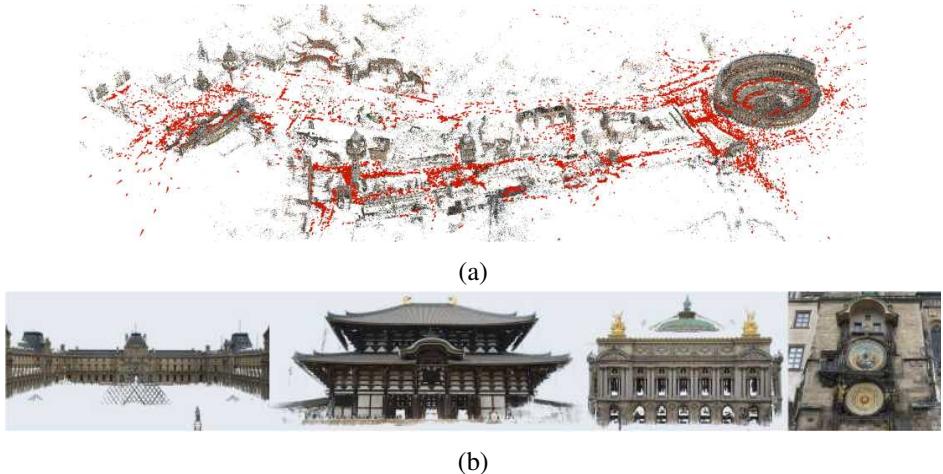


Figure 11.20 Large-scale reconstructions created with the COLMAP structure from motion and multi-view stereo system: (a) sparse model of central Rome constructed from 21K photos (Schönberger and Frahm 2016) © 2016 IEEE; (b) dense models of several landmarks produced with the MVS pipeline (Schönberger, Zheng et al. 2016) © 2016 Springer.

such as the one shown in Figure 11.20 (Schönberger and Frahm 2016).²⁹

The ability to automatically reconstruct 3D models from large, unstructured image collections has also brought to light subtle problems with traditional structure from motion algorithms, including the need to deal with repetitive and duplicate structures (Wu, Frahm, and Pollefeys 2010; Roberts, Sinha et al. 2011; Wilson and Snavely 2013; Heinly, Dunn, and Frahm 2014) as well as dynamic visual objects such as people (Ji, Dunn, and Frahm 2014; Zheng, Wang et al. 2014). It has also opened up a wide variety of additional applications, including the ability to automatically find and label locations and regions of interest (Simon, Snavely, and Seitz 2007; Simon and Seitz 2008; Gammeter, Bossard et al. 2009) and to cluster large image collections so that they can be automatically labeled (Li, Wu et al. 2008; Quack, Leibe, and Van Gool 2008). Some additional applications related to image-based rendering are discussed in more detail in Section 14.1.2.

11.4.7 Global structure from motion

While incremental bundle adjustment algorithms are still the most commonly used approaches for large-scale reconstruction (Schönberger and Frahm 2016), they can be quite slow because

²⁹<https://colmap.github.io>

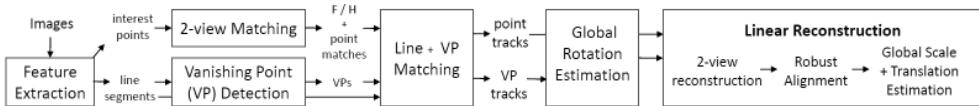


Figure 11.21 Global structure from motion pipeline from Sinha, Steedly, and Szeliski (2010) © 2010 Springer. Vanishing point and feature-based pairwise rotation estimates are used to first estimate a globally consistent set of orientations (rotations). The scales of all pairwise reconstructions along with the camera center positions are then estimated in a single linear least squares minimization.

of the need to successively solve increasing larger optimization problems. An alternative to iteratively growing the solution is to solve for all of the structure and motion unknowns in a single global step, once the feature correspondences have been established.

One approach to this is to set up a linear system of equations that relate all of the camera centers and 3D point, line, and plane equations to the known 2D feature or line positions (Kaucic, Hartley, and Dano 2001; Rother 2003). However, these approaches require a reference plane (e.g., building wall) to be visible and matched in all images, and are also sensitive to distant points, which must first be discarded. These approaches, while theoretically interesting, are not widely used.

A second approach, first proposed by Govindu (2001), starts by computing pairwise Euclidean structure and motion reconstructions using the techniques discussed in Section 11.3.³⁰ Pairwise rotation estimates are then used to compute a globally consistent orientation estimate for each camera, using a process known as *rotation averaging* (Govindu 2001; Martinec and Pajdla 2007; Chatterjee and Govindu 2013; Hartley, Trumpf *et al.* 2013; Dellaert, Rosen *et al.* 2020).³¹ In a final step, the camera positions are determined by scaling each of the local camera translations, after they have been rotated into a global coordinate system (Govindu 2001, 2004; Martinec and Pajdla 2007; Sinha, Steedly, and Szeliski 2010). In the robotics (SLAM) community, this last step is called *pose graph optimization* (Carlone, Tron *et al.* 2015).

Figure 11.21 shows a more recent pipeline implementing this concept, which includes the initial feature point extraction, matching, and two-view reconstruction, followed by global rotation estimation, and then a final solve for the camera centers. The pipeline developed by Sinha, Steedly, and Szeliski (2010) also matches vanishing points, when these can be found, in order to eliminate rotational drift in the global orientation estimates.

³⁰While almost of all of these techniques assume known calibration (focal lengths) for each image, Sweeney, Kneip *et al.* (2015) estimate focal lengths from refined fundamental matrices.

³¹We have already introduced the concept of rotation averaging when we discussed global registration of panoramas in Section 8.3.1.

While there are several alternative algorithms for estimating the global rotations, an even wider variety of algorithms exists for estimating the camera centers. After rotating all of the cameras by their global rotation estimate, we can compute globally oriented local translation direction in each reconstructed pair ij and denote this as $\hat{\mathbf{t}}_{ij}$. The fundamental relationship between the unknown camera centers $\{\mathbf{c}_i\}$ and the translation directions can be written as

$$\mathbf{c}_j - \mathbf{c}_i = s_{ij} \hat{\mathbf{t}}_{ij} \quad (11.69)$$

or

$$\hat{\mathbf{t}}_{ij} \times (\mathbf{c}_j - \mathbf{c}_i) = 0 \quad (11.70)$$

(Govindu 2001). The first set of equations can be solved to obtain the camera centers $\{\mathbf{c}_i\}$ and the scale variables s_{ij} , while the second directly produces only the camera positions. In addition to being homogeneous (only known up to a scale), the camera centers also have a translational *gauge freedom*, i.e., they can all be translated (but this is always the case with structure from motion).

Because these equations minimize the algebraic alignment between local translation directions and global camera center differences, they do not correctly weight reconstructions with different baselines. Several alternatives have been proposed to remediate this (Govindu 2004; Sinha, Steedly, and Szeliski 2010; Jiang, Cui, and Tan 2013; Moulon, Monasse, and Marlet 2013; Wilson and Snavely 2014; Cui and Tan 2015; Özyeşil and Singer 2015; Holynski, Geraghty *et al.* 2020). Some of these techniques also cannot handle collinear cameras, as in the original formulation, as well as some more recent ones, we can shift cameras along a collinear segment and still satisfy the directional constraints.

For community photo collections taken over a large area such as a plaza, this is not a crucial problem (Wilson and Snavely 2014). However, for reconstructions from video or walks around or through a building, the collinear camera problem is a real issue. Sinha, Steedly, and Szeliski (2010) handle this by estimating the relative scales of pairwise reconstructions that share a common camera and then use these relative scales to constraint all of the global scales.

Two open-source structure from motion pipelines that include some of these global techniques are Theia³² (Sweeney, Hollerer, and Turk 2015) and OpenMVG³³ (Moulon, Monasse *et al.* 2016). The papers have nice reviews of the related algorithms.

³²<http://www.theia-sfm.org>

³³<https://github.com/openMVG/openMVG>

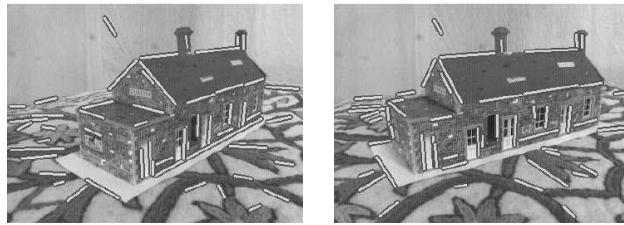


Figure 11.22 *Two images of a toy house along with their matched 3D line segments (Schmid and Zisserman 1997) © 1997 Springer.*

11.4.8 Constrained structure and motion

The most general algorithms for structure from motion make no prior assumptions about the objects or scenes that they are reconstructing. In many cases, however, the scene contains higher-level geometric primitives, such as lines and planes. These can provide information complementary to interest points and also serve as useful building blocks for 3D modeling and visualization. Furthermore, these primitives are often arranged in particular relationships, i.e., many lines and planes are either parallel or orthogonal to each other (Zhou, Furukawa, and Ma 2019; Zhou, Furukawa *et al.* 2020). This is particularly true of architectural scenes and models, which we study in more detail in Section 13.6.1.

Sometimes, instead of exploiting regularity in the scene structure, it is possible to take advantage of a constrained motion model. For example, if the object of interest is rotating on a turntable (Szeliski 1991b), i.e., around a fixed but unknown axis, specialized techniques can be used to recover this motion (Fitzgibbon, Cross, and Zisserman 1998). In other situations, the camera itself may be moving in a fixed arc around some center of rotation (Shum and He 1999). Specialized capture setups, such as mobile stereo camera rigs or moving vehicles equipped with multiple fixed cameras, can also take advantage of the knowledge that individual cameras are (mostly) fixed with respect to the capture rig, as shown in Figure 11.15.³⁴

Line-based techniques

It is well known that pairwise epipolar geometry cannot be recovered from line matches alone, even if the cameras are calibrated. To see this, think of projecting the set of lines in each image into a set of 3D planes in space. You can move the two cameras around into any configuration you like and still obtain a valid reconstruction for 3D lines.

³⁴Because of mechanical compliance and jitter, it may be prudent to allow for a small amount of individual camera rotation around a nominal position.

When lines are visible in three or more views, the trifocal tensor can be used to transfer lines from one pair of images to another (Hartley and Zisserman 2004). The trifocal tensor can also be computed on the basis of line matches alone.

Schmid and Zisserman (1997) describe a widely used technique for matching 2D lines based on the average of 15×15 pixel correlation scores evaluated at all pixels along their common line segment intersection.³⁵ In their system, the epipolar geometry is assumed to be known, e.g., computed from point matches. For wide baselines, all possible homographies corresponding to planes passing through the 3D line are used to warp pixels and the maximum correlation score is used. For triplets of images, the trifocal tensor is used to verify that the lines are in geometric correspondence before evaluating the correlations between line segments. Figure 11.22 shows the results of using their system.

Bartoli and Sturm (2003) describe a complete system for extending three view relations (trifocal tensors) computed from manual line correspondences to a full bundle adjustment of all the line and camera parameters. The key to their approach is to use the Plücker coordinates (2.12) to parameterize lines and to directly minimize reprojection errors. It is also possible to represent 3D line segments by their endpoints and to measure either the reprojection error perpendicular to the detected 2D line segments in each image or the 2D errors using an elongated uncertainty ellipse aligned with the line segment direction (Szeliski and Kang 1994).

Instead of reconstructing 3D lines, Bay, Ferrari, and Van Gool (2005) use RANSAC to group lines into likely coplanar subsets. Four lines are chosen at random to compute a homography, which is then verified for these and other plausible line segment matches by evaluating color histogram-based correlation scores. The 2D intersection points of lines belonging to the same plane are then used as virtual measurements to estimate the epipolar geometry, which is more accurate than using the homographies directly.

An alternative to grouping lines into coplanar subsets is to group lines by parallelism. Whenever three or more 2D lines share a common vanishing point, there is a good likelihood that they are parallel in 3D. By finding multiple vanishing points in an image (Section 7.4.3) and establishing correspondences between such vanishing points in different images, the relative rotations between the various images (and often the camera intrinsics) can be directly estimated (Section 11.1.1). Finding an orthogonal set of vanishing points and using these to establish a global orientation is often called invoking the *Manhattan world assumption* (Coughlan and Yuille 1999). A generalized version where streets can meet at non-orthogonal angles was called the *Atlanta world* by Schindler and Dellaert (2004).

³⁵Because lines often occur at depth or orientation discontinuities, it may be preferable to compute correlation scores (or to match color histograms (Bay, Ferrari, and Van Gool 2005)) separately on each side of the line.

Shum, Han, and Szeliski (1998) describe a 3D modeling system that constructs calibrated panoramas from multiple images (Section 11.4.2) and then has the user draw vertical and horizontal lines in the image to demarcate the boundaries of planar regions. The lines are used to establish an absolute rotation for each panorama and are then used (along with the inferred vertices and planes) to build a 3D structure, which can be recovered up to scale from one or more images (Figure 13.20).

A fully automated approach to line-based structure from motion is presented by Werner and Zisserman (2002). In their system, they first find lines and group them by common vanishing points in each image (Section 7.4.3). The vanishing points are then used to calibrate the camera, i.e., to perform a “metric upgrade” (Section 11.1.1). Lines corresponding to common vanishing points are then matched using both appearance (Schmid and Zisserman 1997) and trifocal tensors. These lines are then used to infer planes and a block-structured model for the scene, as described in more detail in Section 13.6.1. More recent work using deep neural networks can also be used to construct 3D wireframe models from one or more images.

Plane-based techniques

In scenes that are rich in planar structures, e.g., in architecture, it is possible to directly estimate homographies between different planes, using either feature-based or intensity-based methods. In principle, this information can be used to simultaneously infer the camera poses and the plane equations, i.e., to compute plane-based structure from motion.

Luong and Faugeras (1996) show how a fundamental matrix can be directly computed from two or more homographies using algebraic manipulations and least squares. Unfortunately, this approach often performs poorly, because the algebraic errors do not correspond to meaningful reprojection errors (Szeliski and Torr 1998).

A better approach is to *hallucinate* virtual point correspondences within the areas from which each homography was computed and to feed them into a standard structure from motion algorithm (Szeliski and Torr 1998). An even better approach is to use full bundle adjustment with explicit plane equations, as well as additional constraints to force reconstructed co-planar features to lie exactly on their corresponding planes. (A principled way to do this is to establish a coordinate frame for each plane, e.g., at one of the feature points, and to use 2D in-plane parameterizations for the other points.) The system developed by Shum, Han, and Szeliski (1998) shows an example of such an approach, where the directions of lines and normals for planes in the scene are prespecified by the user. In more recent work, Micusik and Wildenauer (2017) use planes as additional constraints inside a bundle adjustment formulation. Other recent papers that use combinations of lines and/or planes to reduce drift in

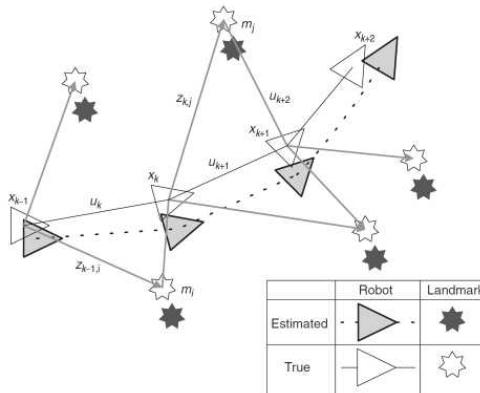


Figure 11.23 In simultaneous localization and mapping (SLAM), the system simultaneously estimates the positions of a robot and its nearby landmarks (Durrant-Whyte and Bailey 2006) © 2006 IEEE.

3D reconstructions include (Zhou, Zou *et al.* 2015), Li, Yao *et al.* (2018), Yang and Scherer (2019), and Holynski, Geraghty *et al.* (2020).

11.5 Simultaneous localization and mapping (SLAM)

While the computer vision community has been studying structure from motion, i.e., the reconstruction of sparse 3D models from multiple images and videos, since the early 1980s (Longuet-Higgins 1981), the mobile robotics community has in parallel been studying the automatic construction of 3D maps from moving robots.³⁶ In robotics, the problem was formulated as the simultaneous estimation of 3D robot and *landmark* poses (Figure 11.23), and was known as *probabilistic mapping* (Thrun, Burgard, and Fox 2005) and *simultaneous localization and mapping* (SLAM) (Durrant-Whyte and Bailey 2006; Bailey and Durrant-Whyte 2006; Cadena, Carlone *et al.* 2016). In the computer vision community, the problem was originally called *visual odometry* (Levin and Szeliski 2004; Nistér, Naroditsky, and Bergen 2006; Maimone, Cheng, and Matthies 2007), although that term is now usually reserved for shorter-range motion estimation that does not involve building a global map with *loop closing* (Cadena, Carlone *et al.* 2016).

Early versions of such algorithms used range-sensing techniques, such as ultrasound, laser

³⁶In the 1980s, the vision and robotics communities were essentially the same set of researchers working in these two sub-fields of artificial intelligence.

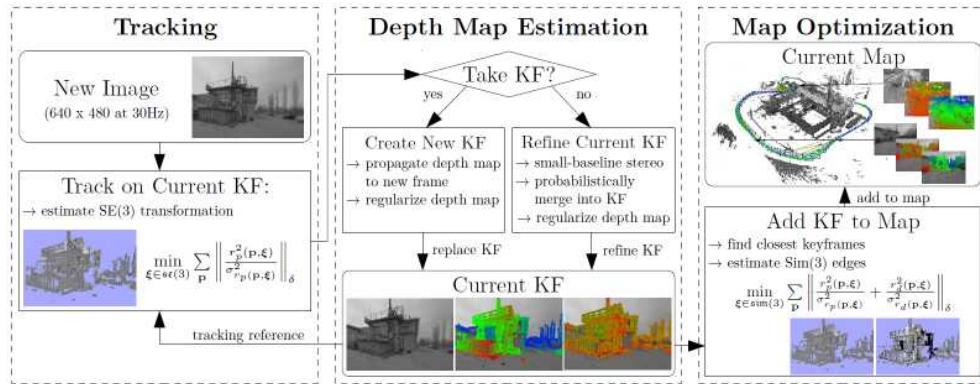


Figure 11.24 The architecture of the LSD-SLAM system (Engel, Schöps, and Cremers 2014) © 2014 Springer, showing the front end, which does the tracking, data association, and local 3D pose and structure (depth map) updating, and the back end, which does global map optimization.

range finders, or stereo matching, to estimate local 3D geometry, which could then be fused into a 3D model. Newer techniques can perform the same task based purely on visual feature tracking from a monocular camera (Davison, Reid *et al.* 2007). Good introductory tutorials can be found in Durrant-Whyte and Bailey (2006) and Bailey and Durrant-Whyte (2006), while more comprehensive surveys of more recent techniques are presented in (Fuentes-Pacheco, Ruiz-Ascencio, and Rendón-Mancha 2015) and Cadena, Carlone *et al.* (2016).

SLAM differs from bundle adjustment in two fundamental aspects. First, it allows for a variety of sensing devices, instead of just being restricted to tracked or matched feature points. Second, it solves the localization problem *online*, i.e., with no or very little lag in providing the current sensor pose. This makes it the method of choice for both time-critical robotics applications such as autonomous navigation (Section 11.5.1) and real-time augmented reality (Section 11.5.2).

Some of the important milestones in SLAM include:

- the application of SLAM to monocular cameras (MonoSLAM) (Davison, Reid *et al.* 2007);
- parallel tracking and mapping (PTAM) (Klein and Murray 2007), which split the front end (tracking) and back end (mapping) processes (Figure 11.24) onto two separate threads running at different rates (Figure 11.27) and then implemented the whole process on a camera phone (Klein and Murray 2009);

- adaptive relative bundle adjustment (Sibley, Mei *et al.* 2009, 2010), which maintains collections of local reconstructions anchored at different keyframes;
- incremental smoothing and mapping (iSAM) (Kaess, Ranganathan, and Dellaert 2008; Kaess, Johannsson *et al.* 2012) and other applications of factor graphs to handle the speed-accuracy-delay tradeoff (Dellaert and Kaess 2017; Dellaert 2021);
- dense tracking and mapping (DTAM) (Newcombe, Lovegrove, and Davison 2011), which estimates and updates a dense depth map for every frame;
- ORB-SLAM (Mur-Artal, Montiel, and Tardos 2015) and ORB-SLAM2 (Mur-Artal and Tardós 2017), which handle monocular, stereo, and RGB-D cameras as well as loop closures;
- SVO (semi-direct visual odometry) (Forster, Zhang *et al.* 2017), which combines patch-based tracking with classic bundle adjustment; and
- LSD-SLAM (large-scale direct SLAM) (Engel, Schöps, and Cremers 2014) and DSO (direct sparse odometry) (Engel, Koltun, and Cremers 2018), which only keep depth estimates at strong gradient locations (Figure 11.24).
- BAD SLAM (bundle adjusted direct RGB-D SLAM) (Schöps, Sattler, and Pollefeys 2019a).

Many of these systems have open source implementations. Some widely used benchmarks include a benchmark for RGB-D SLAM systems (Sturm, Engelhard *et al.* 2012), the KITTI Visual Odometry / SLAM benchmark (Geiger, Lenz *et al.* 2013), the synthetic ICL-NUIM dataset (Handa, Whelan *et al.* 2014), the TUM monoVO dataset (Engel, Usenko, and Cremers 2016), the EuRoC MAV dataset (Burri, Nikolic *et al.* 2016), the ETH3D SLAM benchmark (Schöps, Sattler, and Pollefeys 2019a), and the GSLAM general SLAM benchmark (Zhao, Xu *et al.* 2019).

The most recent trend in SLAM has been the integration with visual-inertial odometry (VIO) algorithms (Mourikis and Roumeliotis 2007; Li and Mourikis 2013; Forster, Carlone *et al.* 2016), which combine higher-frequency inertial measurement unit (IMU) measurements with visual tracks, which serve to remove low-frequency drift. Because IMUs are now commonplace in consumer devices such as cell phones and action cameras, VIO-enhanced SLAM systems serve as the foundation for widely used mobile augmented reality frameworks such as ARKit and ARCore (Section 11.5.2). A dataset and evaluation of open-source VIO systems can be found at Schubert, Goll *et al.* (2018).

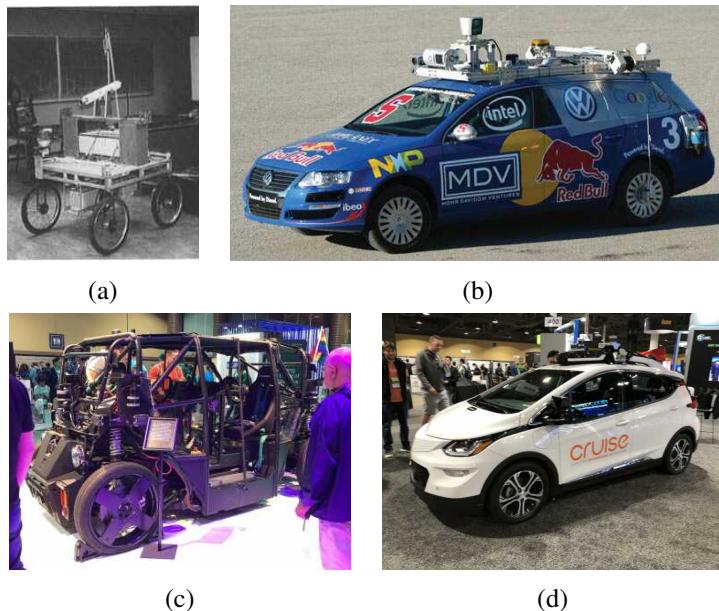


Figure 11.25 Autonomous vehicles: (a) the Stanford Cart (Moravec 1983) ©1983 IEEE; (b) Junior: The Stanford entry in the Urban Challenge (Montemerlo, Becker et al. 2008) © 2008 Wiley; (c–d) self-driving car prototypes from the CVPR 2019 exhibit floor.

As you can tell from this very brief overview, SLAM is an incredibly rich and rapidly evolving field of research, full of challenging robust optimization and real-time performance problems. A good source for finding a list of the most recent papers and algorithms is the KITTI Visual Odometry/SLAM Evaluation³⁷ (Geiger, Lenz, and Urtasun 2012) and the recent survey paper on computer vision for autonomous driving (Janai, Güney *et al.* 2020, Section 13.2).

11.5.1 Application: Autonomous navigation

Since the early days of artificial intelligence and robotics, computer vision has been used to enable manipulation for dexterous robots and navigation for autonomous robots (Janai, Güney *et al.* 2020; Kubota 2019). Some of the earliest vision-based navigation systems include the Stanford Cart (Figure 11.25a) and CMU Rover (Moravec 1980, 1983), the Terregator (Wallace, Stentz *et al.* 1985), and the CMU Nablabb (Thorpe, Hebert *et al.* 1988), which

³⁷http://www.cvlibs.net/datasets/kitti/eval_odometry.php

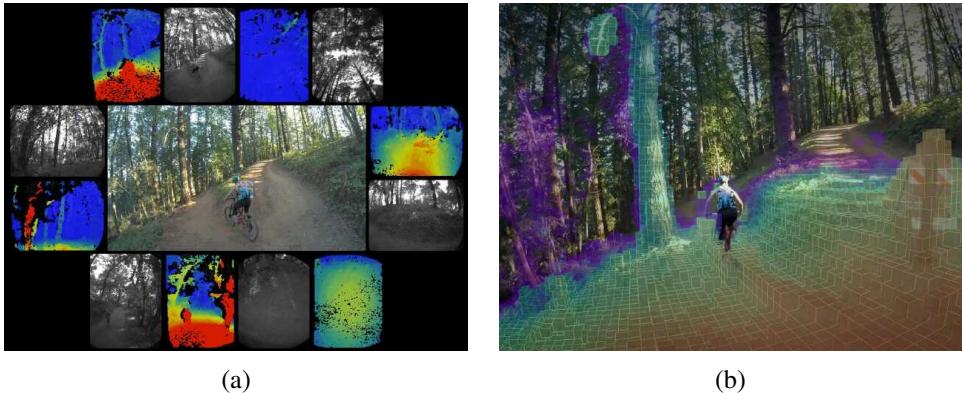


Figure 11.26 Fully autonomous Skydio R1 drone flying in the wild © 2019 Skydio: (a) multiple input images and depth maps; (b) fully integrated 3D map (Cross 2019).

originally could only advance 4m every 10 sec (< 1 mph), and which was also the first system to use a neural network for driving (Pomerleau 1989).

The early algorithms and technologies advanced rapidly, with the VaMoRs system of Dickmanns and Mysliwetz (1992) operating a 25Hz Kalman filter loop and driving with good lane markings at 100 km/h. By the mid 2000s, when DARPA introduced their Grand Challenge and Urban Challenge, vehicles equipped with both range-finding lidar cameras and stereo cameras were able to traverse rough outdoor terrain and navigate city streets at regular human driving speeds (Urmson, Anhalt *et al.* 2008; Montemerlo, Becker *et al.* 2008).³⁸ These systems led to the formation of industrial research projects at companies such as Google and Tesla,³⁹ as well numerous startups, many of which exhibit their vehicles at computer vision conferences (Figure 11.25c–d).

A comprehensive review of computer vision technologies for autonomous vehicles can be found in the survey by Janai, Güney *et al.* (2020), which also comes with a useful on-line visualization tool of relevant papers.⁴⁰ The survey contains chapters on the large number of vision algorithms and components that go into autonomous navigation, which include datasets and benchmarks, sensors, object detection and tracking, segmentation, stereo, flow and scene flow, SLAM, scene understanding, and end-to-end learning of autonomous driving behaviors.

In addition to autonomous navigation for wheeled (and legged) robots and vehicles, com-

³⁸Algorithms that use range data as part of their map building and localization are commonly called *RGB-D SLAM* systems (Sturm, Engelhard *et al.* 2012).

³⁹You can find a number of talks about Tesla's efforts on Andrej Karpathy's web page, <https://karpathy.ai>.

⁴⁰http://www.cvlabs.net/projects/autonomous_vision_survey



Figure 11.27 3D augmented reality: (a) Darth Vader and a horde of Ewoks battle it out on a table-top recovered using real-time, keyframe-based structure from motion (Klein and Murray 2007) © 2007 IEEE; (b) a virtual teapot is fixed to the top of a real-world coffee cup, whose pose is re-recognized at each time frame (Gordon and Lowe 2006) © 2007 Springer.

puter vision algorithms are widely used in the control of autonomous drones for both recreational applications (Ackerman 2019) (Figure 11.26) and drone racing (Jung, Hwang *et al.* 2018; Kaufmann, Gehrig *et al.* 2019). A great talk describing Skydio’s approach to visual autonomous navigation by Gareth Cross (2019) can be found in the ICRA 2019 Workshop on Algorithms and Architectures for Learning In-The-Loop Systems in Autonomous Flight⁴¹ as well as Lecture 23 in Pieter Abbeel’s (2019) class on Advanced Robotics, which has dozens of other interesting related lectures.

11.5.2 Application: Smartphone augmented reality

Another closely related application is *augmented reality*, where 3D objects are inserted into a video feed in real time, often to annotate or help users understand a scene (Azuma, Baillet *et al.* 2001; Feiner 2002; Billinghurst, Clark, and Lee 2015). While traditional systems require prior knowledge about the scene or object being visually tracked (Rosten and Drummond 2005), newer systems can simultaneously build up a model of the 3D environment and then track it so that graphics can be superimposed (Reitmayr and Drummond 2006; Wagner, Reitmayr *et al.* 2008).

Klein and Murray (2007) describe a *parallel tracking and mapping* (PTAM) system, which simultaneously applies full bundle adjustment to keyframes selected from a video stream, while performing robust real-time pose estimation on intermediate frames (Figure 11.27a).

⁴¹<https://uav-learning-icra.github.io/2019>

Once an initial 3D scene has been reconstructed, a dominant plane is estimated (in this case, the table-top) and 3D animated characters are virtually inserted. Klein and Murray (2008) extend this system to handle even faster camera motion by adding edge features, which can still be tracked even when interest points become too blurred. They also use a direct (intensity-based) rotation estimation algorithm for even faster motions.

Instead of modeling the whole scene as one rigid reference frame, Gordon and Lowe (2006) first build a 3D model of an individual object using feature matching and structure from motion. Once the system has been initialized, for every new frame they find the object and its pose using a 3D instance recognition algorithm, and then superimpose a graphical object onto that model, as shown in Figure 11.27b.

While reliably tracking such objects and environments is now a well-solved problem, with frameworks such as ARKit,⁴² ARCore,⁴³ and Spark AR⁴⁴ being widely used for mobile AR application development, determining which pixels should be occluded by foreground scene elements (Chuang, Agarwala *et al.* 2002; Wang and Cohen 2009) still remains an active research area.

One recent example of such work is the Smartphone AR system developed by Valentin, Kowdle *et al.* (2018) shown in Figure 11.28. The system proceeds by generating a semi-dense depth map by matching the current frame to a previous keyframe using a CRF followed by a filtering step. This map is then interpolated to full resolution using a novel planar bilateral solver, and the resulting depth map used for occlusion effects. As accurate per-pixel depth is such an essential component of augmented reality effects, we are likely to see rapid progress in this area, using both active and passive depth sensing technologies.

11.6 Additional reading

Camera calibration was first studied in photogrammetry (Brown 1971; Slama 1980; Atkinson 1996; Kraus 1997) but it has also been widely studied in computer vision (Tsai 1987; Gremban, Thorpe, and Kanade 1988; Chappleboux, Lavallée *et al.* 1992b; Zhang 2000; Grossberg and Nayar 2001). Vanishing points observed either from rectahedral calibration objects or architecture are often used to perform rudimentary calibration (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008). Performing camera calibration without using known targets is known as

⁴²<https://developer.apple.com/augmented-reality>

⁴³<https://developers.google.com/ar>

⁴⁴<https://sparkar.facebook.com/ar-studio>

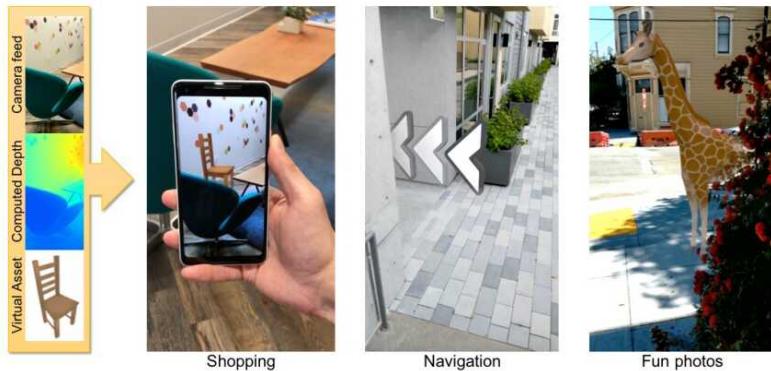


Figure 11.28 Smartphone augmented reality showing real-time depth occlusion effects (Valentin, Kowdle et al. 2018) © 2018 ACM.

self-calibration and is discussed in textbooks and surveys on structure from motion (Faugeras, Luong, and Maybank 1992; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). One popular subset of such techniques uses pure rotational motion (Stein 1995; Hartley 1997b; Hartley, Hayman et al. 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003).

The topic of registering 3D point datasets is called *absolute orientation* (Horn 1987) and *3D pose estimation* (Lorusso, Eggert, and Fisher 1995). A variety of techniques has been developed for simultaneously computing 3D point correspondences and their corresponding rigid transformations (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan et al. 1998; David, DeMenthon et al. 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). When only 2D observations are available, a variety of algorithms for the *linear PnP (perspective n-point)* have been developed (DeMenthon and Davis 1995; Quan and Lan 1999; Moreno-Noguer, Lepetit, and Fua 2007; Terzakis and Lourakis 2020). More recent approaches to pose estimation use deep networks (Arandjelovic, Gronat et al. 2016; Brachmann, Krull et al. 2017; Xiang, Schmidt et al. 2018; Oberweger, Rad, and Lepetit 2018; Hu, Hugonot et al. 2019; Peng, Liu et al. 2019). Estimating pose from RGB-D images is also very active (Drost, Ulrich et al. 2010; Brachmann, Michel et al. 2016; Labb  , Carpentier et al. 2020). In addition to recognizing object pose for robotics tasks, pose estimation is widely used in location recognition (Sattler, Zhou et al. 2019; Revaud, Weinzaepfel et al. 2019; Zhou, Sattler et al. 2019; Sarlin, DeTone et al. 2020; Luo, Zhou et al. 2020).

The topic of structure from motion is extensively covered in books and review articles on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van

Gool, and Vergauwen 2010) with survey of more recent developments in Özyeşil, Voroninski *et al.* (2017). For two-frame reconstruction, Hartley (1997a) wrote a highly cited paper on the “eight-point algorithm” for computing an essential or fundamental matrix with reasonable point normalization. When the cameras are calibrated, the five-point algorithm of Nistér (2004) can be used in conjunction with RANSAC to obtain initial reconstructions from the minimum number of points. When the cameras are uncalibrated, various self-calibration techniques can be found in work by Hartley and Zisserman (2004) and Moons, Van Gool, and Vergauwen (2010).

Triggs, McLauchlan *et al.* (1999) provide a good tutorial and survey on bundle adjustment, while Lourakis and Argyros (2009) and Engels, Stewénius, and Nistér (2006) provide tips on implementation and effective practices. Bundle adjustment is also covered in textbooks and surveys on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). Techniques for handling larger problems are described by Snavely, Seitz, and Szeliski (2008b), Agarwal, Snavely *et al.* (2009), Agarwal, Snavely *et al.* (2010), Jeong, Nistér *et al.* (2012), Wu (2013), Heinly, Schönberger *et al.* (2015), Schönberger and Frahm (2016), and Dellaert and Kaess (2017). While bundle adjustment is often called as an inner loop inside incremental reconstruction algorithms (Snavely, Seitz, and Szeliski 2006), hierarchical (Fitzgibbon and Zisserman 1998; Farenzena, Fusiello, and Gherardi 2009) and global (Rother and Carlsson 2002; Martinec and Pajdla 2007; Sinha, Steedly, and Szeliski 2010; Jiang, Cui, and Tan 2013; Moulou, Monasse, and Marlet 2013; Wilson and Snavely 2014; Cui and Tan 2015; Özyeşil and Singer 2015; Holynski, Geraghty *et al.* 2020) approaches for initialization are also possible and perhaps even preferable.

In the robotics community, techniques for reconstructing a 3D environment from a moving robot are called *simultaneous localization and mapping* (SLAM) (Thrun, Burgard, and Fox 2005; Durrant-Whyte and Bailey 2006; Bailey and Durrant-Whyte 2006; Fuentes-Pacheco, Ruiz-Ascencio, and Rendón-Mancha 2015; Cadena, Carlone *et al.* 2016). SLAM differs from bundle adjustment in that it allows for a variety of sensing devices and that it solves the localization problem *online*. This makes it the method of choice for both time-critical robotics applications such as autonomous navigation (Janai, Güney *et al.* 2020) and real-time augmented reality (Valentin, Kowdle *et al.* 2018). Important papers in this field include (Davison, Reid *et al.* 2007; Klein and Murray 2007, 2009; Newcombe, Lovegrove, and Davison 2011; Kaess, Johannsson *et al.* 2012; Engel, Schöps, and Cremers 2014; Mur-Artal and Tardós 2017; Forster, Zhang *et al.* 2017; Dellaert and Kaess 2017; Engel, Koltun, and Cremers 2018; Schöps, Sattler, and Pollefeyns 2019a) as well as papers that integrate SLAM with IMUs to obtain *visual inertial odometry* (VIO) (Mourikis and Roumeliotis 2007; Li and Mourikis 2013; Forster, Carlone *et al.* 2016; Schubert, Goll *et al.* 2018).

11.7 Exercises

Ex 11.1: Rotation-based calibration. Take an outdoor or indoor sequence from a rotating camera with very little parallax and use it to calibrate the focal length of your camera using the techniques described in Section 11.1.3 or Sections 8.2.3–8.3.1.

1. Take out any radial distortion in the images using one of the techniques from Exercises 11.5–11.6 or using parameters supplied for a given camera by your instructor.
2. Detect and match feature points across neighboring frames and chain them into feature tracks.
3. Compute homographies between overlapping frames and use Equations (11.8–11.9) to get an estimate of the focal length.
4. Compute a full 360° panorama and update your focal length estimate to close the gap (Section 8.2.4).
5. (Optional) Perform a complete bundle adjustment in the rotation matrices and focal length to obtain the highest quality estimate (Section 8.3.1).

Ex 11.2: Target-based calibration. Use a three-dimensional target to calibrate your camera.

1. Construct a three-dimensional calibration pattern with known 3D locations. It is not easy to get high accuracy unless you use a machine shop, but you can get close using heavy plywood and printed patterns.
2. Find the corners, e.g., using a line finder and intersecting the lines.
3. Implement one of the iterative calibration and pose estimation algorithms described in Tsai (1987), Bogart (1991), or Gleicher and Witkin (1992) or the system described in Section 11.2.2.
4. Take many pictures at different distances and orientations relative to the calibration target and report on both your re-projection errors and accuracy. (To do the latter, you may need to use simulated data.)

Ex 11.3: Calibration accuracy. Compare the three calibration techniques (plane-based, rotation-based, and 3D-target-based).

One approach is to have a different student implement each one and to compare the results. Another approach is to use synthetic data, potentially re-using the software you developed

for Exercise 2.3. The advantage of using synthetic data is that you know the ground truth for the calibration and pose parameters, you can easily run lots of experiments, and you can synthetically vary the noise in your measurements.

Here are some possible guidelines for constructing your test sets:

1. Assume a medium-wide focal length (say, 50° field of view).
2. For the plane-based technique, generate a 2D grid target and project it at different inclinations.
3. For a 3D target, create an inner cube corner and position it so that it fills most of field of view.
4. For the rotation technique, scatter points uniformly on a sphere until you get a similar number of points as for other techniques.

Before comparing your techniques, predict which one will be the most accurate (normalize your results by the square root of the number of points used).

Add varying amounts of noise to your measurements and describe the noise sensitivity of your various techniques.

Ex 11.4: Single view metrology. Implement a system to measure dimensions and reconstruct a 3D model from a single image of an architectural scene using visible vanishing directions (Section 11.1.2) (Criminisi, Reid, and Zisserman 2000).

1. Find the three orthogonal vanishing points from parallel lines and use them to establish the three coordinate axes (rotation matrix \mathbf{R} of the camera relative to the scene). If two of the vanishing points are finite (not at infinity), use them to compute the focal length, assuming a known image center. Otherwise, find some other way to calibrate your camera; you could use some of the techniques described by Schaffalitzky and Zisserman (2000).
2. Click on a ground plane point to establish your origin and click on a point a known distance away to establish the scene scale. This lets you compute the translation \mathbf{t} between the camera and the scene. As an alternative, click on a pair of points, one on the ground plane and one above it, and use the known height to establish the scene scale.
3. Write a user interface that lets you click on ground plane points to recover their 3D locations. (Hint: you already know the camera matrix, so knowledge of a point's z

value is sufficient to recover its 3D location.) Click on pairs of points (one on the ground plane, one above it) to measure vertical heights.

4. Extend your system to let you draw quadrilaterals in the scene that correspond to axis-aligned rectangles in the world, using some of the techniques described by Sinha, Steedly *et al.* (2008). Export your 3D rectangles to a VRML or PLY⁴⁵ file.
5. (Optional) Warp the pixels enclosed by the quadrilateral using the correct homography to produce a texture map for each planar polygon.

Ex 11.5: Radial distortion with plumb lines. Implement a plumb-line algorithm to determine the radial distortion parameters.

1. Take some images of scenes with lots of straight lines, e.g., hallways in your home or office, and try to get some of the lines as close to the edges of the image as possible.
2. Extract the edges and link them into curves, as described in Section 7.2.2 and Exercise 7.8.
3. Fit quadratic or elliptic curves to the linked edges using a generalization of the successive line approximation algorithm described in Section 7.4.1 and Exercise 7.11 and keep the curves that fit this form well.
4. For each curved segment, fit a straight line and minimize the perpendicular distance between the curve and the line while adjusting the radial distortion parameters.
5. Alternate between re-fitting the straight line and adjusting the radial distortion parameters until convergence.

Ex 11.6: Radial distortion with a calibration target. Use a grid calibration target to determine the radial distortion parameters.

1. Print out a planar calibration target, mount it on a stiff board, and get it to fill your field of view.
2. Detect the squares, lines, or dots in your calibration target.
3. Estimate the homography mapping the target to the camera from the central portion of the image that does not have any radial distortion.

⁴⁵<https://meshlab.net>.

4. Predict the positions of the remaining targets and use the differences between the observed and predicted positions to estimate the radial distortion.
5. (Optional) Fit a general spline model (for severe distortion) instead of the quartic distortion model.
6. (Optional) Extend your technique to calibrate a fisheye lens.

Ex 11.7: Chromatic aberration. Use the radial distortion estimates for each color channel computed in the previous exercise to clean up wide-angle lens images by warping all of the channels into alignment. (Optional) Straighten out the images at the same time.

Can you think of any reasons why this warping strategy may not always work?

Ex 11.8: Triangulation. Use the calibration pattern you built and tested in Exercise 11.2 to test your triangulation accuracy. As an alternative, generate synthetic 3D points and cameras and add noise to the 2D point measurements.

1. Assume that you know the camera pose, i.e., the camera matrices. Use the 3D distance to rays (11.24) or linearized versions of Equations (11.25–11.26) to compute an initial set of 3D locations. Compare these to your known ground truth locations.
2. Use iterative non-linear minimization to improve your initial estimates and report on the improvement in accuracy.
3. (Optional) Use the technique described by Hartley and Sturm (1997) to perform two-frame triangulation.
4. See if any of the failure modes reported by Hartley and Sturm (1997) or Hartley (1998) occur in practice.

Ex 11.9: Essential and fundamental matrix. Implement the two-frame **E** and **F** matrix estimation techniques presented in Section 11.3, with suitable re-scaling for better noise immunity.

1. Use the data from Exercise 11.8 to validate your algorithms and to report on their accuracy.
2. (Optional) Implement one of the improved **F** or **E** estimation algorithms, e.g., using renormalization (Zhang 1998b; Torr and Fitzgibbon 2004; Hartley and Zisserman 2004), RANSAC (Torr and Murray 1997), least median of squares (LMS), or the five-point algorithm developed by Nistér (2004).

Ex 11.10: View morphing and interpolation. Implement automatic view morphing, i.e., compute two-frame structure from motion and then use these results to generate a smooth animation from one image to the next (Section 11.3.5).

1. Decide how to represent your 3D scene, e.g., compute a Delaunay triangulation of the matched point and decide what to do with the triangles near the border. (Hint: try fitting a plane to the scene, e.g., behind most of the points.)
2. Compute your in-between camera positions and orientations.
3. Warp each triangle to its new location, preferably using the correct perspective projection (Szeliski and Shum 1997).
4. (Optional) If you have a denser 3D model (e.g., from stereo), decide what to do at the “cracks”.
5. (Optional) For a non-rigid scene, e.g., two pictures of a face with different expressions, not all of your matched points will obey the epipolar geometry. Decide how to handle them to achieve the best effect.

Ex 11.11: Bundle adjuster. Implement a full bundle adjuster. This may sound daunting, but it really is not.

1. Devise the internal data structures and external file representations to hold your camera parameters (position, orientation, and focal length), 3D point locations (Euclidean or homogeneous), and 2D point tracks (frame and point identifier as well as 2D locations).
2. Use some other technique, such as factorization, to initialize the 3D point and camera locations from your 2D tracks (e.g., a subset of points that appears in all frames).
3. Implement the code corresponding to the forward transformations in Figure 11.14, i.e., for each 2D point measurement, take the corresponding 3D point, map it through the camera transformations (including perspective projection and focal length scaling), and compare it to the 2D point measurement to get a residual error.
4. Take the residual error and compute its derivatives with respect to all the unknown motion and structure parameters, using backward chaining, as shown, e.g., in Figure 11.14 and Equation (11.19). This gives you the sparse Jacobian \mathbf{J} used in Equations (8.13–8.17) and Equation (11.15).

5. Use a sparse least squares or linear system solver, e.g., MATLAB, SparseSuite, or SPARSKIT (see Appendix A.4 and A.5), to solve the corresponding linearized system, adding a small amount of diagonal preconditioning, as in Levenberg–Marquardt.
6. Update your parameters, make sure your rotation matrices are still orthonormal (e.g., by re-computing them from your quaternions), and continue iterating while monitoring your residual error.
7. (Optional) Use the “Schur complement trick” (11.68) to reduce the size of the system being solved (Triggs, McLauchlan *et al.* 1999; Hartley and Zisserman 2004; Lourakis and Argyros 2009; Engels, Stewénius, and Nistér 2006).
8. (Optional) Implement your own iterative sparse solver, e.g., conjugate gradient, and compare its performance to a direct method.
9. (Optional) Make your bundle adjuster robust to outliers, or try adding some of the other improvements discussed in (Engels, Stewénius, and Nistér 2006). Can you think of any other ways to make your algorithm even faster or more robust?

Ex 11.12: Match move and augmented reality. Use the results of the previous exercise to superimpose a rendered 3D model on top of video. See Section 11.4.4 for more details and ideas. Check for how “locked down” the objects are.

Ex 11.13: Line-based reconstruction. Augment the previously developed bundle adjuster to include lines, possibly with known 3D orientations.

Optionally, use co-planar sets of points and lines to hypothesize planes and to enforce co-planarity (Schaffalitzky and Zisserman 2002; Robertson and Cipolla 2002).

Ex 11.14: Flexible bundle adjuster. Design a bundle adjuster that allows for arbitrary chains of transformations and prior knowledge about the unknowns, as suggested in Figures 11.14–11.15.

Ex 11.15: Unordered image matching. Compute the camera pose and 3D structure of a scene from an arbitrary collection of photographs (Brown and Lowe 2005; Snavely, Seitz, and Szeliski 2006).

Ex 11.16: Augmented reality toolkits. Write a simple mobile AR app based on one of the widely used augmented reality frameworks such as ARKit or ARCore. What fun effects can you create? What are the conditions that make your AR system lose track? Can you move a large distance and come back to your original location without too much drift?

Chapter 12

Depth estimation

12.1	Epipolar geometry	753
12.1.1	Rectification	755
12.1.2	Plane sweep	757
12.2	Sparse correspondence	760
12.2.1	3D curves and profiles	760
12.3	Dense correspondence	762
12.3.1	Similarity measures	764
12.4	Local methods	766
12.4.1	Sub-pixel estimation and uncertainty	768
12.4.2	<i>Application:</i> Stereo-based head tracking	769
12.5	Global optimization	771
12.5.1	Dynamic programming	774
12.5.2	Segmentation-based techniques	775
12.5.3	<i>Application:</i> Z-keying and background replacement	777
12.6	Deep neural networks	778
12.7	Multi-view stereo	781
12.7.1	Scene flow	785
12.7.2	Volumetric and 3D surface reconstruction	786
12.7.3	Shape from silhouettes	794
12.8	Monocular depth estimation	796
12.9	Additional reading	799
12.10	Exercises	800

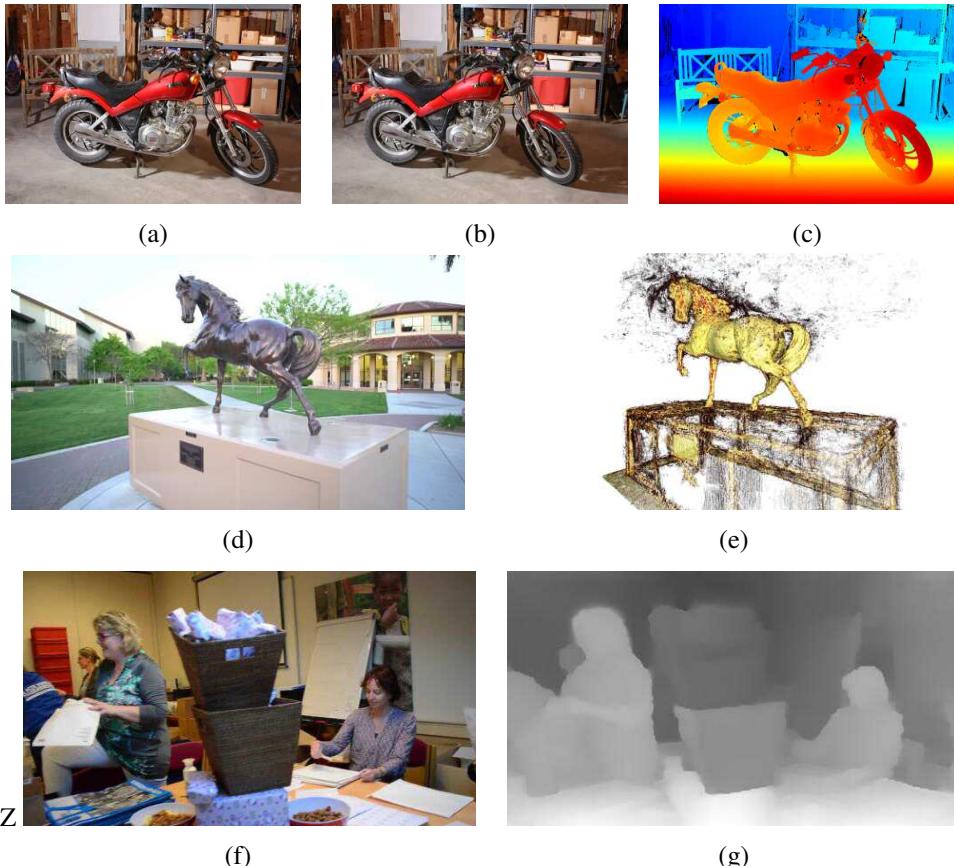


Figure 12.1 Depth estimation algorithms can convert a pair of color images (a–b) into a depth map (c) (Scharstein, Hirschmüller et al. 2014) © 2014 Springer, a sequence of images (d) into a 3D model (e) (Knapitsch, Park et al. 2017) © 2017 ACM, or a single image (f) into a depth map (g) (Li, Dekel et al. 2019) © 2019 IEEE.

Stereo matching is the process of taking two or more images and building a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into 3D depths. In Chapter 11, we described techniques for recovering camera positions and building sparse 3D models of scenes or objects. In this chapter, we address the question of how to build a more complete 3D model, e.g., a sparse or dense *depth map* that assigns relative depths to pixels in the input images. We also look at the topic of *multi-view stereo* algorithms that produce complete 3D volumetric or surface-based object models, as well as *monocular* depth recovery algorithms that infer plausible depths from just a single image.

Why are people interested in depth estimation and stereo matching? From the earliest inquiries into visual perception, it was known that we perceive depth based on the differences in appearance between the left and right eye.¹ As a simple experiment, hold your finger vertically in front of your eyes and close each eye alternately. You will notice that the finger jumps left and right relative to the background of the scene. The same phenomenon is visible in the image pair shown in Figure 12.1a–b, in which the foreground objects shift left and right relative to the background.

As we will shortly see, under simple imaging configurations (both eyes or cameras looking straight ahead), the amount of horizontal motion or *disparity* is inversely proportional to the distance from the observer. While the basic physics and geometry relating visual disparity to scene structure are well understood (Section 12.1), automatically measuring this disparity by establishing dense and accurate inter-image *correspondences* is a challenging task.

The earliest stereo matching algorithms were developed in the field of *photogrammetry* for automatically constructing topographic elevation maps from overlapping aerial images. Prior to this, operators would use photogrammetric stereo plotters, which displayed shifted versions of such images to each eye and allowed the operator to float a dot cursor around constant elevation contours. The development of fully automated stereo matching algorithms was a major advance in this field, enabling much more rapid and less expensive processing of aerial imagery (Hannah 1974; Hsieh, McKeown, and Perlant 1992).

In computer vision, the topic of stereo matching has been one of the most widely studied and fundamental problems (Marr and Poggio 1976; Barnard and Fischler 1982; Dhond and Aggarwal 1989; Scharstein and Szeliski 2002; Brown, Burschka, and Hager 2003; Seitz, Curless *et al.* 2006), and continues to be one of the most active research areas (Poggi, Tosi *et al.* 2021). While photogrammetric matching concentrated mainly on aerial imagery, computer vision applications include modeling the human visual system (Marr 1982), robotic navigation and manipulation (Moravec 1983; Konolige 1997; Thrun, Montemerlo *et al.* 2006; Janai,

¹The word *stereo* comes from the Greek for *solid*; stereo vision is how we perceive solid shape (Koenderink 1990).

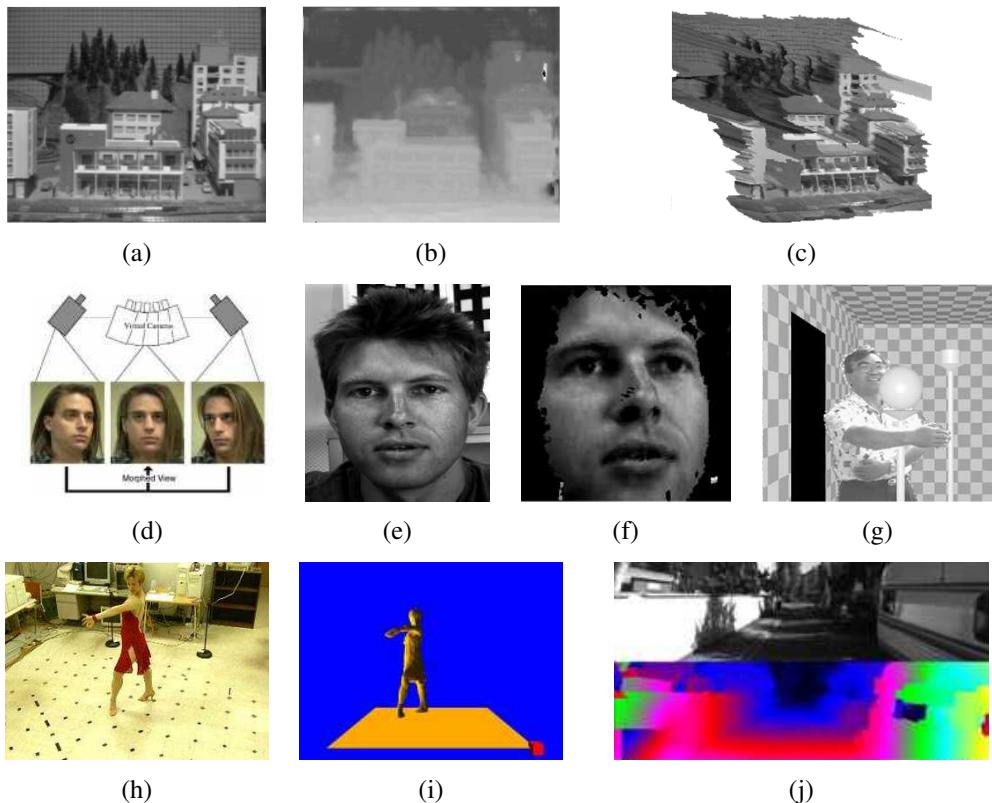


Figure 12.2 Applications of stereo vision: (a) input image, (b) computed depth map, and (c) new view generation from multi-view stereo (Matthies, Kanade, and Szeliski 1989) © 1989 Springer; (d) view morphing between two images (Seitz and Dyer 1996) © 1996 ACM; (e–f) 3D face modeling (images courtesy of Frédéric Devernay); (g) z-keying live and computer-generated imagery (Kanade, Yoshida et al. 1996) © 1996 IEEE; (h–i) building 3D surface models from multiple video streams in Virtualized Reality (Kanade, Rander, and Narayanan 1997) © 1997 IEEE; (j) computing depth maps for autonomous navigation (Geiger, Lenz, and Urtasun 2012) © 2012 IEEE.

Güney *et al.* 2020) and Figures 12.2j and 11.26, as well as view interpolation and image-based rendering (Figure 12.2a–d), 3D model building (Figure 12.2e–f and h–i), mixing live action with computer-generated imagery (Figure 12.2g), and augmented reality (Valentin, Kowdle *et al.* 2018; Chaurasia, Nieuwoudt *et al.* 2020) and Figure 11.28.

In this chapter, we describe the fundamental principles behind stereo matching, following the general taxonomy proposed by Scharstein and Szeliski (2002). We begin in Section 12.1 with a review of the *geometry* of stereo image matching, i.e., how to compute for a given pixel in one image the range of possible locations the pixel might appear at in the other image, i.e., its *epipolar line*. We describe how to pre-warp images so that corresponding epipolar lines are coincident (*rectification*). We also describe a general resampling algorithm called *plane sweep* that can be used to perform multi-image stereo matching with arbitrary camera configurations.

Next, we briefly survey techniques for the *sparse* stereo matching of interest points and edge-like features (Section 12.2). We then turn to the main topic of this chapter, namely the estimation of a *dense* set of pixel-wise correspondences in the form of a *disparity map* (Figure 12.1c). This involves first selecting a pixel matching criterion (Section 12.3) and then using either local area-based aggregation (Section 12.4), global optimization (Section 12.5), or deep networks (Section 12.6), to help disambiguate potential matches. In Section 12.7, we discuss *multi-view stereo* that use more than pairs of images in order to produce higher-quality depth maps or complete 3D object or scene models (Figure 12.1d–e). Finally, in Section 12.8 we present algorithms for inferring depth from just a single image, which has now become possible using machine learning and deep networks.

Throughout this chapter, we will often refer to datasets and benchmarks that have been used to develop depth inference algorithms and gauge their performance. Of these, the most widely used and influential include the Middlebury stereo and multi-view datasets benchmarks, which were among the first to keep up-to-date leaderboards, the EPFL multi-view dataset, the KITTI benchmarks for autonomous driving (stereo, flow, scene flow, and others), the DTU dataset, ETH3D benchmark, Tanks and Temples benchmark, and BlendedMVS dataset, which are all summarized in Table 12.1. Pointers to additional datasets can be found in Mayer, Ilg *et al.* (2018), Janai, Güney *et al.* (2020), Laga, Jospin *et al.* (2020), and Poggi, Tosi *et al.* (2021).

12.1 Epipolar geometry

Given a pixel in one image, how can we compute its correspondence in the other image? In Chapter 9, we saw that a variety of search techniques can be used to match pixels based on

Name/URL	Contents/Reference
Middlebury stereo https://vision.middlebury.edu/stereo	33 high-resolution stereo pairs (Scharstein, Hirschmüller <i>et al.</i> 2014)
Middlebury multi-view https://vision.middlebury.edu/mview	6 3D objects scanned from 300+ views (Seitz, Curless <i>et al.</i> 2006)
EPFL (no longer active)	6 outdoor multi-view sets of images (Strecha, von Hansen <i>et al.</i> 2008)
KITTI 2015 http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php	200 train + 200 test stereo pairs (Menze and Geiger 2015)
DTU https://roboimagedata.compute.dtu.dk/?page_id=36	124 toy scenes with 49–64 images each (Jensen, Dahl <i>et al.</i> 2014)
Freiburg Scene Flow https://lmb.informatik.uni-freiburg.de/resources/datasets	39k synthetic stereo pairs (Mayer, Ilg <i>et al.</i> 2018)
ETH3D https://www.eth3d.net	13 training + 12 test high-res scenes (Schöps, Schönberger <i>et al.</i> 2017)
Tanks and Temples https://www.tanksandtemples.org	7 training + 14 test 4K video scenes (Knapitsch, Park <i>et al.</i> 2017)
BlendedMVS https://github.com/YoYo000/BlendedMVS	17k MVS images covering 113 scenes (Yao, Luo <i>et al.</i> 2020)

Table 12.1 Widely used stereo datasets and benchmarks.

their local appearance as well as the motions of neighboring pixels. In the case of stereo matching, however, we have some additional information available, namely the positions and calibration data for the cameras that took the pictures of the same static scene (Section 11.3).

How can we exploit this information to reduce the number of potential correspondences, and hence both speed up the matching and increase its reliability? Figure 12.3a shows how a pixel in one image x_0 projects to an *epipolar line segment* in the other image. The segment is bounded at one end by the projection of the original viewing ray at infinity p_∞ and at the other end by the projection of the original camera center c_0 into the second camera, which is known as the *epipole* e_1 . If we project the epipolar line in the second image back into the first, we get another line (segment), this time bounded by the other corresponding epipole e_0 . Extending both line segments to infinity, we get a pair of corresponding *epipolar lines* (Figure 12.3b), which are the intersection of the two image planes with the *epipolar plane* that passes through both camera centers c_0 and c_1 as well as the point of interest p (Faugeras and Luong 2001; Hartley and Zisserman 2004).

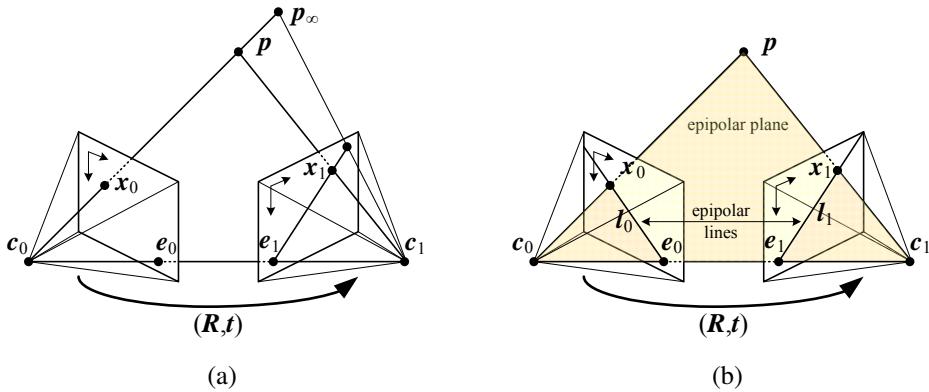


Figure 12.3 Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.

12.1.1 Rectification

As we saw in Section 11.3, the epipolar geometry for a pair of cameras is implicit in the relative pose and calibrations of the cameras, and can easily be computed from seven or more point matches using the fundamental matrix (or five or more points for the calibrated essential matrix) (Zhang 1998a,b; Faugeras and Luong 2001; Hartley and Zisserman 2004). Once this geometry has been computed, we can use the epipolar line corresponding to a pixel in one image to constrain the search for corresponding pixels in the other image. One way to do this is to use a general correspondence algorithm, such as optical flow (Section 9.3), but to only consider locations along the epipolar line (or to project any flow vectors that fall off back onto the line).

A more efficient algorithm can be obtained by first *rectifying* (i.e., warping) the input images so that corresponding horizontal scanlines are epipolar lines (Loop and Zhang 1999; Faugeras and Luong 2001; Hartley and Zisserman 2004).² Afterwards, it is possible to match horizontal scanlines independently or to shift images horizontally while computing matching scores (Figure 12.4).

A simple way to rectify the two images is to first rotate both cameras so that they are looking perpendicular to the line joining the camera centers c_0 and c_1 . As there is a degree of freedom in the *tilt*, the smallest rotations that achieve this should be used. Next, to

²This makes most sense if the cameras are next to each other, although by rotating the cameras, rectification can be performed on any pair that is not *verged* too much or has too much of a scale change. In those latter cases, using plane sweep (below) or hypothesizing small planar patch locations in 3D (Goesle, Snavely *et al.* 2007) may be preferable.

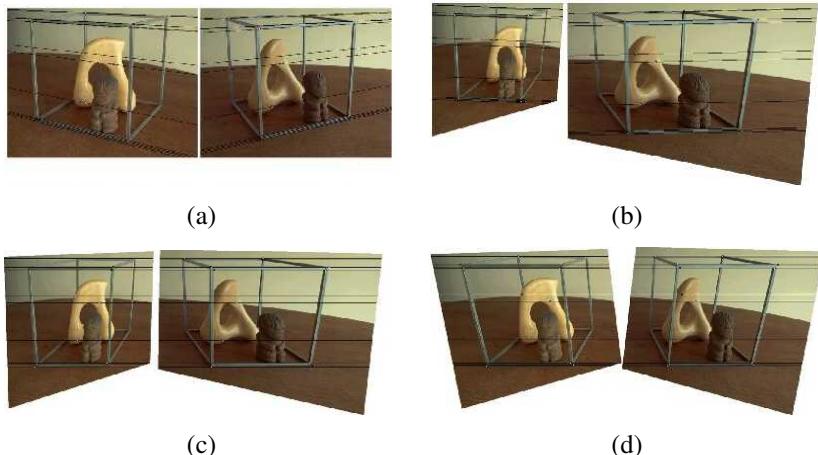


Figure 12.4 The multi-stage stereo rectification algorithm of Loop and Zhang (1999) © 1999 IEEE. (a) Original image pair overlaid with several epipolar lines; (b) images transformed so that epipolar lines are parallel; (c) images rectified so that epipolar lines are horizontal and in vertical correspondence; (d) final rectification that minimizes horizontal distortions.

determine the desired twist around the optical axes, make the *up vector* (the camera y -axis) perpendicular to the camera center line. This ensures that corresponding epipolar lines are horizontal and that the disparity for points at infinity is 0. Finally, re-scale the images, if necessary, to account for different focal lengths, magnifying the smaller image to avoid aliasing. (The full details of this procedure can be found in Fusello, Trucco, and Verri (2000) and Exercise 12.1.) When additional information about the imaging process is available, e.g., that the images were formed on co-planar photographic plates, more specialized and accurate algorithms can be developed (Luo, Kong *et al.* 2020). Note that in general, it is not possible to rectify an arbitrary collection of images simultaneously unless their optical centers are collinear, although rotating the cameras so that they all point in the same direction reduces the inter-camera pixel movements to scalings and translations.

The resulting *standard rectified geometry* is employed in a lot of stereo camera setups and stereo algorithms, and leads to a very simple inverse relationship between 3D depths Z and disparities d ,

$$d = f \frac{B}{Z}, \quad (12.1)$$

where f is the focal length (measured in pixels), B is the baseline, and

$$x' = x + d(x, y), \quad y' = y \quad (12.2)$$

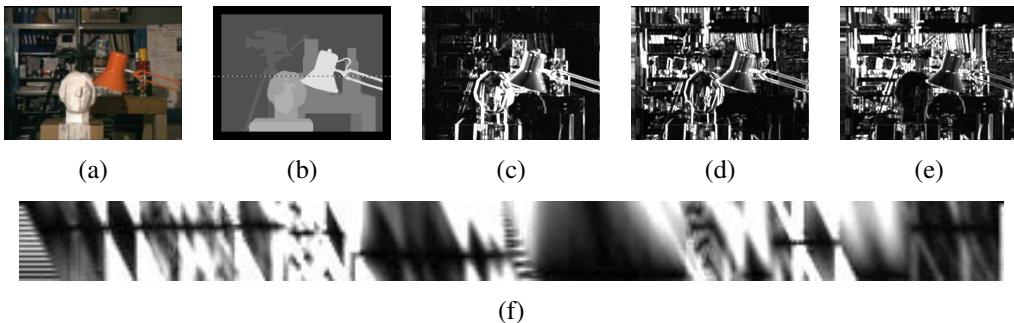


Figure 12.5 Slices through a typical disparity space image (DSI) (Scharstein and Szeliski 2002) © 2002 Springer: (a) original color image; (b) ground truth disparities; (c–e) three (x, y) slices for $d = 10, 16, 21$; (f) an (x, d) slice for $y = 151$ (the dashed line in (b)). Various dark (matching) regions are visible in (c–e), e.g., the bookshelves, table and cans, and head statue, and three disparity levels can be seen as horizontal lines in (f). The dark bands in the DSIs indicate regions that match at this disparity. (Smaller dark regions are often the result of textureless regions.) Additional examples of DSIs are discussed by Bobick and Intille (1999).

describes the relationship between corresponding pixel coordinates in the left and right images (Bolles, Baker, and Marimont 1987; Okutomi and Kanade 1993; Scharstein and Szeliski 2002).³ The task of extracting depth from a set of images then becomes one of estimating the *disparity map* $d(x, y)$.

After rectification, we can easily compare the similarity of pixels at corresponding locations (x, y) and $(x', y') = (x + d, y)$ and store them in a *disparity space image* (DSI) $C(x, y, d)$ for further processing (Figure 12.5). The concept of the disparity space (x, y, d) dates back to early work in stereo matching (Marr and Poggio 1976), while the concept of a disparity space image (volume) is generally associated with Yang, Yuille, and Lu (1993) and Intille and Bobick (1994).

12.1.2 Plane sweep

An alternative to pre-rectifying the images before matching is to sweep a set of planes through the scene and to measure the *photoconsistency* of different images as they are re-projected onto these planes (Figure 12.6). This process is commonly known as the *plane sweep* algorithm (Collins 1996; Szeliski and Golland 1999; Saito and Kanade 1999).

³The term *disparity* was first introduced in the human vision literature to describe the difference in location of corresponding features seen by the left and right eyes (Marr 1982). Horizontal disparity is the most commonly

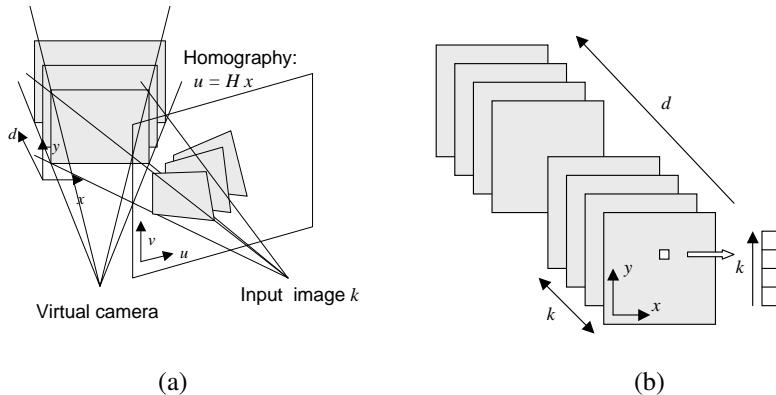


Figure 12.6 Sweeping a set of planes through a scene (Szeliski and Golland 1999) © 1999 Springer: (a) The set of planes seen from a virtual camera induces a set of homographies in any other source (input) camera image. (b) The warped images from all the other cameras can be stacked into a generalized disparity space volume $\tilde{I}(x, y, d, k)$ indexed by pixel location (x, y) , disparity d , and camera k .

As we saw in Section 2.1.4, where we introduced projective depth (also known as *plane plus parallax* (Kumar, Anandan, and Hanna 1994; Sawhney 1994; Szeliski and Coughlan 1997)), the last row of a full-rank 4×4 projection matrix $\tilde{\mathbf{P}}$ can be set to an arbitrary plane equation $\mathbf{p}_3 = s_3[\hat{\mathbf{n}}_0|c_0]$. The resulting four-dimensional projective transform (*collineation*) (2.68) maps 3D world points $\mathbf{p} = (X, Y, Z, 1)$ into screen coordinates $\mathbf{x}_s = (x_s, y_s, 1, d)$, where the *projective depth* (or *parallax*) d (2.66) is 0 on the reference plane (Figure 2.11).

Sweeping d through a series of disparity hypotheses, as shown in Figure 12.6a, corresponds to mapping each input image into the *virtual camera* $\tilde{\mathbf{P}}$ defining the disparity space through a series of homographies (2.68–2.71),

$$\tilde{\mathbf{x}}_k \sim \tilde{\mathbf{P}}_k \tilde{\mathbf{P}}^{-1} \mathbf{x}_s = \tilde{\mathbf{H}}_k \tilde{\mathbf{x}} + \mathbf{t}_k d = (\tilde{\mathbf{H}}_k + \mathbf{t}_k [0 \ 0 \ d]) \tilde{\mathbf{x}}, \quad (12.3)$$

as shown in Figure 2.12b, where $\tilde{\mathbf{x}}_k$ and $\tilde{\mathbf{x}}$ are the homogeneous pixel coordinates in the source and virtual (reference) images (Szeliski and Golland 1999). The members of the family of homographies $\tilde{\mathbf{H}}_k(d) = \tilde{\mathbf{H}}_k + \mathbf{t}_k [0 \ 0 \ d]$, which are parameterized by the addition of a rank-1 matrix, are related to each other through a *planar homology* (Hartley and Zisserman 2004, A5.2).

The choice of virtual camera and parameterization is application dependent and is what gives this framework a lot of its flexibility. In many applications, one of the input cameras (the

studied phenomenon, but vertical disparity is possible if the eyes are verged.

reference camera) is used, thus computing a depth map that is registered with one of the input images and which can later be used for image-based rendering (Sections 14.1 and 14.2). In other applications, such as view interpolation for gaze correction in video-conferencing (Section 12.4.2) (Ott, Lewis, and Cox 1993; Criminisi, Shotton *et al.* 2003), a camera centrally located between the two input cameras is preferable, because it provides the needed per-pixel disparities to hallucinate the virtual middle image.

The choice of disparity sampling, i.e., the setting of the zero parallax plane and the scaling of integer disparities, is also application dependent, and is usually set to bracket the range of interest, i.e., the *working volume*, while scaling disparities to sample the image in pixel (or sub-pixel) shifts. For example, when using stereo vision for obstacle avoidance in robot navigation, it is most convenient to set up disparity to measure per-pixel elevation above the ground (Ivanchenko, Shen, and Coughlan 2009).

As each input image is warped onto the current planes parameterized by disparity d , it can be stacked into a *generalized disparity space image* $\tilde{I}(x, y, d, k)$ for further processing (Figure 12.6b) (Szeliski and Golland 1999). In most stereo algorithms, the photoconsistency (e.g., sum of squared or robust differences) with respect to the reference image I_r is calculated and stored in the DSI

$$C(x, y, d) = \sum_k \rho(\tilde{I}(x, y, d, k) - I_r(x, y)). \quad (12.4)$$

However, it is also possible to compute alternative statistics such as robust variance, focus, or entropy (Section 12.3.1) (Vaish, Szeliski *et al.* 2006) or to use this representation to reason about occlusions (Szeliski and Golland 1999; Kang and Szeliski 2004). The generalized DSI will come in particularly handy when we come back to the topic of multi-view stereo in Section 12.7.2.

Of course, planes are not the only surfaces that can be used to define a 3D sweep through the space of interest. Cylindrical surfaces, especially when coupled with panoramic photography (Section 8.2), are often used (Ishiguro, Yamamoto, and Tsuji 1992; Kang and Szeliski 1997; Shum and Szeliski 1999; Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007). It is also possible to define other manifold topologies, e.g., ones where the camera rotates around a fixed axis (Seitz 2001).

Once the DSI has been computed, the next step in most stereo correspondence algorithms is to produce a univalued function in disparity space $d(x, y)$ that best describes the shape of the surfaces in the scene. This can be viewed as finding a surface embedded in the disparity space image that has some optimality property, such as lowest cost and best (piecewise) smoothness (Yang, Yuille, and Lu 1993). Figure 12.5 shows examples of slices through a typical DSI. More figures of this kind can be found in the paper by Bobick and Intille (1999).

12.2 Sparse correspondence

Early stereo matching algorithms were *feature-based*, i.e., they first extracted a set of potentially matchable image locations, using either interest operators or edge detectors, and then searched for corresponding locations in other images using a patch-based metric (Hannah 1974; Marr and Poggio 1979; Mayhew and Frisby 1980; Baker and Binford 1981; Arnold 1983; Grimson 1985; Ohta and Kanade 1985; Bolles, Baker, and Marimont 1987; Matthies, Kanade, and Szeliski 1989; Hsieh, McKeown, and Perlant 1992; Bolles, Baker, and Hannah 1993). This limitation to sparse correspondences was partially due to computational resource limitations, but was also driven by a desire to limit the answers produced by stereo algorithms to matches with high certainty. In some applications, there was also a desire to match scenes with potentially very different illuminations, where edges might be the only stable features (Collins 1996). Such sparse 3D reconstructions could later be interpolated using surface fitting algorithms such as those discussed in Sections 4.2 and 13.3.1.

More recent work in this area has focused on first extracting highly reliable features and then using these as *seeds* to grow additional matches (Zhang and Shan 2000; Lhuillier and Quan 2002; Čech and Šára 2007) or as inputs to a dense per-pixel depth solver (Valentin, Kowdle *et al.* 2018). Similar approaches have also been extended to wide baseline multi-view stereo problems and combined with 3D surface reconstruction (Lhuillier and Quan 2005; Strecha, Tuytelaars, and Van Gool 2003; Goesele, Snavely *et al.* 2007) or free-space reasoning (Taylor 2003), as described in more detail in Section 12.7.

12.2.1 3D curves and profiles

Another example of sparse correspondence is the matching of *profile curves* (or *occluding contours*), which occur at the boundaries of objects (Figure 12.7) and at interior self occlusions, where the surface curves away from the camera viewpoint.

The difficulty in matching profile curves is that in general, the locations of profile curves vary as a function of camera viewpoint. Therefore, matching curves directly in two images and then triangulating these matches can lead to erroneous shape measurements. Fortunately, if three or more closely spaced frames are available, it is possible to fit a local circular arc to the locations of corresponding edgels (Figure 12.7a) and therefore obtain semi-dense curved surface meshes directly from the matches (Figures 12.7c and g). Another advantage of matching such curves is that they can be used to reconstruct surface shape for untextured surfaces, so long as there is a visible difference between foreground and background colors.

Over the years, a number of different techniques have been developed for reconstructing surface shape from profile curves (Giblin and Weiss 1987; Cipolla and Blake 1992; Vaillant

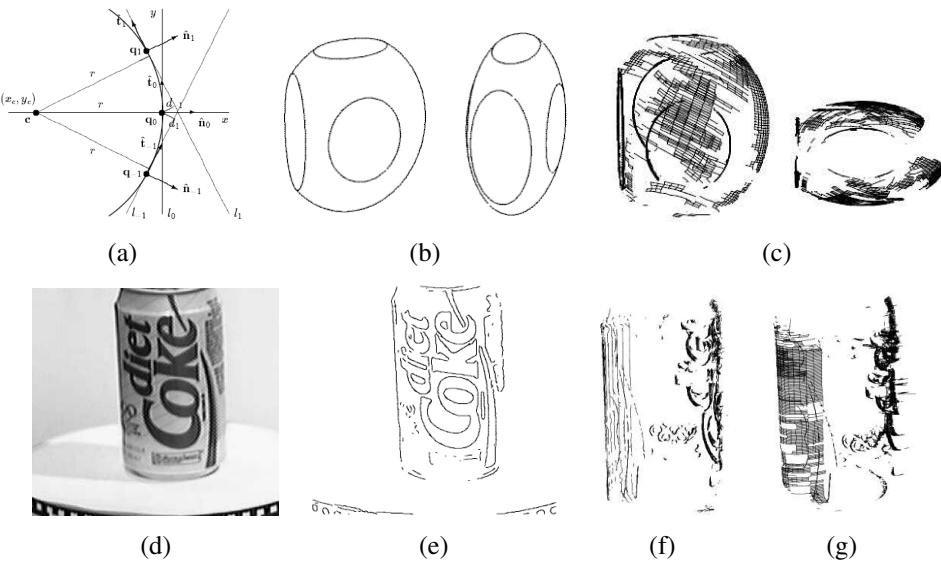


Figure 12.7 Surface reconstruction from occluding contours (Szeliski and Weiss 1998) © 2002 Springer: (a) circular arc fitting in the epipolar plane; (b) synthetic example of an ellipsoid with a truncated side and elliptic surface markings; (c) partially reconstructed surface mesh seen from an oblique and top-down view; (d) real-world image sequence of a soda can on a turntable; (e) extracted edges; (f) partially reconstructed profile curves; (g) partially reconstructed surface mesh. (Partial reconstructions are shown so as not to clutter the images.)

and Faugeras 1992; Zheng 1994; Boyer and Berger 1997; Szeliski and Weiss 1998). Cipolla and Giblin (2000) describe many of these techniques, as well as related topics such as inferring camera motion from profile curve sequences. Below, we summarize the approach developed by Szeliski and Weiss (1998), which assumes a discrete set of images, rather than formulating the problem in a continuous differential framework.

Let us assume that the camera is moving smoothly enough that the local epipolar geometry varies slowly, i.e., the epipolar planes induced by the successive camera centers and an edgel under consideration are nearly co-planar. The first step in the processing pipeline is to extract and link edges in each of the input images (Figures 12.7b and e). Next, edgels in successive images are matched using pairwise epipolar geometry, proximity and (optionally) appearance. This provides a linked set of edges in the spatio-temporal volume, which is sometimes called the *weaving wall* (Baker 1989).

To reconstruct the 3D location of an individual edgel, along with its local in-plane normal

and curvature, we project the viewing rays corresponding to its neighbors onto the instantaneous epipolar plane defined by the camera center, the viewing ray, and the camera velocity, as shown in Figure 12.7a. We then fit an *osculating circle* to the projected lines, from which we can compute a 3D point position (Szeliski and Weiss 1998).

The resulting set of 3D points, along with their spatial (in-image) and temporal (between-image) neighbors, form a 3D surface mesh with local normal and curvature estimates (Figures 12.7c and g). Note that whenever a curve is due to a surface marking or a sharp crease edge, rather than a smooth surface profile curve, this shows up as a 0 or small radius of curvature. Such curves result in isolated 3D space curves, rather than elements of smooth surface meshes, but can still be incorporated into the 3D surface model during a later stage of surface interpolation (Section 13.3.1).

More recent examples of 3D curve reconstruction from sequences of RGB and RGB-D images include (Li, Yao *et al.* 2018; Liu, Chen *et al.* 2018; Wang, Liu *et al.* 2020), the latest of which can even recover camera pose with untextured backgrounds. When the thin structures being modeled are planar manifolds, such as leaves or paper, as opposed to true 3D curves such as wires, specially tailored mesh representations may be more appropriate (Kim, Zimmer *et al.* 2013; Yücer, Kim *et al.* 2016; Yücer, Sorkine-Hornung *et al.* 2016), as discussed in more detail in Sections 12.7.2 and 14.3.

12.3 Dense correspondence

While sparse matching algorithms are still occasionally used, most stereo matching algorithms today focus on dense correspondence, as this is required for applications such as image-based rendering or modeling. This problem is more challenging than sparse correspondence, because inferring depth values in textureless regions requires a certain amount of guesswork. (Think of a solid colored background seen through a picket fence. What depth should it be?)

In this section, we review the taxonomy and categorization scheme for dense correspondence algorithms first proposed by Scharstein and Szeliski (2002). The taxonomy consists of a set of algorithmic “building blocks” from which a large set of algorithms can be constructed. It is based on the observation that stereo algorithms generally perform some subset of the following four steps:

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation and optimization; and

4. disparity refinement.

For example, *local* (window-based) algorithms (Section 12.4), where the disparity computation at a given point depends only on intensity values within a finite window, usually make implicit smoothness assumptions by aggregating support. Some of these algorithms can cleanly be broken down into steps 1, 2, 3. For example, the traditional sum-of-squared-differences (SSD) algorithm can be described as:

1. The matching cost is the squared difference of intensity values at a given disparity.
2. Aggregation is done by summing the matching cost over square windows with constant disparity.
3. Disparities are computed by selecting the minimal (winning) aggregated value at each pixel.

Some local algorithms, however, combine steps 1 and 2 and use a matching cost that is based on a support region, e.g., normalized cross-correlation (Hannah 1974; Bolles, Baker, and Hannah 1993) and the rank transform (Zabih and Woodfill 1994) and other ordinal measures (Bhat and Nayar 1998). (This can also be viewed as a preprocessing step; see Section 12.3.1.)

Global algorithms, on the other hand, make explicit smoothness assumptions and then solve a global optimization problem (Section 12.5). Such algorithms typically do not perform an aggregation step, but rather seek a disparity assignment (step 3) that minimizes a global cost function that consists of data (step 1) terms and smoothness terms. The main distinction among these algorithms is the minimization procedure used, e.g., simulated annealing (Marroquin, Mitter, and Poggio 1987; Barnard 1989), probabilistic (mean-field) diffusion (Scharstein and Szeliski 1998), expectation maximization (EM) (Birchfield, Natarajan, and Tomasi 2007), graph cuts (Boykov, Veksler, and Zabih 2001), or loopy belief propagation (Sun, Zheng, and Shum 2003), to name just a few.

In between these two broad classes are certain iterative algorithms that do not explicitly specify a global function to be minimized, but whose behavior mimics closely that of iterative optimization algorithms (Marr and Poggio 1976; Zitnick and Kanade 2000). Hierarchical (coarse-to-fine) algorithms resemble such iterative algorithms, but typically operate on an image pyramid where results from coarser levels are used to constrain a more local search at finer levels (Witkin, Terzopoulos, and Kass 1987; Quam 1984; Bergen, Anandan *et al.* 1992). Also situated between local and global methods is *semi-global-matching* (SGM) (Hirschmüller 2008), which approximates minimizing a 2D cost function via 1D optimization (see Section 12.5.1), as well as methods that avoid exploring the whole search space, e.g., PatchMatch stereo (Bleyer, Rhemann, and Rother 2011) and local plane sweeps (LPS)

(Sinha, Scharstein, and Szeliski 2014). A large number of neural network algorithms have also been developed for stereo matching, which we review in Section 12.6.

While most stereo matching algorithms produce a single disparity map with respect to a reference input image, or a path through the disparity space that encodes a continuous surface (Figure 12.13), a few algorithms compute fractional opacity values along with depths and colors for each pixel (Szeliski and Golland 1999; Zhou, Tucker *et al.* 2018; Flynn, Broxton *et al.* 2019). As these are closely related to volumetric reconstruction techniques, we discuss them in Section 12.7.2 as well as Section 14.2.1 on image-based rendering with layers.

12.3.1 Similarity measures

The first component of any dense stereo matching algorithm is a similarity measure that compares pixel values in order to determine how likely they are to be in correspondence. In this section, we briefly review the similarity measures introduced in Section 9.1 and mention a few others that have been developed specifically for stereo matching (Scharstein and Szeliski 2002; Hirschmüller and Scharstein 2009).

The most common pixel-based matching costs include sums of *squared intensity differences* (SSD) (Hannah 1974) and *absolute intensity differences* (SAD) (Kanade 1994). In the video processing community, these matching criteria are referred to as the *mean-squared error* (MSE) and *mean absolute difference* (MAD) measures; the term *displaced frame difference* is also often used (Tekalp 1995).

More recently, robust measures (9.2), including truncated quadratics and contaminated Gaussians, have been proposed (Black and Anandan 1996; Black and Rangarajan 1996; Scharstein and Szeliski 1998; Barron 2019). These measures are useful because they limit the influence of mismatches during aggregation. Vaish, Szeliski *et al.* (2006) compare a number of such robust measures, including a new one based on the entropy of the pixel values at each disparity hypothesis (Zitnick, Kang *et al.* 2004), which is particularly useful in multi-view stereo.

Other traditional matching costs include normalized cross-correlation (9.11) (Hannah 1974; Bolles, Baker, and Hannah 1993; Evangelidis and Psarakis 2008), which behaves similarly to sum-of-squared-differences (SSD), and binary matching costs (i.e., match or no match) (Marr and Poggio 1976), based on binary features such as edges (Baker and Binford 1981; Grimson 1985) or the sign of the Laplacian (Nishihara 1984). Because of their poor discriminability, simple binary matching costs are no longer used in dense stereo matching.

Some costs are insensitive to differences in camera gain or bias, for example gradient-based measures (Seitz 1989; Scharstein 1994), phase and filter-bank responses (Marr and Poggio 1979; Kass 1988; Jenkin, Jepson, and Tsotsos 1991; Jones and Malik 1992), filters

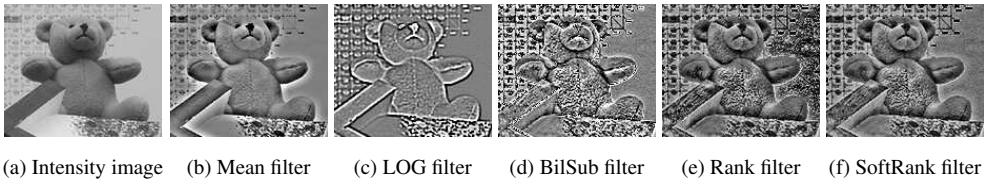


Figure 12.8 *Various similarity measures (pre-processing filters) studied in (Hirschmüller and Scharstein 2009) © 2009 IEEE. The contrast of (b)–(d) has been increased for better visualization.*

that remove regular or robust (bilaterally filtered) means (Ansar, Castano, and Matthies 2004; Hirschmüller and Scharstein 2009), dense feature descriptor (Tola, Lepetit, and Fua 2010), and non-parametric measures such as rank and census transforms (Zabih and Woodfill 1994), ordinal measures (Bhat and Nayar 1998), or entropy (Zitnick, Kang *et al.* 2004; Zitnick and Kang 2007). The census transform, which converts each pixel inside a moving window into a bit vector representing which neighbors are above or below the central pixel, was found by Hirschmüller and Scharstein (2009) to be quite robust against large-scale, non-stationary exposure and illumination changes. Figure 12.8 shows a few of the transformations that can be applied to images to improve their similarity across illumination variations.

It is also possible to correct for differing global camera characteristics by performing a preprocessing or iterative refinement step that estimates inter-image bias–gain variations using global regression (Gennert 1988), histogram equalization (Cox, Roy, and Hingorani 1995), or mutual information (Kim, Kolmogorov, and Zabih 2003; Hirschmüller 2008). Local, smoothly varying compensation fields have also been proposed (Strecha, Tuytelaars, and Van Gool 2003; Zhang, McMillan, and Yu 2006).

To compensate for sampling issues, i.e., dramatically different pixel values in high-frequency areas, Birchfield and Tomasi (1998) proposed a matching cost that is less sensitive to shifts in image sampling. Rather than just comparing pixel values shifted by integral amounts (which may miss a valid match), they compare each pixel in the reference image against a linearly interpolated function of the other image. More detailed studies of these and additional matching costs are explored in Szeliski and Scharstein (2004) and Hirschmüller and Scharstein (2009). In particular, if you expect there to be significant exposure or appearance variation between images that you are matching, some of the more robust measures that performed well in the evaluation by Hirschmüller and Scharstein (2009), such as the census transform (Zabih and Woodfill 1994), ordinal measures (Bhat and Nayar 1998), bilateral subtraction (Ansar, Castano, and Matthies 2004), or hierarchical mutual information (Hirschmüller 2008), should be used. Interestingly, color information does not appear to help when utilized in matching

costs (Bleyer and Chambon 2010), although it is important for aggregation (discussed in next section). When matching more than pairs of images, more sophisticated variants of similarity (photoconsistency) measures can be used, as discussed in Section 12.7 and (Furukawa and Hernández 2015, Chapter 2).

More recently, one of the first successes of deep learning for stereo was the learning of matching costs. Žbontar and LeCun (2016) trained a neural network to compare image patches, trained on data extracted from the Middlebury (Scharstein, Hirschmüller *et al.* 2014) and KITTI (Geiger, Lenz, and Urtasun 2012) datasets. This matching cost is still widely used in top-performing methods on these two benchmarks.

12.4 Local methods

Local and window-based methods aggregate the matching cost by summing or averaging over a *support region* in the DSI $C(x, y, d)$.⁴ A support region can be either two-dimensional at a fixed disparity (favoring fronto-parallel surfaces), or three-dimensional in x - y - d space (supporting slanted surfaces). Two-dimensional evidence aggregation has been implemented using square windows or Gaussian convolution (traditional), multiple windows anchored at different points, i.e., shiftable windows (Arnold 1983; Fusiello, Roberto, and Trucco 1997; Bobick and Intille 1999), windows with adaptive sizes (Okutomi and Kanade 1992; Kanade and Okutomi 1994; Kang, Szeliski, and Chai 2001; Veksler 2001, 2003), windows based on connected components of constant disparity (Boykov, Veksler, and Zabih 1998), the results of color-based segmentation (Yoon and Kweon 2006; Tombari, Mattoccia *et al.* 2008), or with a guided filter (Hosni, Rhemann *et al.* 2013). Three-dimensional support functions that have been proposed include limited disparity difference (Grimson 1985), limited disparity gradient (Pollard, Mayhew, and Frisby 1985), Prazdny's coherence principle (Prazdny 1985), and the work by Zitnick and Kanade (2000), which includes visibility and occlusion reasoning. PatchMatch stereo (Bleyer, Rhemann, and Rother 2011), discussed in more detail below, also does aggregation in 3D via slanted support windows.

Aggregation with a fixed support region can be performed using 2D or 3D convolution,

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d), \quad (12.5)$$

or, in the case of rectangular windows, using efficient moving average box-filters (Section 3.2.2) (Kanade, Yoshida *et al.* 1996; Kimura, Shinbo *et al.* 1999). Shiftable windows can also be implemented efficiently using a separable sliding min-filter (Figure 12.9) (Scharstein

⁴For two surveys and comparisons of such techniques, please see the work of Gong, Yang *et al.* (2007) and Tombari, Mattoccia *et al.* (2008).

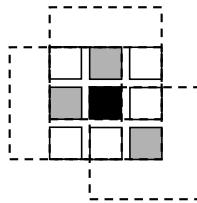


Figure 12.9 Shiftable window (Scharstein and Szeliski 2002) © 2002 Springer. The effect of trying all 3×3 shifted windows around the black pixel is the same as taking the minimum matching score across all centered (non-shifted) windows in the same neighborhood. (For clarity, only three of the neighboring shifted windows are shown here.)

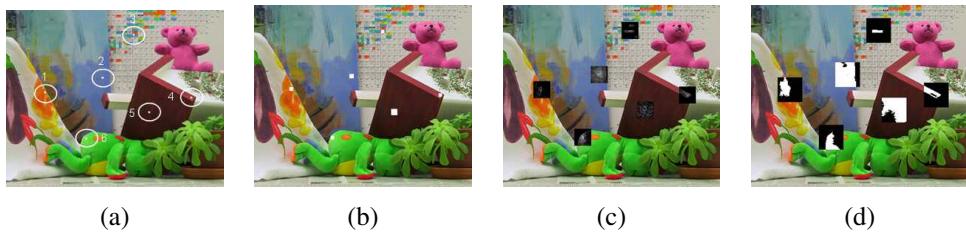


Figure 12.10 Aggregation window sizes and weights adapted to image content (Tombari, Mattoccia et al. 2008) © 2008 IEEE: (a) original image with selected evaluation points; (b) variable windows (Veksler 2003); (c) adaptive weights (Yoon and Kweon 2006); (d) segmentation-based (Tombari, Mattoccia, and Di Stefano 2007). Notice how the adaptive weights and segmentation-based techniques adapt their support to similarly colored pixels.

and Szeliski 2002, Section 4.2). Selecting among windows of different shapes and sizes can be performed more efficiently by first computing a *summed area table* (Section 3.2.3, 3.30–3.32) (Veksler 2003). Selecting the right window is important, because windows must be large enough to contain sufficient texture and yet small enough so that they do not straddle depth discontinuities (Figure 12.10). An alternative method for aggregation is *iterative diffusion*, i.e., repeatedly adding to each pixel’s cost the weighted values of its neighboring pixels’ costs (Szeliski and Hinton 1985; Shah 1993; Scharstein and Szeliski 1998).

Of the local aggregation methods compared by Gong, Yang *et al.* (2007) and Tombari, Mattoccia *et al.* (2008), the fast variable window approach of Veksler (2003) and the locally weighting approach developed by Yoon and Kweon (2006) consistently stood out as having the best tradeoff between performance and speed.⁵ The local weighting technique, in partic-

⁵Extensive results from Tombari, Mattoccia *et al.* (2008) can be found at <http://www.vision.deis.unibo.it/spe>.

ular, is interesting because, instead of using square windows with uniform weighting, each pixel within an aggregation window influences the final matching cost based on its color similarity and spatial distance, just as in bilateral filtering (Figure 12.10c). (In fact, their aggregation step is closely related to doing a joint bilateral filter on the color/disparity image, except that it is done symmetrically in both reference and target images.) The segmentation-based aggregation method of Tombari, Mattoccia, and Di Stefano (2007) did even better, although a fast implementation of this algorithm does not yet exist. Another approach to aggregation is to aggregate along one or more minimum spanning trees based on pixel similarities (Yang 2015; Li, Yu *et al.* 2017).

In local methods, the emphasis is on the matching cost computation and cost aggregation steps. Computing the final disparities is trivial: simply choose at each pixel the disparity associated with the minimum cost value. Thus, these methods perform a local “winner-take-all” (WTA) optimization at each pixel. A limitation of this approach (and many other correspondence algorithms) is that uniqueness of matches is only enforced for one image (the *reference image*), while points in the other image might match multiple points, unless cross-checking and subsequent hole filling is used (Fua 1993; Hirschmüller and Scharstein 2009).

12.4.1 Sub-pixel estimation and uncertainty

Most stereo correspondence algorithms compute a set of disparity estimates in some discretized space, e.g., for integer disparities (exceptions include continuous optimization techniques such as optical flow (Bergen, Anandan *et al.* 1992) or splines (Szeliski and Coughlan 1997)). For applications such as robot navigation or people tracking, these may be perfectly adequate. However for image-based rendering, such quantized maps lead to very unappealing view synthesis results, i.e., the scene appears to be made up of many thin shearing layers. To remedy this situation, many algorithms apply a sub-pixel refinement stage after the initial discrete correspondence stage. (An alternative is to simply start with more discrete disparity levels (Szeliski and Scharstein 2004).)

Sub-pixel disparity estimates can be computed in a variety of ways, including iterative gradient descent and fitting a curve to the matching costs at discrete disparity levels (Ryan, Gray, and Hunt 1980; Lucas and Kanade 1981; Tian and Huhns 1986; Matthies, Kanade, and Szeliski 1989; Kanade and Okutomi 1994). This provides an easy way to increase the resolution of a stereo algorithm with little additional computation. However, to work well, the intensities being matched must vary smoothly, and the regions over which these estimates are computed must be on the same (correct) surface.

Some questions have been raised about the advisability of fitting correlation curves to

integer-sampled matching costs (Shimizu and Okutomi 2001). This situation may even be worse when sampling-insensitive dissimilarity measures are used (Birchfield and Tomasi 1998). These issues are explored in more depth by Szeliski and Scharstein (2004) and Haller and Nedevschi (2012).

Besides sub-pixel computations, there are other ways of post-processing the computed disparities. Occluded areas can be detected using cross-checking, i.e., comparing left-to-right and right-to-left disparity maps (Fua 1993). A median filter can be applied to clean up spurious mismatches, and holes due to occlusion can be filled by surface fitting or by distributing neighboring disparity estimates (Birchfield and Tomasi 1999; Scharstein 1999; Hirschmüller and Scharstein 2009).

Another kind of post-processing, which can be useful in later processing stages, is to associate *confidences* with per-pixel depth estimates (Figure 12.11), which can be done by looking at the curvature of the correlation surface, i.e., how strong the minimum in the DSI image is at the winning disparity. Matthies, Kanade, and Szeliski (1989) show that under the assumption of small noise, photometrically calibrated images, and densely sampled disparities, the variance of a local depth estimate can be estimated as

$$Var(d) = \frac{\sigma_I^2}{a}, \quad (12.6)$$

where a is the curvature of the DSI as a function of d , which can be measured using a local parabolic fit or by squaring all the horizontal gradients in the window, and σ_I^2 is the variance of the image noise, which can be estimated from the minimum SSD score. (See also Section 8.1.4, (9.37), and Appendix B.6.) Over the years, a variety of stereo confidence measures have been proposed. Hu and Mordohai (2012) and Poggi, Kim *et al.* (2021) provide thorough surveys of this topic.

12.4.2 Application: Stereo-based head tracking

A common application of real-time stereo algorithms is for tracking the position of a user interacting with a computer or game system. The use of stereo can dramatically improve the reliability of such a system compared to trying to use monocular color and intensity information (Darrell, Gordon *et al.* 2000). Once recovered, this information can be used in a variety of applications, including controlling a virtual environment or game, correcting the apparent gaze during video conferencing, and background replacement. We discuss the first two applications below and defer the discussion of background replacement to Section 12.5.3.

The use of head tracking to control a user’s virtual viewpoint while viewing a 3D object or environment on a computer monitor is sometimes called *fish tank virtual reality*, as the

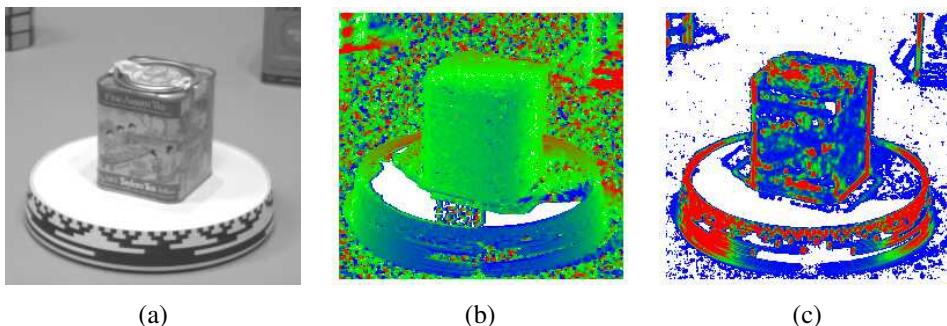


Figure 12.11 *Uncertainty in stereo depth estimation (Szeliski 1991b): (a) input image; (b) estimated depth map (blue is closer); (c) estimated confidence(red is higher). As you can see, more textured areas have higher confidence.*

user is observing a 3D world as if it were contained inside a fish tank (Ware, Arthur, and Booth 1993). Early versions of these systems used mechanical head tracking devices and stereo glasses. Today, such systems can be controlled using stereo-based head tracking and stereo glasses can be replaced with autostereoscopic displays. Head tracking can also be used to construct a “virtual mirror”, where the user’s head can be modified in real time using a variety of visual effects (Darrell, Baker *et al.* 1997).

Another application of stereo head tracking and 3D reconstruction is in gaze correction (Ott, Lewis, and Cox 1993). When a user participates in a desktop video conference or video chat, the camera is usually placed on top of the monitor. Because the person is gazing at a window somewhere on the screen, it appears as if they are looking down and away from the other participants, instead of straight at them. Replacing the single camera with two or more cameras enables a virtual view to be constructed right at the position where they are looking, resulting in virtual eye contact. Real-time stereo matching is used to construct an accurate 3D head model and view interpolation (Section 14.1) is used to synthesize the novel in-between view (Criminisi, Shotton *et al.* 2003). More recent publications on gaze correction in video conferencing include Kuster, Popa *et al.* (2012) and Kononenko and Lempitsky (2015), and the technology has been deployed in several commercial video conferencing systems.⁶

⁶<https://venturebeat.com/2019/10/03/microsofts-ai-powered-eye-gaze-tech-is-exclusive-to-the-surface-pro-x>

12.5 Global optimization

Global stereo matching methods perform some optimization or iteration steps after the disparity computation phase and often skip the aggregation step altogether, because the global smoothness constraints perform a similar function. Many global methods are formulated in an energy-minimization framework, where, as we saw in Chapters 4 (4.24–4.27) and 9, the objective is to find a solution d that minimizes a global energy,

$$E(d) = E_D(d) + \lambda E_S(d). \quad (12.7)$$

The data term, $E_D(d)$, measures how well the disparity function d agrees with the input image pair. Using our previously defined disparity space image, we define this energy as

$$E_D(d) = \sum_{(x,y)} C(x, y, d(x, y)), \quad (12.8)$$

where C is the (initial or aggregated) matching cost DSI.

The smoothness term $E_S(d)$ encodes the smoothness assumptions made by the algorithm. To make the optimization computationally tractable, the smoothness term is often restricted to measuring only the differences between neighboring pixels' disparities,

$$E_S(d) = \sum_{(x,y)} \rho(d(x, y) - d(x+1, y)) + \rho(d(x, y) - d(x, y+1)), \quad (12.9)$$

where ρ is some monotonically increasing function of disparity difference. It is also possible to use larger neighborhoods, such as \mathcal{N}_8 , which can lead to better boundaries (Boykov and Kolmogorov 2003), or to use second-order smoothness terms (Woodford, Reid *et al.* 2008), but such terms require more complex optimization techniques. An alternative to smoothness functionals is to use a lower-dimensional representation, such as splines (Szeliski and Coughlan 1997).

In standard regularization (Section 4.2), ρ is a quadratic function, which makes d smooth everywhere and may lead to poor results at object boundaries. Energy functions that do not have this problem are called *discontinuity-preserving* and are based on robust ρ functions (Terzopoulos 1986b; Black and Rangarajan 1996). The seminal paper by Geman and Geman (1984) gave a Bayesian interpretation of these kinds of energy functions and proposed a discontinuity-preserving energy function based on Markov random fields (MRFs) and additional *line processes*, which are additional binary variables that control whether smoothness penalties are enforced or not. Black and Rangarajan (1996) show how independent line process variables can be replaced by robust pairwise disparity terms.

The terms in E_S can also be made to depend on the intensity differences, e.g.,

$$\rho_D(d(x, y) - d(x + 1, y)) \cdot \rho_I(\|I(x, y) - I(x + 1, y)\|), \quad (12.10)$$

where ρ_I is some monotonically decreasing function of intensity differences that lowers smoothness costs at high-intensity gradients. This idea (Gamble and Poggio 1987; Fua 1993; Bobick and Intille 1999; Boykov, Veksler, and Zabih 2001) encourages disparity discontinuities to coincide with intensity or color edges and appears to account for some of the good performance of global optimization approaches. While most researchers set these functions heuristically, Pal, Weinman *et al.* (2012) show how the free parameters in such *conditional random fields* (Section 4.3, (4.47)) can be learned from ground truth disparity maps.

Once the global energy has been defined, a variety of algorithms can be used to find a (local) minimum. Traditional approaches associated with regularization and Markov random fields include continuation (Blake and Zisserman 1987), simulated annealing (Geman and Geman 1984; Marroquin, Mitter, and Poggio 1987; Barnard 1989), highest confidence first (Chou and Brown 1990), and mean-field annealing (Geiger and Girosi 1991).

Max-flow and *graph cut* methods have been proposed to solve a special class of global optimization problems (Roy and Cox 1998; Boykov, Veksler, and Zabih 2001; Ishikawa 2003). Such methods are more efficient than simulated annealing and have produced good results, as have techniques based on loopy belief propagation (Sun, Zheng, and Shum 2003; Tappen and Freeman 2003). Appendix B.5 and survey papers on MRF inference (Szeliski, Zabih *et al.* 2008; Blake, Kohli, and Rother 2011; Kappes, Andres *et al.* 2015) discuss and compare such techniques in more detail.

While global optimization techniques have largely been displaced by deep learning approaches (Section 12.6) for datasets such as KITI with large amounts of training images and high overlap with the test distributions, they still perform the best on challenging stereo pairs with fine details such as the high-resolution Middlebury pairs (Scharstein, Hirschmüller *et al.* 2014). One example of such an approach is the local expansion moves algorithm developed by Taniai, Matsushita *et al.* (2018). Below, we describe some related techniques that are of historical interest, run faster, or are tailored to handle specific situations.

Cooperative algorithms. Cooperative algorithms, inspired by computational models of human stereo vision, were among the earliest methods proposed for disparity computation (Dev 1974; Marr and Poggio 1976; Marroquin 1983; Szeliski and Hinton 1985; Zitnick and Kanade 2000). Such algorithms iteratively update disparity estimates using non-linear operations based on neighboring disparity and matching values and result in an overall behavior similar to global optimization algorithms. In fact, for some of these algorithms, it is possible to explicitly state a global function that is being minimized (Scharstein and Szeliski 1998). There



Figure 12.12 Stereo matching using local plane sweeps (Sinha, Scharstein, and Szeliski 2014) © 2014 IEEE: (a) input image; (b) initial sparse matches; (c) matches grouped by slanted planes; (d) 3D visualization of planes and grouped features.

are also iterative algorithms that look at a larger neighborhood in the image, such as Patch-Match Stereo (Bleyer, Rhemann, and Rother 2011), which estimates a local 3D plane at each pixel and uses the non-local PatchMatch algorithm (Barnes, Shechtman *et al.* 2009) to quickly find approximate nearest neighbors in plane space. This approach has recently been applied to the multi-view stereo setting to produce an extremely time and space-efficient high-quality algorithm (Wang, Galliani *et al.* 2021).

Coarse-to-fine and incremental warping. Most of today’s best algorithms first enumerate all possible matches at all possible disparities and then select the best set of matches in some way. Faster approaches can sometimes be obtained using methods inspired by classic (infinitesimal) optical flow computation. Here, images are successively warped and disparity estimates incrementally updated until a satisfactory registration is achieved. These techniques are most often implemented within a coarse-to-fine hierarchical refinement framework (Quam 1984; Bergen, Anandan *et al.* 1992; Barron, Fleet, and Beauchemin 1994; Szeliski and Coughlan 1997). Recently, coarse-to-fine or *pyramid* approaches have been having a renaissance in modern deep networks, applied both to optical flow (Ranjan and Black 2017; Sun, Yang *et al.* 2018) and stereo (Chang and Chen 2018).

Local plane sweeps. Instead of sweeping planes perpendicular to the viewing direction, it is also possible to model the scene using a collection of slanted planes, which is beneficial if the scene contains highly slanted planar surfaces such as floors or walls, as shown in Figure 12.12 (Sinha, Scharstein, and Szeliski 2014). Once such planes have been estimated and pixels assigned to each plane, it is then possible to estimate per-pixel out-of-plane displacements to better model curved surfaces. Slanted planes were also used earlier in the the PatchMatch stereo algorithm (Bleyer, Rhemann, and Rother 2011), and have also been used more recently in the *planar bilateral solver* used for smartphone AR (Valentin, Kowdle *et al.* 2018).

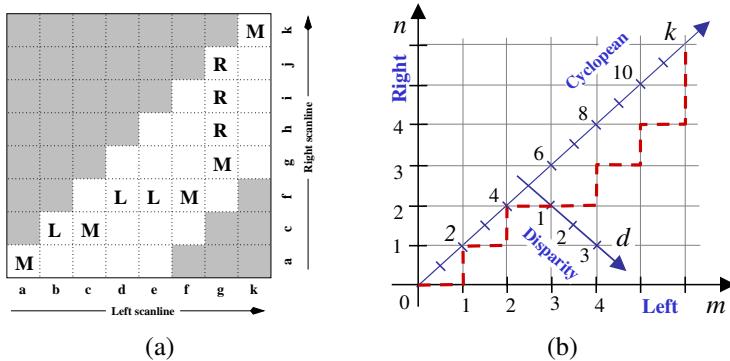


Figure 12.13 Stereo matching using dynamic programming, as illustrated by (a) Scharstein and Szeliski (2002) © 2002 Springer and (b) Kolmogorov, Criminisi et al. (2006) © 2006 IEEE. For each pair of corresponding scanlines, a minimizing path through the matrix of all pairwise matching costs (DSI) is selected. Lowercase letters (a–k) symbolize the intensities along each scanline. Uppercase letters represent the selected path through the matrix. Matches are indicated by M, while partially occluded points (which have a fixed cost) are indicated by L or R, corresponding to points only visible in the left or right images, respectively. Usually, only a limited disparity range is considered (0–4 in the figure, indicated by the non-shaded squares). The representation in (a) allows for diagonal moves while the representation in (b) does not. Note that these diagrams, which use the Cyclopean representation of depth, i.e., depth relative to a camera between the two input cameras, show an “unskewed” x-d slice through the DSI.

12.5.1 Dynamic programming

A different class of global optimization algorithm is based on *dynamic programming*. While the 2D optimization of Equation (12.7) can be shown to be NP-hard for common classes of smoothness functions (Veksler 1999), dynamic programming can find the global minimum for independent scanlines in polynomial time. Dynamic programming was first used for stereo vision in sparse, edge-based methods (Baker and Binford 1981; Ohta and Kanade 1985). More recent approaches have focused on the dense (intensity-based) scanline matching problem (Belhumeur 1996; Geiger, Ladendorf, and Yuille 1992; Cox, Hingorani *et al.* 1996; Bobick and Intille 1999; Birchfield and Tomasi 1999). These approaches work by computing the minimum-cost path through the matrix of all pairwise matching costs between two corresponding scanlines, i.e., through a horizontal slice of the DSI. Partial occlusion is handled explicitly by assigning a group of pixels in one image to a single pixel in the other

image. Figure 12.13 schematically shows how DP works, while Figure 12.5f shows a real DSI slice over which the DP is applied.

To implement dynamic programming for a scanline y , each entry (state) in a 2D cost matrix $D(m, n)$ is computed by combining its DSI matching cost value with one of its predecessor cost values while also including a fixed penalty for occluded pixels. The aggregation rules corresponding to Figure 12.13b are given by Kolmogorov, Criminisi *et al.* (2006), who also use a two-state foreground–background model for bi-layer segmentation.

Problems with dynamic programming stereo include the selection of the right cost for occluded pixels and the difficulty of enforcing inter-scanline consistency, although several methods propose ways of addressing the latter (Ohta and Kanade 1985; Belhumeur 1996; Cox, Hingorani *et al.* 1996; Bobick and Intille 1999; Birchfield and Tomasi 1999; Kolmogorov, Criminisi *et al.* 2006). Another problem is that the dynamic programming approach requires enforcing the *monotonicity* or *ordering constraint* (Yuille and Poggio 1984). This constraint requires that the relative ordering of pixels on a scanline remain the same between the two views, which may not be the case in scenes containing narrow foreground objects.

An alternative to traditional dynamic programming, introduced by Scharstein and Szeliski (2002), is to neglect the vertical smoothness constraints in (12.9) and simply optimize independent scanlines in the global energy function (12.7). The advantage of this *scanline optimization* algorithm is that it computes the same representation and minimizes a reduced version of the same energy function as the full 2D energy function (12.7). Unfortunately, it still suffers from the same streaking artifacts as dynamic programming. Dynamic programming is also possible on tree structures, which can ameliorate the streaking (Veksler 2005).

Much higher quality results can be obtained by summing up the cumulative cost function from multiple directions, e.g., from the eight cardinal directions, N, E, W, S, NE, SE, SW, NW (Hirschmüller 2008). The resulting *semi-global matching* (SGM) algorithm performs quite well and is extremely efficient, enabling real-time low-power implementations (Gehrig, Eberli, and Meyer 2009). Drory, Haubold *et al.* (2014) show that SGM is equivalent to early stopping for a particular variant of belief propagation. Semi-global matching has also been extended using learned components, e.g., SGM-Net (Seki and Pollefeys 2017), which uses a CNN to adjust transition costs, and SGM-Forest (Schönberger, Sinha, and Pollefeys 2018), which uses a random-forest classifier to fuse disparity proposals from different directions.

12.5.2 Segmentation-based techniques

While most stereo matching algorithms perform their computations on a per-pixel basis, some techniques first segment the images into regions and then try to label each region with a disparity.

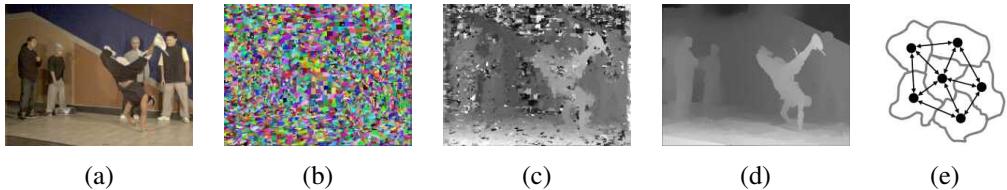


Figure 12.14 Segmentation-based stereo matching (Zitnick, Kang et al. 2004) © 2004 ACM: (a) input color image; (b) color-based segmentation; (c) initial disparity estimates; (d) final piecewise-smoothed disparities; (e) MRF neighborhood defined over the segments in the disparity space distribution (Zitnick and Kang 2007) © 2007 Springer.

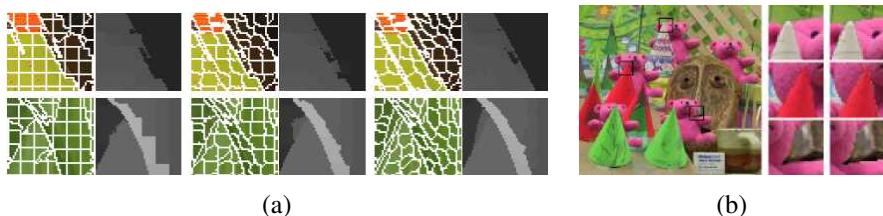


Figure 12.15 Stereo matching with adaptive over-segmentation and matting (Taguchi, Wilburn, and Zitnick 2008) © 2008 IEEE: (a) segment boundaries are refined during the optimization, leading to more accurate results (e.g., the thin green leaf in the bottom row); (b) alpha mattes are extracted at segment boundaries, which leads to visually better compositing results (middle column).

For example, Tao, Sawhney, and Kumar (2001) segment the reference image, estimate per-pixel disparities using a local technique, and then do local plane fits inside each segment before applying smoothness constraints between neighboring segments. Zitnick, Kang *et al.* (2004) and Zitnick and Kang (2007) use over-segmentation to mitigate initial bad segmentations. After a set of initial cost values for each segment has been stored into a *disparity space distribution* (DSD), iterative relaxation (or loopy belief propagation, in the more recent work of Zitnick and Kang (2007)) is used to adjust the disparity estimates for each segment, as shown in Figure 12.14. Taguchi, Wilburn, and Zitnick (2008) refine the segment shapes as part of the optimization process, which leads to much improved results, as shown in Figure 12.15.

Even more accurate results are obtained by Klaus, Sormann, and Karner (2006), who first segment the reference image using mean shift, run a small (3×3) SAD plus gradient SAD (weighted by cross-checking) to get initial disparity estimates, fit local planes, re-fit with global planes, and then run a final MRF on plane assignments with loopy belief propagation.

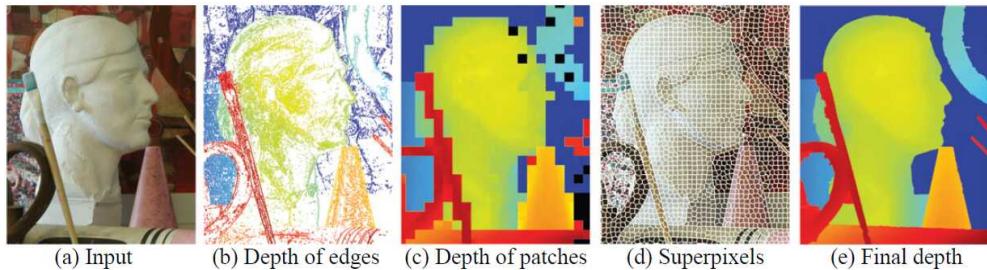


Figure 12.16 Multiframe matching using edges, planes, and superpixels (Xue, Owens et al. 2019) © 2019 Elsevier.

When the algorithm was first introduced in 2006, it was the top ranked algorithm on the existing Middlebury benchmark.

The algorithm by Wang and Zheng (2008) follows a similar approach of segmenting the image, doing local plane fits, and then performing cooperative optimization of neighboring plane fit parameters. The algorithm by Yang, Wang *et al.* (2009), uses the color correlation approach of Yoon and Kweon (2006) and hierarchical belief propagation to obtain an initial set of disparity estimates. Gallup, Frahm, and Pollefeys (2010) segment the image into planar and non-planar regions and use different representations for these two classes of surfaces.

More recently, Xue, Owens *et al.* (2019) start by matching edges across a multi-frame stereo sequence and then fit overlapping square patches to obtain local plane hypotheses. These are then refined using superpixels and a final edge-aware relaxation to get continuous depth maps.

Another important ability of segmentation-based stereo algorithms, which they share with algorithms that use explicit layers (Baker, Szeliski, and Anandan 1998; Szeliski and Golland 1999) or boundary extraction (Hasinoff, Kang, and Szeliski 2006), is the ability to extract fractional pixel alpha mattes at depth discontinuities (Bleyer, Gelautz *et al.* 2009). This ability is crucial when attempting to create virtual view interpolation without clinging boundary or tearing artifacts (Zitnick, Kang *et al.* 2004) and also to seamlessly insert virtual objects (Taguchi, Wilburn, and Zitnick 2008), as shown in Figure 12.15b.

12.5.3 Application: Z-keying and background replacement

Another application of real-time stereo matching is *z-keying*, which is the process of segmenting a foreground actor from the background using depth information, usually for the purpose of replacing the background with some computer-generated imagery, as shown in



Figure 12.17 *Background replacement using z-keying with a bi-layer segmentation algorithm (Kolmogorov, Criminisi et al. 2006) © 2006 IEEE.*

Figure 12.2g.

Originally, z-keying systems required expensive custom-built hardware to produce the desired depth maps in real time and were, therefore, restricted to broadcast studio applications (Kanade, Yoshida *et al.* 1996; Iddan and Yahav 2001). Off-line systems were also developed for estimating 3D multi-viewpoint geometry from video streams (Section 14.5.4) (Kanade, Rander, and Narayanan 1997; Carranza, Theobalt *et al.* 2003; Zitnick, Kang *et al.* 2004; Vedula, Baker, and Kanade 2005). Highly accurate real-time stereo matching subsequently made it possible to perform z-keying on regular PCs, enabling desktop video conferencing applications such as those shown in Figure 12.17 (Kolmogorov, Criminisi *et al.* 2006), but these have mostly been replaced with deep networks for background replacement (Sengupta, Jayaram *et al.* 2020) and real-time 3D phone-based reconstruction algorithms for augmented reality (Figure 11.28 and Valentin, Kowdle *et al.* 2018).

12.6 Deep neural networks

As with other areas of computer vision, deep neural networks and end-to-end learning have had a large impact on stereo matching. In this section, we briefly review how DNNs have been used in stereo correspondence algorithms. We follow the same structure as the two recent surveys by Poggi, Tosi *et al.* (2021) and Laga, Jospin *et al.* (2020), which classify techniques into three categories, namely,

1. learning in the stereo pipeline,
2. end-to-end learning with 2D architectures, and

3. end-to-end learning with 3D architectures.

We briefly discuss a few papers in each group and refer the reader to the full surveys for more details (Janai, Güney *et al.* 2020; Poggi, Tosi *et al.* 2021; Laga, Jospin *et al.* 2020).

Learning in the stereo pipeline

Even before the advent of deep learning, several authors proposed learning components of the traditional stereo pipeline, e.g., to learn hyperparameters of MRF and CRF stereo models (Zhang and Seitz 2007; Pal, Weinman *et al.* 2012). Źbontar and LeCun (2016) were the first to bring deep learning to stereo by training features to optimize a pairwise matching cost. These learned matching costs are still widely used in top-performing methods on the Middlebury stereo evaluation. Many other authors have since proposed CNNs for matching cost computation and aggregation (Luo, Schwing, and Urtasun 2016; Park and Lee 2017; Zhang, Prisacariu *et al.* 2019).

Learning has also been used to improve traditional optimization techniques, in particular the widely used SGM algorithm of Hirschmüller (2008). This includes SGM-Net (Seki and Pollefey 2017), which uses a CNN to adjust transition costs, and SGM-Forest (Schönberger, Sinha, and Pollefey 2018), which uses a random-forest classifier to select among disparity values from multiple incident directions. CNNs have also been used in the refinement stage, replacing earlier techniques such as bilateral filtering (Gidaris and Komodakis 2017; Batsos and Mordohai 2018; Knöbelreiter and Pock 2019).

End-to-end learning with 2D architectures

The availability of large synthetic datasets with ground truth disparities, in particular the Freiburg SceneFlow dataset (Mayer, Ilg *et al.* 2016, 2018) enabled the end-to-end training of stereo networks and resulted in a proliferation of new methods. These methods work well on benchmarks that provide enough training data so that the network can be tuned to the domain, notably KITTI (Geiger, Lenz, and Urtasun 2012; Geiger, Lenz *et al.* 2013; Menze and Geiger 2015), where deep-learning based methods started to dominate the leaderboards in 2016.

The first deep learning architectures for stereo were similar to those designed for dense regression tasks such as semantic segmentation (Chen, Zhu *et al.* 2018). These *2D architectures* typically employ an encoder-decoder design inspired by U-Net (Ronneberger, Fischer, and Brox 2015). The first such model was DispNet-C, introduced in the seminal paper by Mayer, Ilg *et al.* (2016), utilizing a *correlation layer* (Dosovitskiy, Fischer *et al.* 2015) to compute the similarity between image layers.

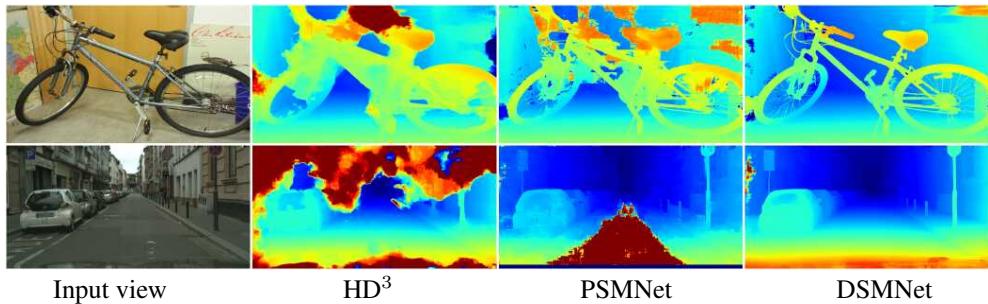


Figure 12.18 Disparity maps computed by three different DNN stereo matchers trained on synthetic data and applied to real-world image pairs (Zhang, Qi et al. 2020) © 2020 Springer.

Subsequent improvements to 2D architectures included the idea of residual networks that apply residual corrections to the original disparities (Pang, Sun et al. 2017), which can also be done in an iterative fashion (Liang, Feng et al. 2018). Coarse-to-fine processing can be used (Tonioni, Tosi et al. 2019; Yin, Darrell, and Yu 2019), and networks can estimate occlusions and depth boundaries (Ilg, Saikia et al. 2018; Song, Zhao et al. 2020) or use neural architecture search (NAS) to improve performance (Saikia, Marrakchi et al. 2019). HITNet incorporates several of these ideas and produces efficient state-of-the-art results using local slanted plane hypotheses and iterative refinement (Tankovich, Hane et al. 2021).

The 2D architecture developed by Knöbelreiter, Reinbacher et al. (2017) uses a joint CNN and Conditional Random Field (CRF) model to infer dense disparity maps. Another promising approach is multi-task learning, for instance, jointly estimating disparities and semantic segmentation (Yang, Zhao et al. 2018; Jiang, Sun et al. 2019). It is also possible to increase the apparent resolution of the output depth map and reduce over-smoothing by representing the output as a bimodal mixture distribution (Tosi, Liao et al. 2021).

End-to-end learning with 3D architectures

An alternative approach is to use *3D architectures*, which explicitly encode geometry by processing features over a 3D volume, where the third dimension corresponds to the disparity search range. In other words, such architectures explicitly represent the disparity space image (DSI), while still keeping multiple feature channels instead of just scalar cost values. Compared to 2D architectures, they incur much higher memory requirements and runtimes.

The first examples of such architectures include GC-Net (Kendall, Martirosyan et al. 2017) and PSMNet (Chang and Chen 2018). 3D architectures also allow the integration of traditional local aggregation methods (Zhang, Prisacariu et al. 2019) and methods to avoid

geometric inconsistencies (Chabra, Straub *et al.* 2019). While resource constraints often mean that 3D DNN-based stereo methods operate at fairly low resolutions, the Hierarchical Stereo Matching (HSM) network (Yang, Manela *et al.* 2019) uses a pyramid approach that selectively restricts the search space at higher resolutions and enables *anytime on-demand inference*, i.e., stopping the processing early for higher frame rates. Duggal, Wang *et al.* (2019) address limited resources by developing a differentiable version of PatchMatch (Bleyer, Rheinmann, and Rother 2011) in a recurrent neural net. Cheng, Zhong *et al.* (2020) use neural architecture search (NAS) to create a state-of-the-art 3D architecture.

While supervised deep learning approaches have come to dominate individual benchmarks that include dedicated training sets such as KITTI, they do not yet generalize well across domains (Zendel *et al.* 2020). On the Middlebury benchmark, which features high-resolution images and only provides very limited training data, deep learning methods are still notably absent. Poggi, Tosi *et al.* (2021) identify the following two major challenges that remain open: (1) generalization across different domains, and (2) applicability on high-resolution images. For cross-domain generalization, Poggi, Tosi *et al.* (2021) describe techniques for both offline and online self-supervised adaptation and guided deep learning, while Laga, Jospin *et al.* (2020) discuss both fine-tuning and data transformation. A recent example of domain generalization is the domain-invariant stereo matching network (DSMNet) of Zhang, Qi *et al.* (2020), which compares favorably with alternative state-of-the-art models such as HD³ (Yin, Darrell, and Yu 2019) and PSMNet (Chang and Chen 2018), as shown in Figure 12.18. Another example of domain adaptation is AdaStereo (Song, Yang *et al.* 2021). For high-resolution images, techniques have been developed to increase resolution in a coarse-to-fine manner (Khamis, Fanello *et al.* 2018; Chabra, Straub *et al.* 2019).

12.7 Multi-view stereo

While matching pairs of images is a useful way of obtaining depth information, using more images can significantly improve results. In this section, we review not only techniques for creating complete 3D object models, but also simpler techniques for improving the quality of depth maps using multiple source images. A good survey of techniques developed up through 2015 can be found in Furukawa and Hernández (2015) and a more recent review in Janai, Güney *et al.* (2020, Chapter 10).

As we saw in our discussion of plane sweep (Section 12.1.2), it is possible to resample all neighboring k images at each disparity hypothesis d into a generalized disparity space volume $\tilde{I}(x, y, d, k)$. The simplest way to take advantage of these additional images is to sum

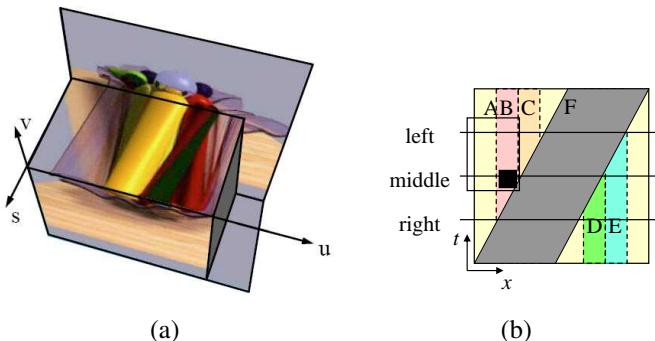


Figure 12.19 Epipolar plane image (EPI) (Gortler, Grzeszczuk et al. 1996) © 1996 ACM and a schematic EPI (Kang, Szeliski, and Chai 2001) © 2001 IEEE. (a) The Lumigraph (light field) (Section 14.3) is the 4D space of all light rays passing through a volume of space. Taking a 2D slice results in all of the light rays embedded in a plane and is equivalent to a scanline taken from a stacked EPI volume. Objects at different depths move sideways with velocities (slopes) proportional to their inverse depth. Occlusion (and translucency) effects can easily be seen in this representation. (b) The EPI corresponding to Figure 12.20 showing the three images (middle, left, and right) as slices through the EPI volume. The spatially and temporally shifted window around the black pixel is indicated by the rectangle, showing that the right image is not being used in matching.

up their differences from the reference image I_r as in (12.4),

$$C(x, y, d) = \sum_k \rho(\tilde{I}(x, y, d, k) - I_r(x, y)). \quad (12.11)$$

This is the basis of the well-known sum of summed-squared-difference (SSSD) and SSAD approaches (Okutomi and Kanade 1993; Kang, Webb et al. 1995), which can be extended to reason about likely patterns of occlusion (Nakamura, Matsuura et al. 1996). More recent work by Gallup, Frahm et al. (2008) shows how to adapt the baselines used to the expected depth to get the best tradeoff between geometric accuracy (wide baseline) and robustness to occlusion (narrow baseline). Alternative multi-view cost metrics include measures such as synthetic focus sharpness and the entropy of the pixel color distribution (Vaish, Szeliski et al. 2006).

A useful way to visualize the multi-frame stereo estimation problem is to examine the *epipolar plane image* (EPI) formed by stacking corresponding scanlines from all the images, as shown in Figures 9.11c and 12.19 (Bolles, Baker, and Marimont 1987; Baker and Bolles 1989; Baker 1989). As you can see in Figure 12.19, as a camera translates horizontally (in a

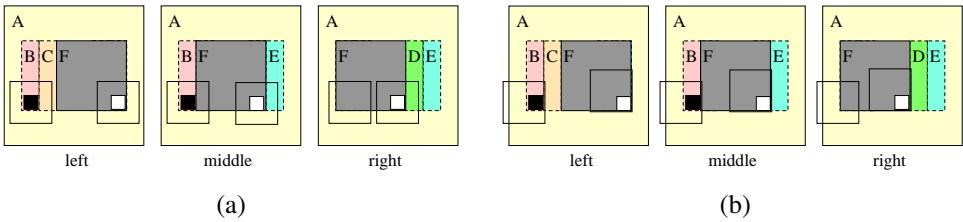


Figure 12.20 Spatio-temporally shiftable windows (Kang, Szeliski, and Chai 2001) © 2001 IEEE: A simple three-image sequence (the middle image is the reference image), which has a moving frontal gray square (marked F) and a stationary background. Regions B, C, D, and E are partially occluded. (a) A regular SSD algorithm will make mistakes when matching pixels in these regions (e.g., the window centered on the black pixel in region B) and in windows straddling depth discontinuities (the window centered on the white pixel in region F). (b) Shiftable windows help mitigate the problems in partially occluded regions and near depth discontinuities. The shifted window centered on the white pixel in region F matches correctly in all frames. The shifted window centered on the black pixel in region B matches correctly in the left image, but requires temporal selection to disable matching the right image. Figure 12.19b shows an EPI corresponding to this sequence and describes in more detail how temporal selection works.

standard horizontally rectified geometry), objects at different depths move sideways at a rate inversely proportional to their depth (12.1).⁷ Foreground objects occlude background objects, which can be seen as *EPI-strips* (Criminisi, Kang *et al.* 2005) occluding other strips in the EPI. If we are given a dense enough set of images, we can find such strips and reason about their relationships to both reconstruct the 3D scene and make inferences about translucent objects (Tsin, Kang, and Szeliski 2006) and specular reflections (Swaminathan, Kang *et al.* 2002; Criminisi, Kang *et al.* 2005). Alternatively, we can treat the series of images as a set of sequential observations and merge them using Kalman filtering (Matthies, Kanade, and Szeliski 1989) or maximum likelihood inference (Cox 1994).

When fewer images are available, it becomes necessary to fall back on aggregation techniques, such as sliding windows or global optimization. With additional input images, however, the likelihood of occlusions increases. It is therefore prudent to adjust not only the best window locations using a shiftable window approach, as shown in Figure 12.20a, but also to optionally select a subset of neighboring frames to discount those images where the region of interest is occluded, as shown in Figure 12.20b (Kang, Szeliski, and Chai 2001). Fig-

⁷The four-dimensional generalization of the EPI is the *light field*, which we study in Section 14.3.

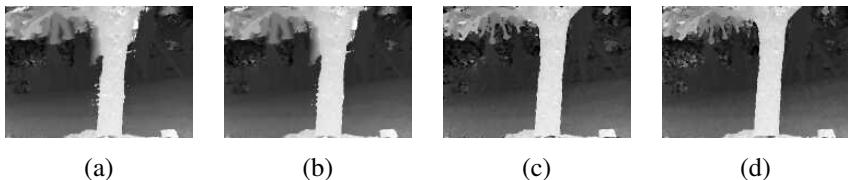


Figure 12.21 Local (5×5 window-based) matching results (Kang, Szeliski, and Chai 2001) © 2001 IEEE: (a) window that is not spatially perturbed (centered); (b) spatially perturbed window; (c) using the best five of 10 neighboring frames; (d) using the better half sequence. Notice how the results near the tree trunk are improved using temporal selection.

ure 12.19b shows how such spatio-temporal selection or shifting of windows corresponds to selecting the most likely un-occluded volumetric region in the epipolar plane image volume.

The results of applying these techniques to the multi-frame *flower garden* image sequence are shown in Figure 12.21, which compares the results of using regular (non-shifted) SSSD with spatially shifted windows and full spatio-temporal window selection. (The task of applying stereo to a rigid scene filmed with a moving camera is sometimes called *motion stereo*). Similar improvements from using spatio-temporal selection are reported by Kang and Szeliski (2004) and are evident even when local measurements are combined with global optimization.

While computing a depth map from multiple inputs outperforms pairwise stereo matching, even more dramatic improvements can be obtained by estimating multiple depth maps simultaneously (Szeliski 1999a; Kang and Szeliski 2004). The existence of multiple depth maps enables more accurate reasoning about occlusions, as regions that are occluded in one image may be visible (and matchable) in others. The multi-view reconstruction problem can be formulated as the simultaneous estimation of depth maps at key frames (Figure 9.11c) while maximizing not only photoconsistency and piecewise disparity smoothness, but also the consistency between disparity estimates at different frames. While Szeliski (1999a) and Kang and Szeliski (2004) use soft (penalty-based) constraints to encourage multiple disparity maps to be consistent, Kolmogorov and Zabih (2002) show how such consistency measures can be encoded as hard constraints, which guarantee that the multiple depth maps are not only similar but actually identical in overlapping regions. Additional algorithms that simultaneously estimate multiple disparity maps include those of Maitre, Shinagawa, and Do (2008) and Zhang, Jia *et al.* (2008) and the widely used COLMAP algorithm (Schönberger, Zheng *et al.* 2016), which uses view selection and geometric consistency between multiple depth maps to filter matches, as shown in Figure 12.26b.

The latest multi-view stereo algorithms use deep neural networks to compute matching

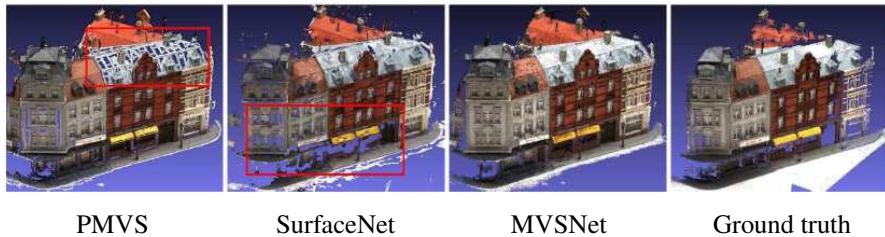


Figure 12.22 Depth maps computed using three different multi-view stereo algorithms shown as colored point clouds (Yao, Luo et al. 2018) © 2018 Springer. The red boxes indicate problem areas where MVSNet does better.

(cost) volumes and to fuse these into disparity maps. The DeepMVS system computes pairwise matching costs between a reference image and neighboring images and then fuses them together with max pooling, followed by a dense CRF refinement (Huang, Matzen et al. 2018). MVSNet computes the variance between all encoded images warped onto each sweep plane, uses a 3D U-Net to regularize the costs, and then a soft argmin and depth refinement network to produce good results on the DTU and Tanks and Temples datasets (Yao, Luo et al. 2018), as shown in Figure 12.22.

More recent variants on such networks include P-MVSNet (Luo, Guan et al. 2019), which uses a patch-wise matching confidence aggregator, and CasMVSNet (Gu, Fan et al. 2020) and CVP-MVSNet (Yang, Mao et al. 2020), both of which use coarse-to-fine pyramid processing. Four even more recent papers that all score well on the DTU, ETH3D, Tanks and Temples, and/or Blended MVS datasets are Vis-MVSNet (Zhang, Yao et al. 2020), D²HC-RMVSNet (Yan, Wei et al. 2020), DeepC-MVS (Kuhn, Sormann et al. 2020), and PatchmatchNet (Wang, Galliani et al. 2021). These algorithms use various combinations of visibility and occlusion reasoning, confidence or uncertainty maps, and geometric consistency checks, and efficient propagation schemes to achieve good results. As so many new multi-view stereo papers continue to get published, the ETH3D and Tanks and Temples leaderboards (Table 12.1) are good places to look for the latest results.

12.7.1 Scene flow

A closely related topic to multi-frame stereo estimation is *scene flow*, in which multiple cameras are used to capture a dynamic scene. The task is then to simultaneously recover the 3D shape of the object at every instant in time and to estimate the full 3D motion of every surface point between frames. Representative papers in this area include Vedula, Baker et al. (2005),

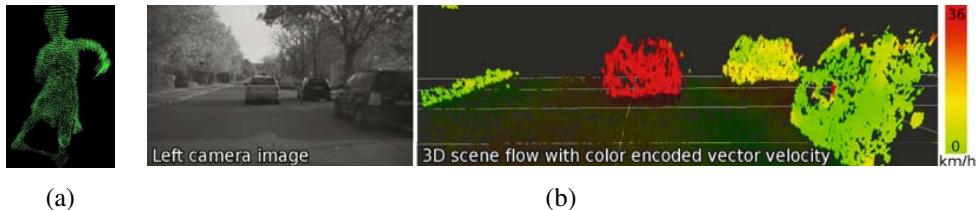


Figure 12.23 Three-dimensional scene flow: (a) computed from a multi-camera dome surrounding the dancer shown in Figure 12.2h–j (Vedula, Baker et al. 2005) © 2005 IEEE; (b) computed from stereo cameras mounted on a moving vehicle (Wedel, Rabe et al. 2008) © 2008 Springer.

Zhang and Kambhamettu (2003), Pons, Keriven, and Faugeras (2007), Huguet and Devernay (2007), Wedel, Rabe et al. (2008), and Rabe, Müller et al. (2010). Figure 12.23a shows an image of the 3D scene flow for the tango dancer shown in Figure 12.2h–j, while Figure 12.23b shows 3D scene flows captured from a moving vehicle for the purpose of obstacle avoidance. In addition to supporting mensuration and safety applications, scene flow can be used to support both spatial and temporal view interpolation (Section 14.5.4), as demonstrated by Vedula, Baker, and Kanade (2005).

The creation of the KITTI scene flow dataset (Geiger, Lenz, and Urtasun 2012) as well as increased interest in autonomous driving have accelerated research into scene flow algorithms (Janai, Güney et al. 2020, Chapter 12). One way to help regularize the problem is to adopt a piecewise planar representation (Vogel, Schindler, and Roth 2015). Another is to decompose the scene into rigid separately moving objects such as vehicles (Menze and Geiger 2015), using semantic segmentation (Behl, Hosseini Jafari et al. 2017), as well as to use other segmentation cues (Ilg, Saikia et al. 2018; Ma, Wang et al. 2019; Jiang, Sun et al. 2019). The more widespread availability of 3D sensors has enabled the extension of scene flow algorithms to use this modality as an additional input (Sun, Sudderth, and Pfister 2015; Behl, Paschalidou et al. 2019).

12.7.2 Volumetric and 3D surface reconstruction

The most challenging but also most useful variant of multi-view stereo reconstruction is the construction of globally consistent 3D models (Seitz, Curless et al. 2006). This topic has a long history in computer vision, starting with surface mesh reconstruction techniques such as the one developed by Fua and Leclerc (1995) (Figure 12.24a). A variety of approaches and representations have been used to solve this problem, including 3D voxels (Seitz and

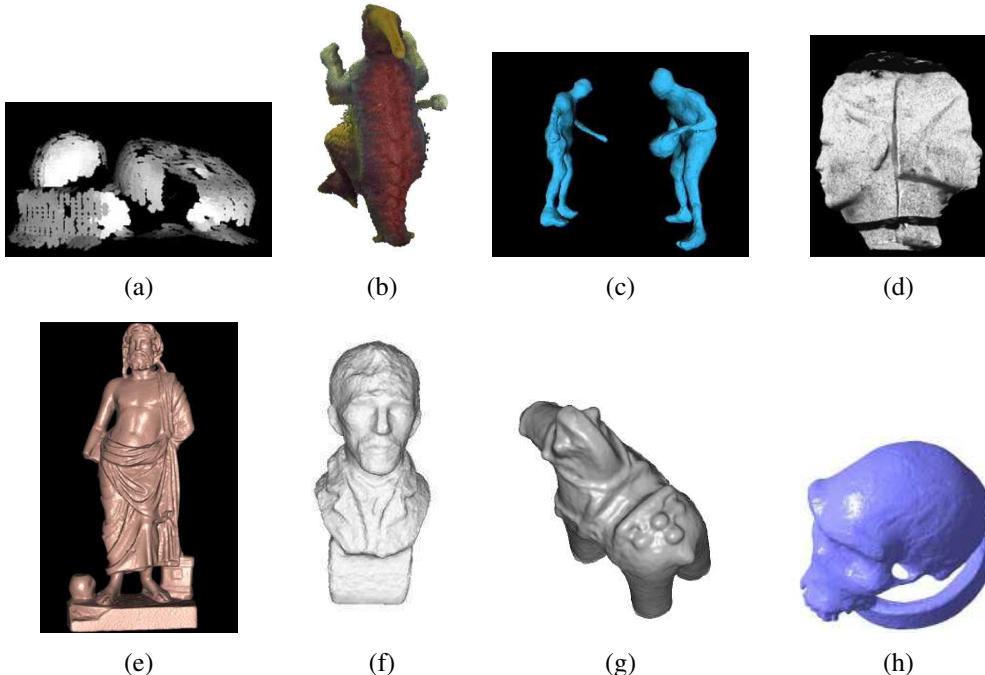


Figure 12.24 Multi-view stereo algorithms: (a) surface-based stereo (Fua and Leclerc 1995) © 1995 Springer; (b) voxel coloring (Seitz and Dyer 1999) © 1999 Springer; (c) depth map merging (Narayanan, Rander, and Kanade 1998) © 1998 IEEE; (d) level set evolution (Faugeras and Keriven 1998) © 1998 IEEE; (e) silhouette and stereo fusion (Hernández and Schmitt 2004) © 2004 Elsevier; (f) multi-view image matching (Pons, Keriven, and Faugeras 2005) © 2005 IEEE; (g) volumetric graph cut (Vogiatzis, Torr, and Cipolla 2005) © 2005 IEEE; (h) carved visual hulls (Furukawa and Ponce 2009) © 2009 Springer.

Dyer 1999; Szeliski and Golland 1999; De Bonet and Viola 1999; Kutulakos and Seitz 2000; Eisert, Steinbach, and Girod 2000; Slabaugh, Culbertson *et al.* 2004; Sinha and Pollefeys 2005; Vogiatzis, Hernández *et al.* 2007; Hiep, Keriven *et al.* 2009), level sets (Faugeras and Keriven 1998; Pons, Keriven, and Faugeras 2007), polygonal meshes (Fua and Leclerc 1995; Narayanan, Rander, and Kanade 1998; Hernández and Schmitt 2004; Furukawa and Ponce 2009), and multiple depth maps (Kolmogorov and Zabih 2002). Figure 12.24 shows representative examples of 3D object models reconstructed using some of these techniques.

To organize and compare all these techniques, Seitz, Curless *et al.* (2006) developed a six-point taxonomy that can help classify algorithms according to the *scene representation*,

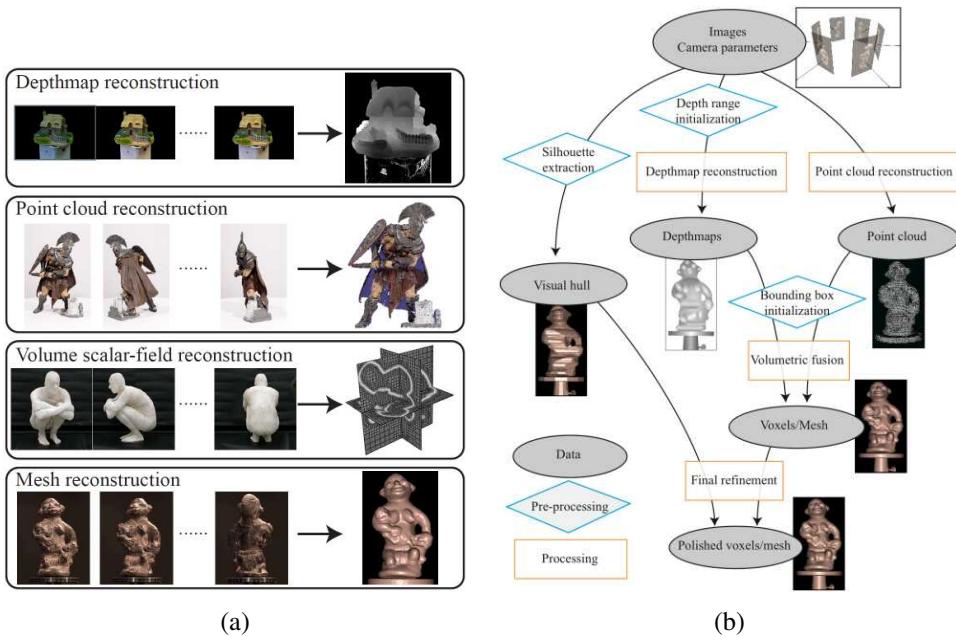


Figure 12.25 Multi-view stereo (a) scene representations and (b) processing pipelines, from Furukawa and Hernández (2015) © 2015 now publishers.

photoconsistency measure, visibility model, shape priors, reconstruction algorithm, and initialization requirements they use. Below, we summarize some of these choices and list a few representative papers. For more details, please consult the full survey paper (Seitz, Curless *et al.* 2006) as well as more recent surveys by Furukawa and Ponce (2010) and Janai, Güney *et al.* (2020, Chapter 10). The ETH3D and Tanks and Temples leaderboards list the most up-to-date results and pointers to recent papers.

Scene representation. According to the taxonomy proposed by Furukawa and Ponce (2010), multi-view stereo algorithms primarily use four scene representations, namely *depth maps*, *point clouds*, *volumetric fields*, and *3D meshes*, as shown in Figure 12.25a. These are often combined into a complete pipeline that includes camera pose estimation, per-image depth map or point cloud computation, volumetric fusion, and surface mesh refinement (Pollefeys, Nistér *et al.* 2008), as shown in Figure 12.25b.

We have already discussed multi-view depth map estimation earlier in this section. An example of a point cloud representation is the patch-based multi-view stereo (PMVS) algorithm developed by Furukawa and Ponce (2010), which starts with sparse 3D points reconstructed

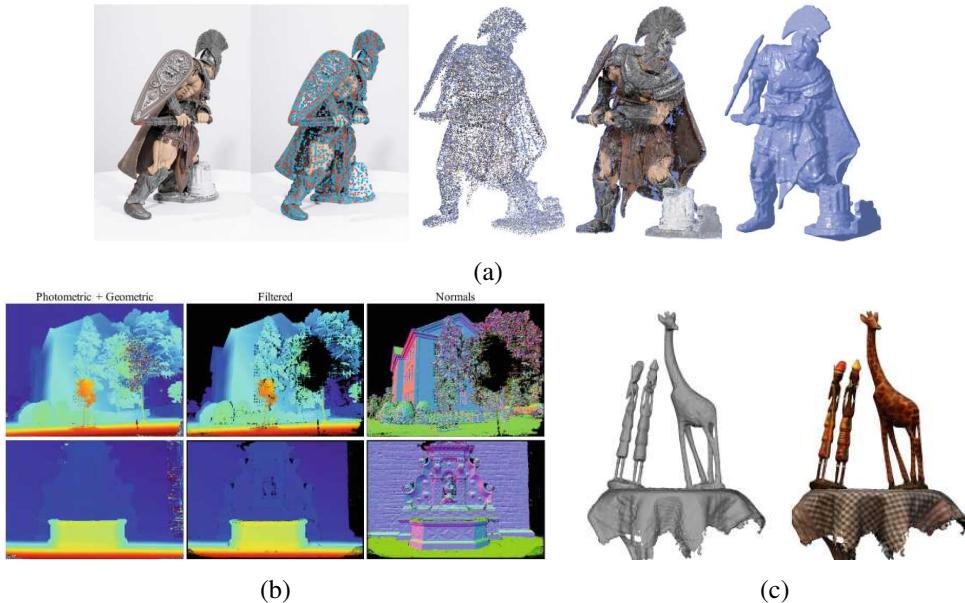


Figure 12.26 Point cloud reconstruction: (a) the PMVS pipeline, showing a sample input image, detected features, initial reconstructed patches, patches after expansion and filtering, and the final mesh model (Furukawa and Ponce 2010) © 2010 IEEE; (b) depth maps and surface normals from two stages of the COLMAP multi-view stereo pipeline (Schönberger, Zheng et al. 2016) © 2016 Springer; (c) thin structures recovered from gradients in a dense orbiting camera light field (Yücer, Kim et al. 2016) © 2016 IEEE.

using structure from motion, then optimizes and densifies local oriented patches or *surfels* (Szeliski and Tonnesen 1992; Section 13.4) while taking into account visibility constraints, as shown in Figure 12.26a. This representation can then be turned into a mesh for final optimization. Since then, improved techniques have been developed for view selection and filtering as well as normal estimation, as exemplified in the systems developed by Fuhrmann and Goesele (2014) and Schönberger, Zheng et al. (2016), the latter of which (shown in Figures 11.20b and 12.26b) provides the dense multi-view stereo component of the popular COLMAP open-source reconstruction system (Schönberger and Frahm 2016). When highly sampled video sequences are available, reconstructing point clouds from tracked edges may be more appropriate, as discussed in Section 12.2.1, Kim, Zimmer et al. (2013) and Yücer, Kim et al. (2016) and illustrated in Figure 12.26c.

One of the more popular 3D representations is a uniform grid of 3D voxels,⁸ which can be reconstructed using a variety of carving techniques (Seitz and Dyer 1999; Kutulakos and Seitz 2000) or optimization (Sinha and Pollefeys 2005; Vogiatzis, Hernández *et al.* 2007; Hiep, Keriven *et al.* 2009; Jancosek and Pajdla 2011; Vu, Labatut *et al.* 2012), some of which are illustrated in Figure 12.24. Level set techniques (Section 7.3.2) also operate on a uniform grid but, instead of representing a binary occupancy map, they represent the signed distance to the surface (Faugeras and Keriven 1998; Pons, Keriven, and Faugeras 2007), which can encode a finer level of detail and also be used to merge multiple point clouds or range data scans, as discussed extensively in Section 13.2.1. Instead of using a uniformly sampled volume, which works best for compact 3D objects, it is also possible to create a view frustum corresponding to one of the input images and to sample the z dimension as inverse depths, i.e., uniform disparities for a set of co-planar cameras (Figure 14.7). This kind of representation is called a *stack of acetates* in Szeliski and Golland (1999) and *multiplane images* in Zhou, Tucker *et al.* (2018).

Polygonal meshes are another popular representation (Fua and Leclerc 1995; Narayanan, Rander, and Kanade 1998; Isidoro and Sclaroff 2003; Hernández and Schmitt 2004; Furukawa and Ponce 2009; Hiep, Keriven *et al.* 2009). Meshes are the standard representation used in computer graphics and also readily support the computation of visibility and occlusions. Finally, as we discussed in the previous section, multiple depth maps can also be used (Szeliski 1999a; Kolmogorov and Zabih 2002; Kang and Szeliski 2004). Many algorithms also use more than a single representation, e.g., they may start by computing multiple depth maps and then merge them into a 3D object model (Narayanan, Rander, and Kanade 1998; Furukawa and Ponce 2009; Goesele, Curless, and Seitz 2006; Goesele, Snavely *et al.* 2007; Pollefeys, Nistér *et al.* 2008; Furukawa, Curless *et al.* 2010; Furukawa and Ponce 2010; Vu, Labatut *et al.* 2012; Schönberger, Zheng *et al.* 2016), as illustrated in Figure 12.25b.

An example of a recent system that combines several representations into a scalable distributed approach that can handle datasets with hundreds of high-resolution images is the LTVRE multi-view stereo system by Kuhn, Hirschmüller *et al.* (2017). The system starts from pairwise disparity maps computed with SGM (Hirschmüller 2008). These depth estimates are fused with a probabilistic multi-scale approach using a learned stereo error model, using an octree to handle variable resolution, followed by filtering of conflicting points based on visibility constraints, and finally triangulation. Figure 12.27 shows an illustration of the processing pipeline.

⁸For outdoor scenes that go to infinity, an inverted gridding of space may be preferable (Slabaugh, Culbertson *et al.* 2004; Zhang, Riegler *et al.* 2020).

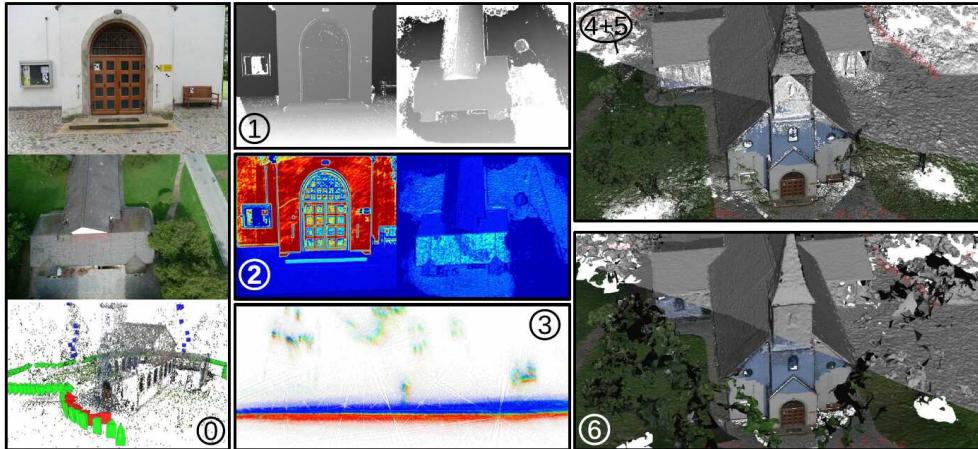


Figure 12.27 3D reconstruction pipeline from Kuhn, Hirschmüller et al. (2017) © 2017 Springer: (0) structure from motion; (1) stereo matching using semi-global matching; (2) depth quality estimation; (3) probabilistic space occupancy; (4+5) probabilistic point optimization and outlier filtering; (6) triangulation. The images in (4+5) and (6) are half texture-mapped and half flat shaded to show more surface detail.

Photoconsistency measure. As we discussed in (Section 12.3.1), a variety of similarity measures can be used to compare pixel values in different images, including measures that try to discount illumination effects or be less sensitive to outliers. In multi-view stereo, algorithms have a choice of computing these measures directly on the surface of the model, i.e., in *scene space*, or projecting pixel values from one image (or from a textured model) back into another image, i.e., in *image space*. (The latter corresponds more closely to a Bayesian approach, because input images are noisy measurements of the colored 3D model.) The geometry of the object, i.e., its distance to each camera and its local surface normal, when available, can be used to adjust the matching windows used in the computation to account for foreshortening and scale change (Goesele, Snavely et al. 2007).

Visibility model. A big advantage that multi-view stereo algorithms have over single-depth-map approaches is their ability to reason in a principled manner about visibility and occlusions. Techniques that use the current state of the 3D model to predict which surface pixels are visible in each image (Kutulakos and Seitz 2000; Faugeras and Keriven 1998; Vogiatzis, Hernández et al. 2007; Hiep, Keriven et al. 2009; Furukawa and Ponce 2010; Schönberger, Zheng et al. 2016) are classified as using *geometric visibility models* in the taxonomy of Seitz,

Curless *et al.* (2006). Techniques that select a neighboring subset of image to match are called *quasi-geometric* (Narayanan, Rander, and Kanade 1998; Kang and Szeliski 2004; Hernández and Schmitt 2004), while techniques that use traditional robust similarity measures are called *outlier-based*. While full geometric reasoning is the most principled and accurate approach, it can be very slow to evaluate and depends on the evolving quality of the current surface estimate to predict visibility, which can be a bit of a chicken-and-egg problem, unless conservative assumptions are used, as they are by Kutulakos and Seitz (2000).

Shape priors. Because stereo matching is often underconstrained, especially in texture-less regions, most matching algorithms adopt (either explicitly or implicitly) some form of prior model for the expected shape. Many of the techniques that rely on optimization use a 3D smoothness or area-based photoconsistency constraint, which, because of the natural tendency of smooth surfaces to shrink inwards, often results in a *minimal surface* prior (Faugeras and Keriven 1998; Sinha and Pollefeys 2005; Vogiatzis, Hernández *et al.* 2007). Approaches that carve away the volume of space often stop once a photoconsistent solution is found (Seitz and Dyer 1999; Kutulakos and Seitz 2000), which corresponds to a *maximal surface* bias, i.e., these techniques tend to over-estimate the true shape. Finally, multiple depth map approaches often adopt traditional *image-based* smoothness (regularization) constraints.

Higher-level shape priors can also be used, such as Manhattan world assumptions that assume most surfaces of interest are axis-aligned (Furukawa, Curless *et al.* 2009a,b) or at related orientations such as slanted roofs (Sinha, Steedly, and Szeliski 2009; Osman Ulusoy, Black, and Geiger 2017). These kinds of *architectural priors* are discussed in more detail in Section 13.6.1. It is also possible to use 2D semantic segmentation in images, e.g., into wall, ground, and foliage classes, to apply different kinds of regularization and surface normal priors in different regions of the model (Häne, Zach *et al.* 2013).

Reconstruction algorithm. The details of how the actual reconstruction algorithm proceeds is where the largest variety—and greatest innovation—in multi-view stereo algorithms can be found.

Some approaches use global optimization defined over a three-dimensional photoconsistency volume to recover a complete surface. Approaches based on graph cuts use polynomial complexity binary segmentation algorithms to recover the object model defined on the voxel grid (Sinha and Pollefeys 2005; Vogiatzis, Hernández *et al.* 2007; Hiep, Keriven *et al.* 2009; Jancosek and Pajdla 2011; Vu, Labatut *et al.* 2012). Level set approaches use a continuous surface evolution to find a good minimum in the configuration space of potential surfaces and therefore require a reasonably good initialization (Faugeras and Keriven 1998; Pons, Keriven,

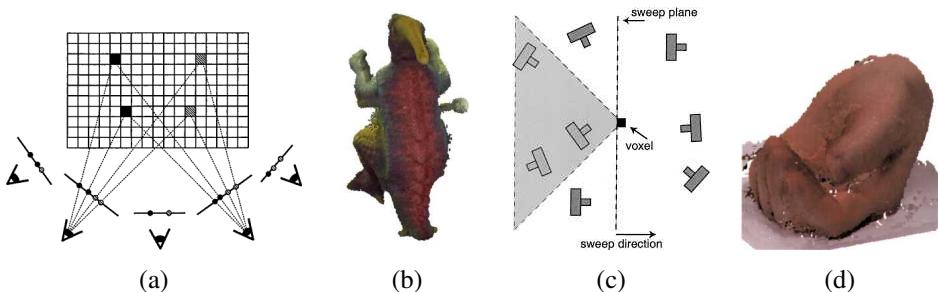


Figure 12.28 *Voxel coloring (Seitz and Dyer 1999) © 1999 Springer and space carving (Kutulakos and Seitz 2000) © 2000 Springer.* (a–b): voxel coloring sweeps a plane through the scene in a front-to-back manner with respect to the cameras. (c–d): space carving uses multiple sweep directions to deal with more general camera configurations.

and Faugeras 2007). For the photoconsistency volume to be meaningful, matching costs need to be computed in some robust fashion, e.g., using sets of limited views or by aggregating multiple depth maps.

An alternative approach to global optimization is to sweep through the 3D volume while computing both photoconsistency and visibility simultaneously. The *voxel coloring* algorithm of Seitz and Dyer (1999) performs a front-to-back plane sweep. On every plane, any voxels that are sufficiently photoconsistent are labeled as part of the object. The corresponding pixels in the source images can then be “erased”, as they are already accounted for, and therefore do not contribute to further photoconsistency computations. (A similar approach, albeit without the front-to-back sweep order, is used by Szeliski and Golland (1999).) The resulting 3D volume, under noise- and resampling-free conditions, is guaranteed to produce both a photoconsistent 3D model and to enclose whatever true 3D object model generated the images (Figure 12.28a–b).

Unfortunately, voxel coloring is only guaranteed to work if all of the cameras lie on the same side of the sweep planes, which is not possible in general ring configurations of cameras. Kutulakos and Seitz (2000) generalize voxel coloring to *space carving*, where subsets of cameras that satisfy the voxel coloring constraint are iteratively selected and the 3D voxel grid is alternately carved away along different axes (Figure 12.28c–d).

Another popular approach to multi-view stereo is to first independently compute multiple depth maps and then merge these partial maps into a complete 3D model. Approaches to depth map merging, which are discussed in more detail in Section 13.2.1, include signed distance functions (Curless and Levoy 1996), used by Goesele, Curless, and Seitz (2006), and Poisson surface reconstruction (Kazhdan, Bolitho, and Hoppe 2006), used by Goesele,

Snavely *et al.* (2007).

Initialization requirements. One final element discussed by Seitz, Curless *et al.* (2006) is the varying degrees of initialization required by different algorithms. Because some algorithms refine or evolve a rough 3D model, they require a reasonably accurate (or overcomplete) initial model, which can often be obtained by reconstructing a volume from object silhouettes, as discussed in Section 12.7.3. However, if the algorithm performs a global optimization (Kolev, Klodt *et al.* 2009; Kolev and Cremers 2009), this dependence on initialization is not an issue.

Empirical evaluation. The widespread adoption of datasets and benchmarks has led to the rapid advances in multi-view reconstruction over the last two decades. Table 12.1 lists some of the most widely used and influential ones, with sample images and/or results shown in Figures 12.1, 12.22, and 12.26. Pointers to additional datasets can be found in Mayer, Ilg *et al.* (2018), Janai, Güney *et al.* (2020), Laga, Jospin *et al.* (2020), and Poggi, Tosi *et al.* (2021). Pointers to the most recent algorithms can usually be found on the leaderboards of the ETH3D and Tanks and Temples benchmarks.

12.7.3 Shape from silhouettes

In many situations, performing a foreground–background segmentation of the object of interest is a good way to initialize or fit a 3D model (Grauman, Shakhnarovich, and Darrell 2003; Vlasic, Baran *et al.* 2008) or to impose a convex set of constraints on multi-view stereo (Kolev and Cremers 2008). Over the years, a number of techniques have been developed to reconstruct a 3D volumetric model from the intersection of the binary silhouettes projected into 3D. The resulting model is called a *visual hull* (or sometimes a *line hull*), analogous with the convex hull of a set of points, because the volume is maximal with respect to the visual silhouettes and surface elements are tangent to the viewing rays (lines) along the silhouette boundaries (Laurentini 1994). It is also possible to carve away a more accurate reconstruction using multi-view stereo (Sinha and Pollefeys 2005) or by analyzing cast shadows (Savarese, Andreetto *et al.* 2007).

Some techniques first approximate each silhouette with a polygonal representation and then intersect the resulting faceted conical regions in three-space to produce polyhedral models (Baumgart 1974; Martin and Aggarwal 1983; Matusik, Buehler, and McMillan 2001), which can later be refined using triangular splines (Sullivan and Ponce 1998). Other approaches use voxel-based representations, usually encoded as octrees (Samet 1989), because of the resulting space–time efficiency. Figures 12.29a–b show an example of a 3D octree

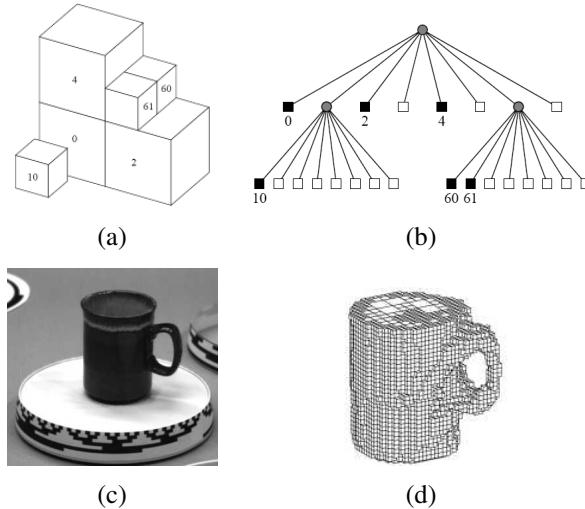


Figure 12.29 Volumetric octree reconstruction from binary silhouettes (Szeliski 1993) © 1993 Elsevier: (a) octree representation and its corresponding (b) tree structure; (c) input image of an object on a turntable; (d) computed 3D volumetric octree model.

model and its associated colored tree, where black nodes are interior to the model, white nodes are exterior, and gray nodes are of mixed occupancy. Examples of octree-based reconstruction approaches include Potmesil (1987), Noborio, Fukada, and Arimoto (1988), Srivasan, Liang, and Hackwood (1990), and Szeliski (1993).

The approach of Szeliski (1993) first converts each binary silhouette into a one-sided variant of a distance map, where each pixel in the map indicates the largest square that is completely inside (or outside) the silhouette. This makes it fast to project an octree cell into the silhouette to confirm whether it is completely inside or outside the object, so that it can be colored black, or white, or left as gray (mixed) for further refinement on a smaller grid. The octree construction algorithm proceeds in a coarse-to-fine manner, first building an octree at a relatively coarse resolution, and then refining it by revisiting and subdividing all the input images for the gray (mixed) cells whose occupancy has not yet been determined. Figure 12.29d shows the resulting octree model computed from a coffee cup rotating on a turntable.

More recent work on visual hull computation borrows ideas from image-based rendering, and is hence called an *image-based visual hull* (Matusik, Buehler *et al.* 2000). Instead of precomputing a global 3D model, an image-based visual hull is recomputed for each new viewpoint, by successively intersecting viewing ray segments with the binary silhouettes in

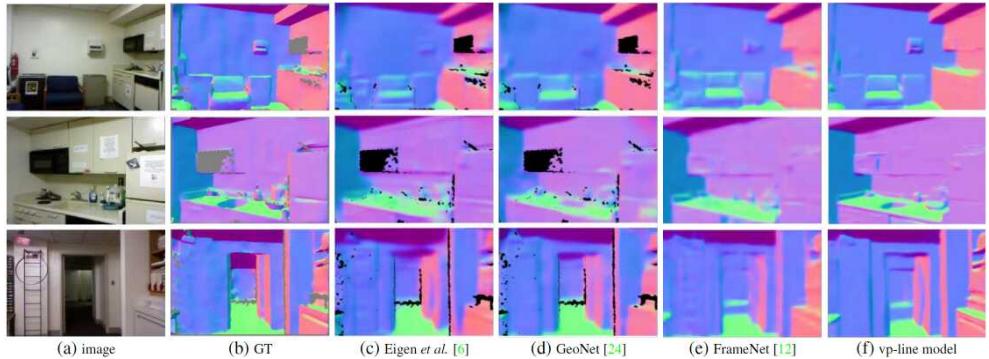


Figure 12.30 *Monocular depth inference (shown as color-coded normal maps) from images in the NYU Depth Dataset V2 (Wang, Geraghty et al. 2020) © 2020 IEEE.*

each image. This not only leads to a fast computation algorithm but also enables fast texturing of the recovered model with color values from the input images. This approach can also be combined with high-quality deformable templates to capture and re-animate whole body motion (Vlasic, Baran *et al.* 2008).

While the methods described above start with a binary foreground/background silhouette image, it is also possible to extract silhouette curves, usually to sub-pixel precision, and to reconstruct partial surface meshes from tracking these, as discussed in Section 12.2.1. Such silhouette curves can also be combined with regular image edges to construct complete surface models (Yücer, Kim *et al.* 2016), such as the ones shown in Figure 12.26c.

12.8 Monocular depth estimation

The ability to infer (or hallucinate?) depth maps from single images opens up all kinds of creative possibilities, such as displaying them in 3D (Figure 6.41 and Kopf, Matzen *et al.* 2020), creating soft focus effects (Section 10.3.2), and potentially to aid scene understanding. It can also be used in robotics applications such as autonomous navigation (Figure 12.31), although most (autonomous and regular) vehicles have more than one camera or range sensors, if equipped with computer vision.

We already saw in Section 6.4.4 how the automatic photo pop-up system can use image segmentation and classification to create “cardboard cut-out” versions of a photo (Hoiem, Efros, and Hebert 2005a; Saxena, Sun, and Ng 2009). More recent systems to infer depth from single images use deep neural networks. These are described in two recent surveys (Poggi, Tosi *et al.* 2021, Section 7; Zhao, Sun *et al.* 2020), which discuss 20 and over 50

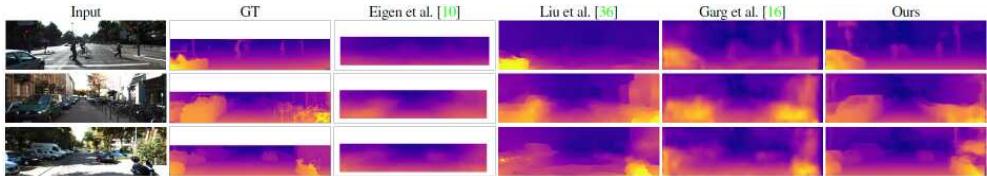


Figure 12.31 *Monocular depth map estimates from images in the KITTI dataset (Godard, Mac Aodha, and Brostow 2017) © 2017 IEEE.*

related techniques, respectively, and benchmark them on the KITTI dataset (Geiger, Lenz, and Urtasun 2012) shown in Figure 12.31.

One of the first papers to use a neural network to compute a depth map was the system developed by Eigen, Puhrsch, and Fergus (2014), which was subsequently extended to also infer surface normals and semantic labels (Eigen and Fergus 2015). These systems were trained and tested on the NYU Depth Dataset V2 (Silberman, Hoiem *et al.* 2012), shown in Figure 12.30, and the KITTI dataset. Most of the subsequent work in this area trains and tests on these two fairly restricted datasets (indoor apartments or outdoor street scenes), although authors sometimes use Make3D (Saxena, Sun, and Ng 2009) or Cityscapes (Cordts, Omran *et al.* 2016), which are both outdoor street scenes, or ScanNet (Dai, Chang *et al.* 2017), which has indoor scenes. The danger in training and testing on such “closed world” datasets is that the network can learn shortcuts, such as inferring depth based on the location along the ground plane or failing to “pop up” objects that are not in commonly occurring classes (van Dijk and de Croon 2019).

Early systems were trained on the depth images that came with datasets such as NYU Depth, KITTI, and ScanNet, where it turns out that adding soft constraints such as coplanarity can improve performance (Wang, Shen *et al.* 2016; Yin, Liu *et al.* 2019). Later, “unsupervised” techniques were introduced that use photometric consistency between warped stereo pairs of images (Godard, Mac Aodha, and Brostow 2017; Xian, Shen *et al.* 2018) or image pairs in video sequences (Zhou, Brown *et al.* 2017). It is also possible to train on 3D reconstructions of famous landmarks (Li and Snavely 2018), image sets containing people posing in a “Mannequin Challenge” (Li, Dekel *et al.* 2019), or to take more diverse “images in the wild” and have them labeled with relative depths (Chen, Fu *et al.* 2016).

A recent paper that federates several such datasets is the MiDaS system developed by Ranftl, Lasinger *et al.* (2020), who not only use a number of existing “in the wild” datasets to train a network based on Xian, Shen *et al.* (2018), but also use thousands of stereo image pairs from over a dozen 3D movies as additional training, validation, and test data. In their paper, they not only show that their system produces superior results to previous approaches



Figure 12.32 Monocular depth map estimates and novel views from images in COCO dataset (Ranftl, Lasinger et al. 2020) © 2020 IEEE.

(Figure 12.32), but also argue that their *zero-shot cross-dataset transfer* protocol, i.e., testing on data sets separate from training sets, rather than using random train and test subsets, produces a system that works better on real-world images and avoids unintended dataset bias (Torralba and Efros 2011).

An alternative to inferring depth maps from single images is to infer complete closed 3D shapes, using either volumetric (Choy, Xu et al. 2016; Girdhar, Fouhey et al. 2016; Tulsiani, Gupta et al. 2018) or mesh-based (Gkioxari, Malik, and Johnson 2019) representations (Han, Laga, and Bennamoun 2021). Instead of applying deep networks to just a single color image, it is also possible to augment such networks with additional cues and representations, such as oriented lines and planes (Wang, Geraghty et al. 2020), which serve as higher-level *shape priors* (Sections 12.7.2 and 13.6.1). Neural rendering can also be used to create novel views (Tucker and Snavely 2020; Wiles, Gkioxari et al. 2020; Figure 14.22d), and to make the monocular depth predictions consistent over time (Luo, Huang et al. 2020; Teed and Deng 2020a; Kopf, Rong, and Huang 2021). An example of a consumer application of monocular depth inference is One Shot 3D Photography (Kopf, Matzen et al. 2020), where the system, implemented on a mobile phone using a compact and efficient DNN, first infers a depth map, then converts this to multiple layers, inpaints the background, creates a mesh and texture atlas, and then provides real-time interactive viewing on the phone, as shown in Figure 14.10c.

12.9 Additional reading

The field of stereo correspondence and depth estimation is one of the oldest and most widely studied topics in computer vision. A number of good surveys have been written over the years (Marr and Poggio 1976; Barnard and Fischler 1982; Dhond and Aggarwal 1989; Scharstein and Szeliski 2002; Brown, Burschka, and Hager 2003; Seitz, Curless *et al.* 2006; Furukawa and Hernández 2015; Janai, Güney *et al.* 2020; Laga, Jospin *et al.* 2020; Poggi, Tosi *et al.* 2021) and they can serve as good guides to this extensive literature.

Because of computational limitations and the desire to find appearance-invariant correspondences, early algorithms often focused on finding *sparse correspondences* (Hannah 1974; Marr and Poggio 1979; Mayhew and Frisby 1980; Ohta and Kanade 1985; Bolles, Baker, and Marimont 1987; Matthies, Kanade, and Szeliski 1989).

The topic of computing epipolar geometry and pre-rectifying images is covered in Sections 11.3 and 12.1 and is also treated in textbooks on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004) and articles specifically on this topic (Torr and Murray 1997; Zhang 1998a,b). The concepts of the *disparity space* and *disparity space image* are often associated with the seminal work by Marr (1982) and the papers of Yang, Yuille, and Lu (1993) and Intille and Bobick (1994). The plane sweep algorithm was first popularized by Collins (1996) and then generalized to a full arbitrary projective setting by Szeliski and Golland (1999) and Saito and Kanade (1999). Plane sweeps can also be formulated using cylindrical surfaces (Ishiguro, Yamamoto, and Tsuji 1992; Kang and Szeliski 1997; Shum and Szeliski 1999; Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007) or even more general topologies (Seitz 2001).

Once the topology for the cost volume or DSI has been set up, we need to compute the actual photoconsistency measures for each pixel and potential depth. A wide range of such measures have been proposed, as discussed in Section 12.3.1. Some of these are compared in recent surveys and evaluations of matching costs (Scharstein and Szeliski 2002; Hirschmüller and Scharstein 2009).

To compute an actual depth map from these costs, some form of optimization or selection criterion must be used. The simplest of these are sliding windows of various kinds, which are discussed in Section 12.4 and surveyed by Gong, Yang *et al.* (2007) and Tombari, Mattoccia *et al.* (2008). Global optimization frameworks are often used to compute the best disparity field, as described in Section 12.5. These techniques include dynamic programming and truly global optimization algorithms, such as graph cuts and loopy belief propagation. More recently, deep neural networks have become popular for computing correspondence and disparity maps, as discussed in Section 12.6 and surveyed in Laga, Jospin *et al.* (2020) and

Poggi, Tosi *et al.* (2021). A good place to find pointers to the latest results in this field is the list of benchmarks in Table 12.1.

Algorithms for multi-view stereo typically fall into two categories (Furukawa and Hernández 2015). The first include algorithms that compute traditional depth maps using several images for computing photoconsistency measures (Okutomi and Kanade 1993; Kang, Webb *et al.* 1995; Szeliski and Golland 1999; Vaish, Szeliski *et al.* 2006; Gallup, Frahm *et al.* 2008; Huang, Matzen *et al.* 2018; Yao, Luo *et al.* 2018). Some of these techniques compute multiple depth maps and use additional constraints to encourage the different depth maps to be consistent (Szeliski 1999a; Kolmogorov and Zabih 2002; Kang and Szeliski 2004; Maitre, Shinagawa, and Do 2008; Zhang, Jia *et al.* 2008; Yan, Wei *et al.* 2020; Zhang, Yao *et al.* 2020).

The second category consists of papers that compute true 3D volumetric or surface-based object models. Again, because of the large number of papers published on this topic, rather than citing them here, we refer you to the material in Section 12.7.2, the surveys by Seitz, Curless *et al.* (2006), Furukawa and Hernández (2015), and Janai, Güney *et al.* (2020), and the online evaluation websites listed in Table 12.1.

The topic of monocular depth inference is currently very active. Good places to start, in addition to Section 12.8, are the recent surveys by Poggi, Tosi *et al.* (2021, Section 7) and Zhao, Sun *et al.* (2020).

12.10 Exercises

Ex 12.1: Stereo pair rectification. Implement the following simple algorithm (Section 12.1.1):

1. Rotate both cameras so that they are looking perpendicular to the line joining the two camera centers \mathbf{c}_0 and \mathbf{c}_1 . The smallest rotation can be computed from the cross product between the original and desired optical axes.
2. Twist the optical axes so that the horizontal axis of each camera looks in the direction of the other camera. (Again, the cross product between the current x -axis after the first rotation and the line joining the cameras gives the rotation.)
3. If needed, scale up the smaller (less detailed) image so that it has the same resolution (and hence line-to-line correspondence) as the other image.

Now compare your results to the algorithm proposed by Loop and Zhang (1999). Can you think of situations where their approach may be preferable?

Ex 12.2: Rigid direct alignment. Modify your spline-based or optical flow motion estimator from Exercise 9.4 to use epipolar geometry, i.e. to only estimate disparity.

(Optional) Extend your algorithm to simultaneously estimate the epipolar geometry (without first using point correspondences) by estimating a base homography corresponding to a reference plane for the dominant motion and then an epipole for the residual parallax (motion).

Ex 12.3: Shape from profiles. Reconstruct a surface model from a series of edge images (Section 12.2.1).

1. Extract edges and link them (Exercises 7.7–7.8).
2. Based on previously computed epipolar geometry, match up edges in triplets (or longer sets) of images.
3. Reconstruct the 3D locations of the curves using osculating circles (Szeliski and Weiss 1998).
4. Render the resulting 3D surface model as a sparse mesh, i.e., drawing the reconstructed 3D profile curves and links between 3D points in neighboring images with similar osculating circles.

Ex 12.4: Plane sweep. Implement a plane sweep algorithm (Section 12.1.2).

If the images are already pre-rectified, this consists simply of shifting images relative to each other and comparing pixels. If the images are not pre-rectified, compute the homography that resamples the target image into the reference image's coordinate system for each plane.

Evaluate a subset of the following similarity measures (Section 12.3.1) and compare their performance by visualizing the disparity space image (DSI), which should be dark for pixels at correct depths:

- squared difference (SD);
- absolute difference (AD);
- truncated or robust measures;
- gradient differences;
- rank or census transform (the latter usually performs better);
- mutual information from a precomputed joint density function.

Consider using the Birchfield and Tomasi (1998) technique of comparing ranges between neighboring pixels (different shifted or warped images). Also, try pre-compensating images for bias or gain variations using one or more of the techniques discussed in Section 12.3.1.

Ex 12.5: Aggregation and window-based stereo. Implement one or more of the matching cost aggregation strategies described in Section 12.4:

- convolution with a box or Gaussian kernel;
- shifting window locations by applying a min filter (Scharstein and Szeliski 2002);
- picking a window that maximizes some match-reliability metric (Veksler 2001, 2003);
- weighting pixels by their similarity to the central pixel (Yoon and Kweon 2006).

Once you have aggregated the costs in the DSI, pick the winner at each pixel (winner-take-all), and then optionally perform one or more of the following post-processing steps:

1. compute matches both ways and pick only the reliable matches (draw the others in another color);
2. tag matches that are unsure (whose confidence is too low);
3. fill in the matches that are unsure from neighboring values;
4. refine your matches to sub-pixel disparity by either fitting a parabola to the DSI values around the winner or by using an iteration of Lukas–Kanade.

Ex 12.6: Optimization-based stereo. Compute the disparity space image (DSI) volume using one of the techniques you implemented in Exercise 12.4 and then implement one (or more) of the global optimization techniques described in Section 12.5 to compute the depth map. Potential choices include:

- dynamic programming or scanline optimization (relatively easy);
- semi-global optimization (Hirschmüller 2008), which is a simple extension of scanline optimization and performs well;
- graph cuts using alpha expansions (Boykov, Veksler, and Zabih 2001), for which you will need to find a max-flow or min-cut algorithm (<https://vision.middlebury.edu/stereo>);
- loopy belief propagation (Freeman, Pasztor, and Carmichael 2000);
- deep neural networks, as described in Section 12.6.

Evaluate your algorithm by running it on the Middlebury stereo datasets.

How well does your algorithm do against local aggregation (Yoon and Kweon 2006)? Can you think of some extensions or modifications to make it even better?

Ex 12.7: View interpolation, revisited. Compute a dense depth map using one of the techniques you developed above and use it (or, better yet, a depth map for each source image) to generate smooth in-between views from a stereo dataset.

Compare your results against using the ground truth depth data (if available).

What kinds of artifacts do you see? Can you think of ways to reduce them?

More details on implementing such algorithms can be found in Section 14.1 and Exercises 14.1–14.4.

Ex 12.8: Multi-frame stereo. Extend one of your previous techniques to use multiple input frames (Section 12.7) and try to improve the results you obtained with just two views.

If helpful, try using temporal selection (Kang and Szeliski 2004) to deal with the increased number of occlusions in multi-frame datasets.

You can also try to simultaneously estimate multiple depth maps and make them consistent (Kolmogorov and Zabih 2002; Kang and Szeliski 2004).

Or just use one of the latest DNN-based multi-view stereo algorithms.

Test your algorithms out on some standard multi-view datasets.

Ex 12.9: Volumetric stereo. Implement voxel coloring (Seitz and Dyer 1999) as a simple extension to the plane sweep algorithm you implemented in Exercise 12.4.

1. Instead of computing the complete DSI all at once, evaluate each plane one at a time from front to back.
2. Tag every voxel whose photoconsistency is below a certain threshold as being part of the object and remember its average (or robust) color (Seitz and Dyer 1999; Eisert, Steinbach, and Girod 2000; Kutulakos 2000; Slabaugh, Culbertson *et al.* 2004).
3. Erase the input pixels corresponding to tagged voxels in the input images, e.g., by setting their alpha value to 0 (or to some reduced number, depending on occupancy).
4. As you evaluate the next plane, use the source image alpha values to modify your photoconsistency score, e.g., only consider pixels that have full alpha or weight pixels by their alpha values.
5. If the cameras are not all on the same side of your plane sweeps, use space carving (Kutulakos and Seitz 2000) to cycle through different subsets of source images while carving away the volume from different directions.

Ex 12.10: Depth map merging. Use the technique you developed for multi-frame stereo in Exercise 12.8 or a different technique, such as the one described by Goesele, Snavely *et al.* (2007), to compute a depth map for every input image.

Merge these depth maps into a coherent 3D model, e.g., using Poisson surface reconstruction (Kazhdan, Bolitho, and Hoppe 2006).

Ex 12.11: Monocular depth estimation. Test out of the recent monocular depth inference algorithms on your own images. Can you create a “3D photo” effect where wiggling your camera or moving your mouse makes the photo move in 3D. Tabulate the failure cases of the depth inference and conjecture some possible reasons and/or avenues for improvement.

Chapter 13

3D reconstruction

13.1	Shape from X	809
13.1.1	Shape from shading and photometric stereo	809
13.1.2	Shape from texture	814
13.1.3	Shape from focus	814
13.2	3D scanning	816
13.2.1	Range data merging	820
13.2.2	<i>Application:</i> Digital heritage	824
13.3	Surface representations	825
13.3.1	Surface interpolation	826
13.3.2	Surface simplification	827
13.3.3	Geometry images	828
13.4	Point-based representations	829
13.5	Volumetric representations	830
13.5.1	Implicit surfaces and level sets	831
13.6	Model-based reconstruction	833
13.6.1	Architecture	833
13.6.2	Facial modeling and tracking	838
13.6.3	<i>Application:</i> Facial animation	839
13.6.4	Human body modeling and tracking	843
13.7	Recovering texture maps and albedos	850
13.7.1	Estimating BRDFs	852
13.7.2	<i>Application:</i> 3D model capture	854
13.8	Additional reading	855

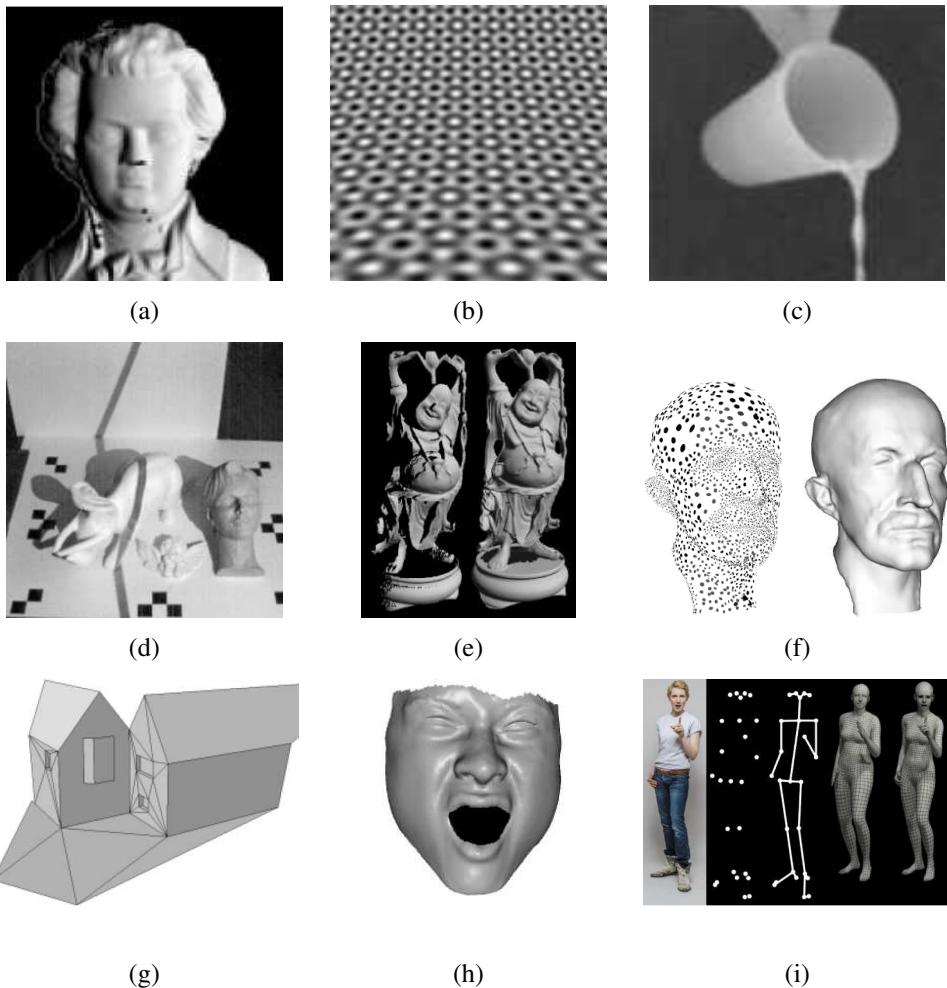


Figure 13.1 3D shape acquisition and modeling techniques: (a) shaded image (Zhang, Tsai et al. 1999) © 1999 IEEE; (b) texture gradient (Gårding 1992) © 1992 Springer; (c) real-time depth from focus (Nayar, Watanabe, and Noguchi 1996) © 1996 IEEE; (d) scanning a scene with a stick shadow (Bouguet and Perona 1999) © 1999 Springer; (e) merging range maps into a 3D model (Curless and Levoy 1996) © 1996 ACM; (f) point-based surface modeling (Pauly, Keiser et al. 2003) © 2003 ACM; (g) automated modeling of a 3D building using lines and planes (Werner and Zisserman 2002) © 2002 Springer; (h) 3D face model from spacetime stereo (Zhang, Snavely et al. 2004) © 2004 ACM; (i) whole body, expression, and gesture fitting from a single image (Pavlakos, Choutas et al. 2019) © 2019 IEEE.

13.9 Exercises	857
--------------------------	-----

As we saw in the previous chapter, many stereo matching techniques have been developed to reconstruct high-quality 3D models from two or more images. However, stereo is just one of the many potential cues that can be used to infer shape from images. In this chapter, we investigate a number of such techniques, which include not only visual cues such as shading and focus, but also techniques for merging multiple range or depth images into 3D models, as well as techniques for reconstructing specialized models, such as heads, bodies, or architecture.

Among the various cues that can be used to infer shape, the shading on a surface (Figure 13.1a) can provide a lot of information about local surface orientations and hence overall surface shape (Section 13.1.1). This approach becomes even more powerful when lights shining from different directions can be turned on and off separately (*photometric stereo*). Texture gradients (Figure 13.1b), i.e., the foreshortening of regular patterns as the surface slants or bends away from the camera, can provide similar cues on local surface orientation (Section 13.1.2). Focus is another powerful cue to scene depth, especially when two or more images with different focus settings are used (Section 13.1.3).

3D shape can also be estimated using active illumination techniques such as light stripes (Figure 13.1d) or time of flight range finders (Section 13.2). The partial surface models obtained using such techniques (or passive image-based stereo) can then be merged into more coherent 3D surface models (Figure 13.1e), as discussed in Section 13.2.1. Such techniques have been used to construct highly detailed and accurate models of cultural heritage such as historic sites (Section 13.2.2). The resulting surface models can then be simplified to support viewing at different resolutions and streaming across the web (Section 13.3.2). An alternative to working with continuous surfaces is to represent 3D surfaces as dense collections of 3D oriented points (Section 13.4) or as volumetric primitives (Section 13.5).

3D modeling can be more efficient and effective if we know something about the objects we are trying to reconstruct. In Section 13.6, we look at three specialized but commonly occurring examples, namely architecture (Figure 13.1g), heads and faces (Figure 13.1h), and whole bodies (Figure 13.1i). In addition to modeling people, we also discuss techniques for tracking them.

The last stage of shape and appearance modeling is to extract some colored textures to paint onto our 3D models (Section 13.7). Some techniques go beyond this and actually estimate full BRDFs (Section 13.7.1), although if there is no desire to re-light the scene, Surface Light Fields may be easier to acquire (Section 14.3.2).

Because there exists such a large variety of techniques to perform 3D modeling, this chapter does not go into detail on any one of these. Readers are encouraged to find more information in the cited references and recent computer vision conferences, as well as more

specialized conferences devoted to these topics, e.g., the International Conference on 3D Vision (3DV) and the IEEE International Conference on Automatic Face and Gesture Recognition (FG).

13.1 Shape from X

In addition to binocular disparity, shading, texture, and focus all play a role in how we perceive shape. The study of how shape can be inferred from such cues is sometimes called *shape from X*, because the individual instances are called *shape from shading*, *shape from texture*, and *shape from focus*.¹ In this section, we look at these three cues and how they can be used to reconstruct 3D geometry. A good overview of all these topics can be found in the collection of papers on physics-based shape inference edited by Wolff, Shafer, and Healey (1992b), the survey by Ackermann and Goesele (2015) and the book by Ikeuchi, Matsushita *et al.* (2020).

13.1.1 Shape from shading and photometric stereo

When you look at images of smooth shaded objects, such as the ones shown in Figure 13.2, you can clearly see the shape of the object from just the shading variation. How is this possible? The answer is that as the surface normal changes across the object, the apparent brightness changes as a function of the angle between the local surface orientation and the incident illumination, as shown in Figure 2.15 (Section 2.2.2).

The problem of recovering the shape of a surface from this intensity variation is known as *shape from shading* and is one of the classic problems in computer vision (Horn 1975). The collection of papers edited by Horn and Brooks (1989) is a great source of information on this topic, especially the chapter on variational approaches. The survey by Zhang, Tsai *et al.* (1999) not only reviews more recent techniques, but also provides some comparative results.

Most shape from shading algorithms assume that the surface under consideration is of a uniform albedo and reflectance, and that the light source directions are either known or can be calibrated by the use of a reference object. Under the assumptions of distant light sources and observer, the variation in intensity (*irradiance equation*) becomes purely a function of the local surface orientation,

$$I(x, y) = R(p(x, y), q(x, y)), \quad (13.1)$$

¹We have already seen examples of shape from stereo, shape from profiles, and shape from silhouettes in Chapter 12.

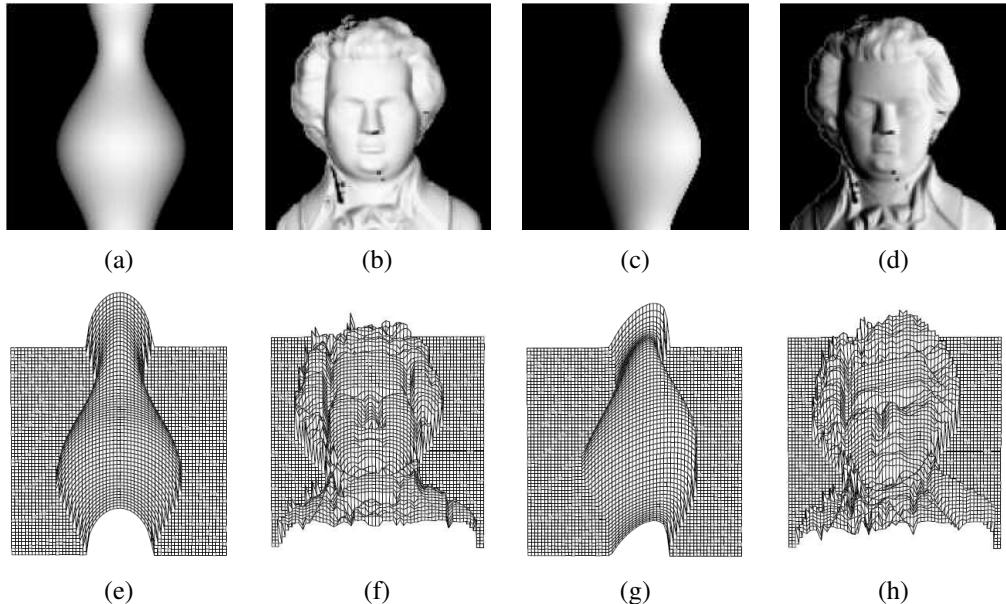


Figure 13.2 Synthetic shape from shading (Zhang, Tsai et al. 1999) © 1999 IEEE: shaded images, (a–b) with light from in front $(0, 0, 1)$ and (c–d) with light from the front right $(1, 0, 1)$; (e–f) corresponding shape from shading reconstructions using the technique of Tsai and Shah (1994).

where $(p, q) = (z_x, z_y)$ are the depth map derivatives and $R(p, q)$ is called the *reflectance map*. For example, a diffuse (Lambertian) surface has a reflectance map that is the (non-negative) dot product (2.89) between the surface normal $\hat{\mathbf{n}} = (p, q, 1)/\sqrt{1 + p^2 + q^2}$ and the light source direction $\mathbf{v} = (v_x, v_y, v_z)$,

$$R(p, q) = \max \left(0, \rho \frac{pv_x + qv_y + v_z}{\sqrt{1 + p^2 + q^2}} \right), \quad (13.2)$$

where ρ is the surface reflectance factor (albedo).

In principle, Equations (13.1–13.2) can be used to estimate (p, q) using non-linear least squares or some other method. Unfortunately, unless additional constraints are imposed, there are more unknowns per pixel (p, q) than there are measurements (I). One commonly used constraint is the smoothness constraint,

$$\mathcal{E}_s = \int p_x^2 + p_y^2 + q_x^2 + q_y^2 dx dy = \int \|\nabla p\|^2 + \|\nabla q\|^2 dx dy, \quad (13.3)$$

which we have already seen in Section 4.2 (4.18). The other is the *integrability constraint*,

$$\mathcal{E}_i = \int (p_y - q_x)^2 dx dy, \quad (13.4)$$

which arises naturally, because for a valid depth map $z(x, y)$ with $(p, q) = (z_x, z_y)$, we have $p_y = z_{xy} = z_{yx} = q_x$.

Instead of first recovering the orientation fields (p, q) and integrating them to obtain a surface, it is also possible to directly minimize the discrepancy in the image formation equation (13.1) while finding the optimal depth map $z(x, y)$ (Horn 1990). Unfortunately, shape from shading is susceptible to local minima in the search space and, like other variational problems that involve the simultaneous estimation of many variables, can also suffer from slow convergence. Using multi-resolution techniques (Szeliski 1991a) can help accelerate the convergence, while using more sophisticated optimization techniques (Dupuis and Olien-sis 1994) can help avoid local minima.

In practice, surfaces other than plaster casts are rarely of a single uniform albedo. Shape from shading therefore needs to be combined with some other technique or extended in some way to make it useful. One way to do this is to combine it with stereo matching (Fua and Leclerc 1995; Logothetis, Mecca, and Cipolla 2019) or known texture (surface patterns) (White and Forsyth 2006). The stereo and texture components provide information in textured regions, while shape from shading helps fill in the information across uniformly colored regions and also provides finer information about surface shape.

Photometric stereo. Another way to make shape from shading more reliable is to use multiple light sources that can be selectively turned on and off. This technique is called *photometric stereo*, as the light sources play a role analogous to the cameras located at different locations in traditional stereo (Woodham 1981).² For each light source, we have a different reflectance map, $R_1(p, q)$, $R_2(p, q)$, etc. Given the corresponding intensities I_1 , I_2 , etc. at a pixel, we can in principle recover both an unknown albedo ρ and a surface orientation estimate (p, q) .

For diffuse surfaces (13.2), if we parameterize the local orientation by $\hat{\mathbf{n}}$, we get (for non-shadowed pixels) a set of linear equations of the form

$$I_k = \rho \hat{\mathbf{n}} \cdot \mathbf{v}_k, \quad (13.5)$$

from which we can recover $\rho \hat{\mathbf{n}}$ using linear least squares. These equations are well conditioned as long as the (three or more) vectors \mathbf{v}_k are linearly independent, i.e., they are not along the same azimuth (direction away from the viewer).

²An alternative to turning lights on-and-off is to use three colored lights (Woodham 1994; Hernandez, Vogiatzis *et al.* 2007; Hernández and Vogiatzis 2010).



Figure 13.3 Multi-view photometric stereo (Logothetis, Mecca, and Cipolla 2019) © 2019 IEEE: initial COLMAP multi-view stereo reconstruction; refined with (Park, Sinha et al. 2017); and (Logothetis, Mecca, and Cipolla 2019).

Once the surface normals or gradients have been recovered at each pixel, they can be integrated into a depth map using a variant of regularized surface fitting (4.24). Nehab, Rusinkiewicz *et al.* (2005) and Harker and O’Leary (2008) discuss more sophisticated techniques for doing this. The combination of multi-view stereo for coarse shape and photometric stereo for fine detail continues to be an active area of research (Hernández, Vogiatzis, and Cipolla 2008; Wu, Liu *et al.* 2010; Park, Sinha *et al.* 2017). Logothetis, Mecca, and Cipolla (2019) describe such a system that can produce very high-quality scans (Figure 13.3), although it requires a sophisticated laboratory setup. A more practical setup that only requires a stereo camera and a flash to produce a flash/non-flash pair is described by Cao, Waechter *et al.* (2020). It is also possible to apply photometric stereo to outdoor web camera sequences (Figure 13.4), using the trajectory of the Sun as a variable direction illuminator (Ackermann, Langguth *et al.* 2012).

When surfaces are specular, more than three light directions may be required. In fact, the irradiance equation given in (13.1) not only requires that the light sources and camera be distant from the surface, it also neglects inter-reflections, which can be a significant source of the shading observed on object surfaces, e.g., the darkening seen inside concave structures such as grooves and crevasses (Nayar, Ikeuchi, and Kanade 1991). However, if one can control the placements of lights and cameras so that they are *reciprocal*, i.e., the position of lights and cameras can be (conceptually) switched, it is possible to recover constraints on surface depths and normals using a procedure known as *Helmholtz stereopsis* (Zickler, Belhumeur, and Kriegman 2002).

While earlier work on photometric stereo assumed known illuminant directions and reflectance (BRDF) functions, more recent work aims to loosen these constraints. Ackermann

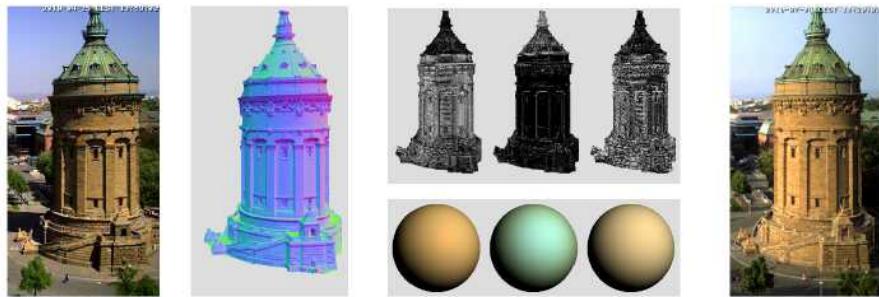


Figure 13.4 Webcam-based outdoor photometric stereo (Ackermann, Langguth et al. 2012) © 2012 IEEE: an input image, the recovered normal map, three basis BRDFs below their respective material maps, and a synthetic rendering from a new sun position.

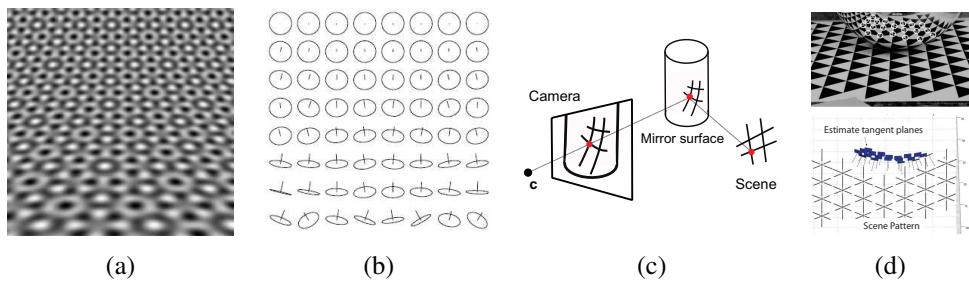


Figure 13.5 Synthetic shape from texture (Gårding 1992) © 1992 Springer: (a) regular texture wrapped onto a curved surface and (b) the corresponding surface normal estimates. Shape from mirror reflections (Savarese, Chen, and Perona 2005) © 2005 Springer: (c) a regular pattern reflecting off a curved mirror gives rise to (d) curved lines, from which 3D point locations and normals can be inferred.

and Goesele (2015) provide an extensive survey of such techniques, while Shi, Mo *et al.* (2019) describe their DiLiGenT dataset and benchmark for evaluating non-Lambertian photometric stereo and cite over 100 related papers. As with other areas of computer vision, deep networks and end-to-end learning are now commonly used to recover shape and illuminant direction from photometrics stereo. Some recent papers include Chen, Han *et al.* (2019), Li, Robles-Kelly *et al.* (2019), Haefner, Ye *et al.* (2019), Chen, Waechter *et al.* (2020), and Santo, Waechter, and Matsushita (2020).

13.1.2 Shape from texture

The variation in foreshortening observed in regular textures can also provide useful information about local surface orientation. Figure 13.5 shows an example of such a pattern, along with the estimated local surface orientations. Shape from texture algorithms require a number of processing steps, including the extraction of repeated patterns or the measurement of local frequencies to compute local affine deformations, and a subsequent stage to infer local surface orientation. Details on these various stages can be found in the research literature (Witkin 1981; Ikeuchi 1981; Blostein and Ahuja 1987; Gårding 1992; Malik and Rosenholtz 1997; Lobay and Forsyth 2006). A more recent paper uses a generative model to represent the repetitive appearance of textures and jointly optimizes the model along with the local surface orientations at every pixel (Verbin and Zickler 2020).

When the original pattern is regular, it is possible to fit a regular but slightly deformed grid to the image and use this grid for a variety of image replacement or analysis tasks (Liu, Collins, and Tsin 2004; Liu, Lin, and Hays 2004; Hays, Leordeanu *et al.* 2006; Lin, Hays *et al.* 2006; Park, Brocklehurst *et al.* 2009). This process becomes even easier if specially printed textured cloth patterns are used (White and Forsyth 2006; White, Crane, and Forsyth 2007).

The deformations induced in a regular pattern when it is viewed in the reflection of a curved mirror, as shown in Figure 13.5c–d, can be used to recover the shape of the surface (Savarese, Chen, and Perona 2005; Rozenfeld, Shimshoni, and Lindenbaum 2011). It is also possible to infer local shape information from *specular flow*, i.e., the motion of specularities when viewed from a moving camera (Oren and Nayar 1997; Zisserman, Giblin, and Blake 1989; Swaminathan, Kang *et al.* 2002).

13.1.3 Shape from focus

A strong cue for object depth is the amount of blur, which increases as the object's surface moves away from the camera's focusing distance. As shown in Figure 2.19, moving the object surface away from the focus plane increases the circle of confusion, according to a formula that is easy to establish using similar triangles (Exercise 2.4).

A number of techniques have been developed to estimate depth from the amount of defocus (*depth from defocus*) (Pentland 1987; Nayar and Nakagawa 1994; Nayar, Watanabe, and Noguchi 1996; Watanabe and Nayar 1998; Chaudhuri and Rajagopalan 1999; Favaro and Soatto 2006). To make such a technique practical, a number of issues need to be addressed:

- The amount of blur increase in *both* directions as you move away from the focus plane. Therefore, it is necessary to use two or more images captured with different focus

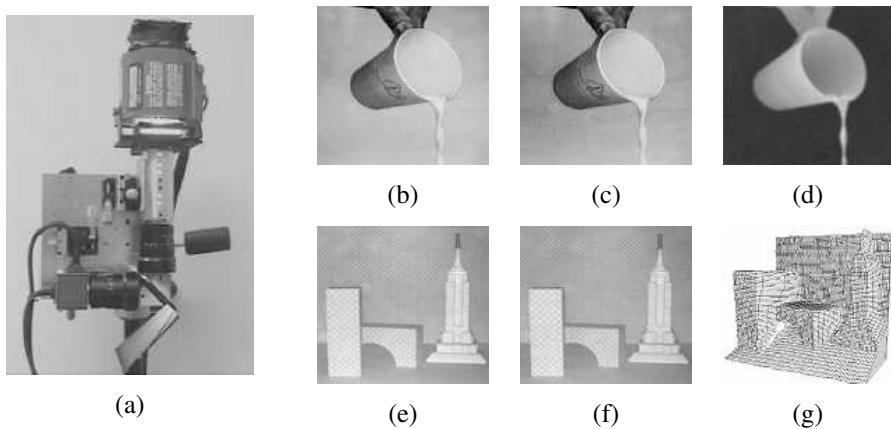


Figure 13.6 *Real-time depth from defocus (Nayar, Watanabe, and Noguchi 1996)* © 1996 IEEE: (a) the real-time focus range sensor, which includes a half-silvered mirror between the two telecentric lenses (lower right), a prism that splits the image into two CCD sensors (lower left), and an edged checkerboard pattern illuminated by a Xenon lamp (top); (b–c) input video frames from the two cameras along with (d) the corresponding depth map; (e–f) two frames (you can see the texture if you zoom in) and (g) the corresponding 3D mesh model.

distance settings (Pentland 1987; Nayar, Watanabe, and Noguchi 1996) or to translate the object in depth and look for the point of maximum sharpness (Nayar and Nakagawa 1994).

- The magnification of the object can vary as the focus distance is changed or the object is moved. This can be modeled either explicitly (making correspondence more difficult) or using *telecentric optics*, which approximate an orthographic camera and require an aperture in front of the lens (Nayar, Watanabe, and Noguchi 1996).
- The amount of defocus must be reliably estimated. A simple approach is to average the squared gradient in a region, but this suffers from several problems, including the image magnification problem mentioned above. A better solution is to use carefully designed *rational filters* (Watanabe and Nayar 1998).

Figure 13.6 shows an example of a real-time depth from defocus sensor, which employs two imaging chips at slightly different depths sharing a common optical path, as well as an active illumination system that projects a checkerboard pattern from the same direction. As you can see in Figure 13.6b–g, the system produces high-accuracy real-time depth maps for both static and dynamic scenes.

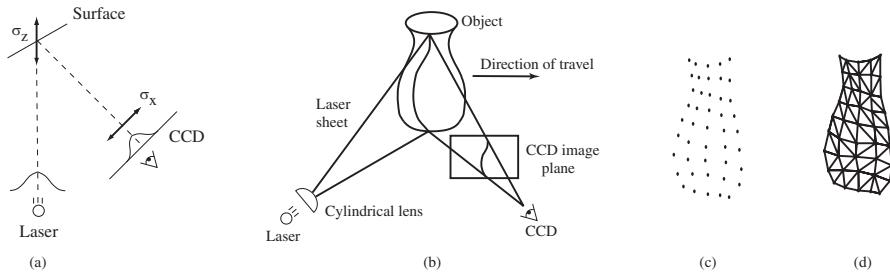


Figure 13.7 Range data scanning (Curless and Levoy 1996) © 1996 ACM: (a) a laser dot on a surface is imaged by a CCD sensor; (b) a laser stripe (sheet) is imaged by the sensor (the deformation of the stripe encodes the distance to the object); (c) the resulting set of 3D points are turned into (d) a triangulated mesh.

13.2 3D scanning

As we have seen in the previous section, actively lighting a scene, whether for the purpose of estimating normals using photometric stereo, or for adding artificial texture for shape from defocus, can greatly improve the performance of vision systems. This kind of *active illumination* has been used from the earliest days of machine vision to construct highly precise sensors for estimating 3D depth images using a variety of *rangefinding* (or *range sensing*) techniques (Besl 1989; Curless 1999; Hebert 2000; Zhang 2018).³ While rangefinders such as lidar (Light Detection and Ranging) and laser-based 3D scanners were once limited to commercial and laboratory applications, the development of low-cost *depth cameras* such as the Microsoft Kinect (Zhang 2012) have revolutionized many aspects of computer vision. It is now common to refer to the registered color and depth frames produced by such cameras as *RGB-D* (or *RGBD*) images (Silberman, Hoiem *et al.* 2012).

One of the early active illumination sensors used in computer vision and computer graphics was a laser or light stripe sensor, which sweeps a plane of light across the scene or object while observing it from an offset viewpoint, as shown in Figure 13.7b (Rioux and Bird 1993; Curless and Levoy 1995). As the stripe falls across the object, it deforms its shape according to the shape of the surface it is illuminating. It is then a simple matter of using *optical triangulation* to estimate the 3D locations of all the points seen in a particular stripe. In more detail, knowledge of the 3D plane equation of the light stripe allows us to infer the 3D location corresponding to each illuminated pixel, as previously discussed in (2.70–2.71). The accuracy of light striping techniques can be improved by finding the exact temporal peak in

³Rangefinding is an old-fashioned word for measuring distance, often using passive or active optical means.

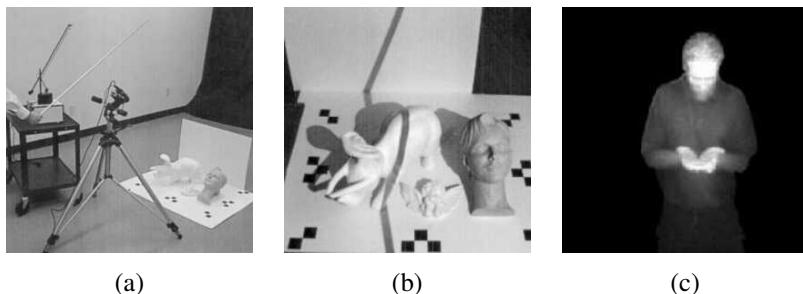


Figure 13.8 *Shape scanning using cast shadows (Bouguet and Perona 1999) © 1999 Springer: (a) camera setup with a point light source (a desk lamp without its reflector), a hand-held stick casting a shadow, and (b) the objects being scanned in front of two planar backgrounds. (c) Real-time depth map using a pulsed illumination system (Iddan and Yahav 2001) © 2001 SPIE.*

illumination for each pixel (Curless and Levoy 1995). The final accuracy of a scanner can be determined using slant edge modulation techniques, i.e., by imaging sharp creases in a calibration object (Goesele, Fuchs, and Seidel 2003).

An interesting variant on light stripe rangefinding is presented by Bouguet and Perona (1999). Instead of projecting a light stripe, they simply wave a stick casting a shadow over a scene or object illuminated by a point light source such as a lamp or the Sun (Figure 13.8a). As the shadow falls across two background planes whose orientation relative to the camera is known (or inferred during pre-calibration), the plane equation for each stripe can be inferred from the two projected lines, whose 3D equations are known (Figure 13.8b). The deformation of the shadow as it crosses the object being scanned then reveals its 3D shape, as with regular light stripe rangefinding (Exercise 13.2). This technique can also be used to estimate the 3D geometry of a background scene and how its appearance varies as it moves into shadow, to cast new shadows onto the scene (Chuang, Goldman *et al.* 2003) (Section 10.4.3).

The time it takes to scan an object using a light stripe technique is proportional to the number of depth planes used, which is usually comparable to the number of pixels across an image. A much faster scanner can be constructed by turning different projector pixels on and off in a structured manner, e.g., using a binary or Gray code (Besl 1989). For example, let us assume that the LCD projector we are using has 1,024 columns of pixels. Taking the 10-bit binary code corresponding to each column's address (0...1,023), we project the first bit, then the second, etc. After 10 projections (e.g., a third of a second for a synchronized 30Hz camera-projector system), each pixel in the camera knows which of the 1,024 columns of projector light it is seeing. A similar approach can also be used to estimate the refractive

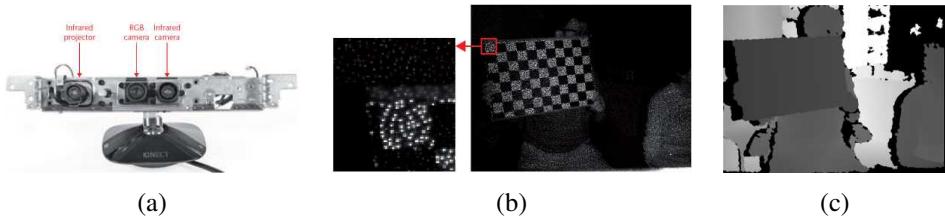


Figure 13.9 The Microsoft Kinect depth camera (Zhang 2012) © 2012 IEEE: (a) the hardware, comprising an infrared (IR) speckle pattern projector and a color and IR camera pair; (b) close-up of a sample infrared image, showing the projected dots; (c) the final depth map, which has black “shadows” in the areas not illuminated by the projector.

properties of an object by placing a monitor behind the object (Zongker, Werner *et al.* 1999; Chuang, Zongker *et al.* 2000) (Section 14.4). Very fast scanners can also be constructed with a single laser beam, i.e., a real-time *flying spot* optical triangulation scanner (Rioux, Bechthold *et al.* 1987).

If even faster, i.e., frame-rate, scanning is required, we can project a single textured pattern into the scene. Proesmans, Van Gool, and Defoort (1998) describe a system where a checkerboard grid is projected onto an object and the deformation of the grid is used to infer 3D shape. Unfortunately, such a technique only works if the surface is continuous enough to link all of the grid points. Instead of projecting a grid, it is also possible to project one or more sinusoidal *fringe patterns* and to then recover deformations in the surface from the relative phase displacements using a process called *fringe projection profilometry* (Su and Zhang 2010; Zuo, Huang *et al.* 2016; Zhang 2018).

The Microsoft Kinect (Zhang 2012) depth camera uses a variant of this technique, projecting an infrared (IR) *speckle pattern*, which looks like a bunch of random dots, but which in fact consists of a known calibrated pseudo-random pattern (Figure 13.9). By measuring the horizontal displacement (parallax) between the dots seen in the IR camera and their expected locations, a depth map can be computed, interpolating over the pixels not illuminated by the dots (Fanello, Rhemann *et al.* 2016; Fanello, Valentin *et al.* 2017b). Since its release, the Kinect camera has been widely used in computer vision research (Zhang 2012; Han, Shao *et al.* 2013), as well as applications such as 3D body tracking (Section 13.6.4) and object scanning and home interior reconstruction (Section 13.2.1). Kinect sensors were used to create the first widely used dataset for 3D semantic scene understanding (Silberman, Hoiem *et al.* 2012), although larger 3D scanned datasets have since been created (Dai, Chang *et al.* 2017).

A higher resolution system can be constructed using high-speed custom illumination and sensing hardware. Iddan and Yahav (2001) describe the construction of their 3DV Zcam

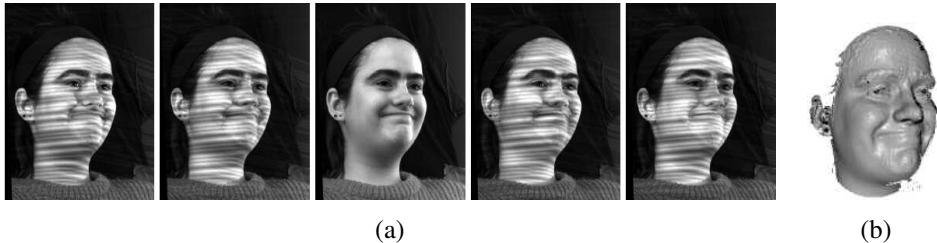


Figure 13.10 *Real-time dense 3D face capture using spacetime stereo (Zhang, Snavely et al. 2004) © 2004 ACM:* (a) set of five consecutive video frames from one of two stereo cameras (every fifth frame is free of stripe patterns, in order to extract texture); (b) resulting high-quality 3D surface model (depth map visualized as a shaded rendering).

video-rate depth sensing camera, which projects a pulsed plane of light onto the scene and then integrates the returning light for a short interval, essentially obtaining time-of-flight measurement for the distance to individual pixels in the scene. A good description of earlier time-of-flight systems, including amplitude and frequency modulation schemes for lidar, can be found in (Besl 1989), and a more recent description can be found in the book by Hansard, Lee *et al.* (2012). While the initial version of the Microsoft Kinect depth camera used a speckle pattern structured light system (Zhang 2012), the newer Kinect V2 uses a time-of-flight (ToF) sensor that uses phase measurements of amplitude-modulated light signals (Bamji, O'Connor *et al.* 2014). Traditional multi-frequency phase unwrapping techniques can be used to estimate absolute depth, but more accurate depths for dynamic scenes can be obtained by simultaneously modeling depths and object velocities (Stühmer, Nowozin *et al.* 2015).

Instead of using a single camera, it is also possible to construct an active illumination range sensor using stereo imaging setups, resulting in a system that is often called *active (illumination) stereo*. The simplest way to do this is to just project random stripe patterns onto the scene to create synthetic texture, which helps match textureless surfaces (Kang, Webb *et al.* 1995). Projecting a known series of stripes, just as in coded pattern single-camera rangefinding, makes the correspondence between pixels unambiguous and allows for the recovery of depth estimates at pixels only seen in a single camera (Scharstein and Szeliski 2003). This technique has been used to produce large numbers of highly accurate registered multi-image stereo pairs and depth maps for the purpose of evaluating stereo correspondence algorithms (Scharstein and Szeliski 2002; Hirschmüller and Scharstein 2009; Scharstein, Hirschmüller *et al.* 2014) and learning depth map priors and parameters (Pal, Weinman *et al.* 2012). Carefully designed algorithms can perform local matching of patterns at 500Hz (Fanello, Valentin

et al. 2017a,b).

While projecting multiple patterns usually requires the scene or object to remain still, additional processing can enable the production of real-time depth maps for dynamic scenes. The basic idea (Davis, Ramamoorthi, and Rusinkiewicz 2003; Zhang, Curless, and Seitz 2003) is to assume that depth is nearly constant within a 3D space–time window around each pixel and to use the 3D window for matching and reconstruction. Depending on the surface shape and motion, this assumption may be error-prone, as shown in Davis, Nahab *et al.* (2005). To model shapes more accurately, Zhang, Curless, and Seitz (2003) model the linear disparity variation within the space–time window and show that better results can be obtained by globally optimizing disparity and disparity gradient estimates over video volumes (Zhang, Snavely *et al.* 2004). Figure 13.10 shows the results of applying this system to a person’s face; the frame-rate 3D surface model can then be used for further model-based fitting and computer graphics manipulation (Section 13.6.2). As mentioned previously, motion modeling can also be applied to phase-based time-of-flight sensors (Stühmer, Nowozin *et al.* 2015).

One word of caution about active range sensing. When the surfaces being scanned are too reflective, the camera may see a reflection off the object’s surface and assume that this *virtual image* is the true scene. For surfaces with moderate amounts of reflection, such as the ceramic models in Wood, Azuma *et al.* (2000) or the Corn Cho puffs in Park, Newcombe, and Seitz (2018), there is still sufficient diffuse reflection under the specular layer to obtain a 3D range map. (The specular part can then be recovered separately to produce a *surface light field*, as described in Section 14.3.2.) However, for true mirrors, active range scanners will invariably capture the virtual 3D model seen reflected in the mirror, so that additional techniques such as looking for a reflection of the scanning device must be used (Whelan, Goesele *et al.* 2018).

13.2.1 Range data merging

While individual range images can be useful for applications such as real-time z-keying or facial motion capture, they are often used as building blocks for more complete 3D object modeling. In such applications, the next two steps in processing are the registration (alignment) of partial 3D surface models and their integration into coherent 3D surfaces (Curless 1999). If desired, this can be followed by a model fitting stage using either parametric representations such as generalized cylinders (Agin and Binford 1976; Nevatia and Binford 1977; Marr and Nishihara 1978; Brooks 1981), superquadrics (Pentland 1986; Solina and Bajcsy 1990; Terzopoulos and Metaxas 1991), or non-parametric models such as triangular meshes (Boissonnat 1984) or physically based models (Terzopoulos, Witkin, and Kass 1988; Delingette, Hebert, and Ikeuchi 1992; Terzopoulos and Metaxas 1991; McInerney and Terzopoulos 1993; Terzopoulos 1999). A number of techniques have also been developed for segmenting range

images into simpler constituent surfaces (Hoover, Jean-Baptiste *et al.* 1996).

The most widely used 3D registration technique is the *iterative closest point* (ICP) algorithm, which alternates between finding the closest point matches between the two surfaces being aligned and then solving a 3D *absolute orientation* problem (Section 8.1.5, (8.31–8.32) (Besl and McKay 1992; Chen and Medioni 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan *et al.* 1998; David, DeMenthon *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). Some techniques, such as the one developed by Chen and Medioni (1992), use local surface tangent planes to make this computation more accurate and to accelerate convergence. More recently, Rusinkiewicz (2019) proposed a symmetric oriented point distance similar to the energy terms used in *oriented particles* (Szeliski and Tonnesen 1992). A nice review of ICP and its related variants can be found in the papers by Tam, Cheng *et al.* (2012) and Pomerleau, Colas, and Siegwart (2015).

As the two surfaces being aligned usually only have partial overlap and may also have outliers, robust matching criteria (Section 8.1.4 and Appendix B.3) are typically used. To speed up the determination of the closest point, and also to make the distance-to-surface computation more accurate, one of the two point sets (e.g., the current merged model) can be converted into a *signed distance function*, optionally represented using an *octree spline* for compactness (Lavallée and Szeliski 1995). Variants on the basic ICP algorithm can be used to register 3D point sets under non-rigid deformations, e.g., for medical applications (Feldmar and Ayache 1996; Szeliski and Lavallée 1996). Color values associated with the points or range measurements can also be used as part of the registration process to improve robustness (Johnson and Kang 1997; Pulli 1999).

Unfortunately, the ICP algorithm and its variants can only find a locally optimal alignment between 3D surfaces. If this is not known *a priori*, more global correspondence or search techniques, based on local descriptors invariant to 3D rigid transformations, need to be used. An example of such a descriptor is the *spin image*, which is a local circular projection of a 3D surface patch around the local normal axis (Johnson and Hebert 1999). Another (earlier) example is the *splash* representation introduced by Stein and Medioni (1992). More recent work along these lines studies the problem of *pose estimation* (Section 11.2) from RGB-D images, which is essentially the same problem as aligning a range map to a 3D model. Recent papers on this topic (Drost, Ulrich *et al.* 2010; Brachmann, Michel *et al.* 2016; Vidal, Lin *et al.* 2018) typically evaluate themselves on the Benchmark for 6DOF Object Pose Estimation,⁴ which also hosts a series of yearly workshops on this topic.

Once two or more 3D surfaces have been aligned, they can be merged into a single model. One approach is to represent each surface using a triangulated mesh and combine these

⁴<https://bop.felk.cvut.cz/home>

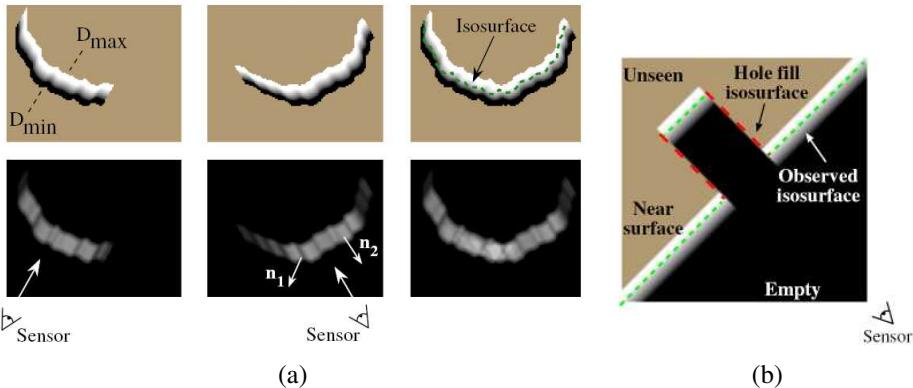


Figure 13.11 Range data merging (Curless and Levoy 1996) © 1996 ACM: (a) two signed distance functions (top left) are merged with their (weights) bottom left to produce a combined set of functions (right column) from which an isosurface can be extracted (green dashed line); (b) the signed distance functions are combined with empty and unseen space labels to fill holes in the isosurface.

meshes using a process that is sometimes called *zippering* (Soucy and Laurendeau 1992; Turk and Levoy 1994). Another, now more widely used, approach is to compute a (truncated) signed distance function that fits all of the 3D data points (Hoppe, DeRose *et al.* 1992; Curless and Levoy 1996; Hilton, Stoddart *et al.* 1996; Wheeler, Sato, and Ikeuchi 1998).

Figure 13.11 shows one such approach, the *volumetric range image processing* (VRIP) technique developed by Curless and Levoy (1996), which first computes a weighted signed distance function from each range image and then merges them using a weighted averaging process. To make the representation more compact, run-length coding is used to encode the empty, seen, and varying (signed distance) voxels, and only the signed distance values near each surface are stored.⁵ Once the merged signed distance function has been computed, a zero-crossing surface extraction algorithm, such as *marching cubes* (Lorensen and Cline 1987), can be used to recover a meshed surface model. Figure 13.12 shows an example of the complete range data merging and isosurface extraction pipeline. Rusinkiewicz, Hall-Holt, and Levoy (2002) present a real-time system that combines fast ICP and point-based merging and rendering.

The advent of consumer-level RGB-D cameras such as Kinect created renewed interest in large-scale range data registration and merging (Zhang 2012; Han, Shao *et al.* 2013). An influential paper in this area is Kinect Fusion (Izadi, Kim *et al.* 2011; Newcombe, Izadi *et*

⁵An alternative, even more compact, representation could be to use octrees (Lavallée and Szeliski 1995).

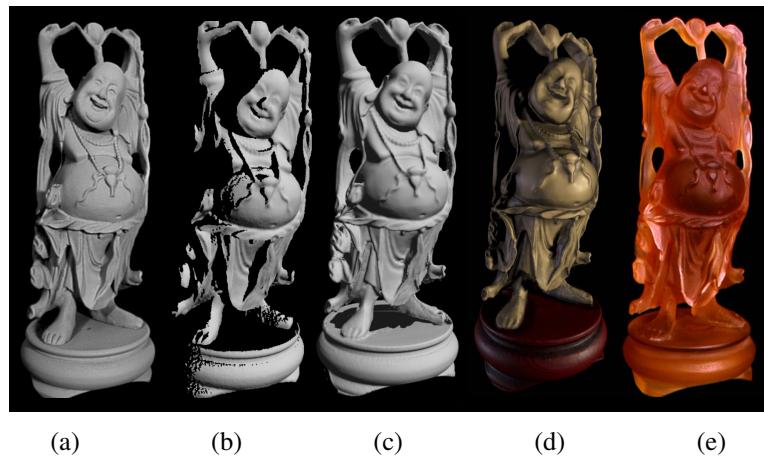


Figure 13.12 Reconstruction and hardcopy of the “Happy Buddha” statuette (Curless and Levoy 1996) © 1996 ACM: (a) photograph of the original statue after spray painting with matte gray; (b) partial range scan; (c) merged range scans; (d) colored rendering of the reconstructed model; (e) hardcopy of the model constructed using stereolithography.



Figure 13.13 Fusing multiple depth images using the KinectFusion real-time system (Newcombe, Izadi et al. 2011) © 2011 IEEE. The three images show an original (noisy) range scan, rendered as a colored normal map, and the fused 3D model, rendered as both a normal map and Phong-shaded.

al. 2011), which combines an ICP-like SLAM technique called DTAM (Newcombe, Lovegrove, and Davison 2011) with real-time TSDF (truncated signed distance function) volumetric integration, which is described in more detail in Section 13.5.1. Follow-on papers include Elastic Fragments for non-rigid alignment (Zhou, Miller, and Koltun 2013), Octomap (Hornung, Wurm *et al.* 2013), which uses an octree and probabilistic occupancy, and Voxel Hashing (Nießner, Zollhöfer *et al.* 2013) and Chisel (Klingensmith, Dryanovski *et al.* 2015), both of which uses spatial hashing to compress the TSDF. KinectFusion has also been extended to handle highly variable scanning resolution (Fuhrmann and Gosele 2011, 2014), dynamic scenes (DynamicFusion (Newcombe, Fox, and Seitz 2015), VolumeDeform (Innmann, Zollhöfer *et al.* 2016), and Motion2Fusion (Dou, Davidson *et al.* 2017)), to use non-rigid surface deformations for global model refinement (ElasticFusion: Whelan, Salas-Moreno *et al.* (2016)), to produce a globally consistent BundleFusion model (Dai, Nießner *et al.* 2017), and to use a deep network to perform the non-rigid matching (Božič, Zollhöfer *et al.* 2020). More details on these and other techniques for constructing 3D models from RGB-D scans can be found in the survey by Zollhöfer, Stotko *et al.* (2018).

Some of the most recent work in range data merging uses neural networks to represent the TSDF (Park, Florence *et al.* 2019), update a TSDF with incoming range data scans (Weder, Schonberger *et al.* 2020, 2021), or provide local priors (Chabra, Lenssen *et al.* 2020). Range data merging techniques are often used for both 3D object scanning and for visual map building and navigation (RGB-D SLAM), which we discussed in Section 11.5. And now that depth sensing (aka lidar) technology is starting to appear in mobile phones, it can be used to build complete texture-mapped 3D room models, e.g., using Occipital’s Canvas app (Stein 2020).⁶

Volumetric range data merging techniques based on signed distance or characteristic (inside–outside) functions are also widely used to extract smooth well-behaved surfaces from oriented or unoriented point sets (Hoppe, DeRose *et al.* 1992; Ohtake, Belyaev *et al.* 2003; Kazhdan, Bolitho, and Hoppe 2006; Lempitsky and Boykov 2007; Zach, Pock, and Bischof 2007b; Zach 2008), as discussed in more detail in Section 13.5.1 and the survey paper by Berger, Tagliasacchi *et al.* (2017).

13.2.2 Application: Digital heritage

Active rangefinding technologies, combined with surface modeling and appearance modeling techniques (Section 13.7), are widely used in the fields of archaeological and historical preservation, which often also goes under the name *digital heritage* (MacDonald 2006). In

⁶<https://canvas.io>

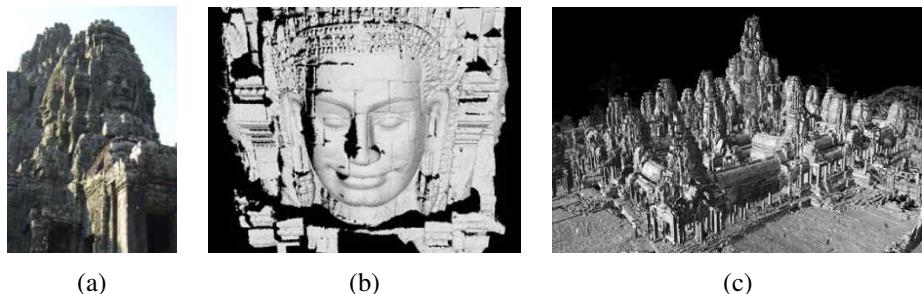


Figure 13.14 *Laser range modeling of the Bayon temple at Angkor-Thom (Banno, Masuda et al. 2008) © 2008 Springer: (a) sample photograph from the site; (b) a detailed head model scanned from the ground; (c) final merged 3D model of the temple scanned using a laser range sensor mounted on a balloon.*

such applications, detailed 3D models of cultural objects are acquired and later used for applications such as analysis, preservation, restoration, and the production of duplicate artwork (Rioux and Bird 1993).

A notable example of such an endeavor is the Digital Michelangelo project of Levoy, Pulli *et al.* (2000), which used Cyberware laser stripe scanners and high-quality digital SLR cameras mounted on a large gantry to obtain detailed scans of Michelangelo's David and other sculptures in Florence. The project also took scans of the *Forma Urbis Romae*, an ancient stone map of Rome that had shattered into pieces, for which new matches were obtained using digital techniques. The whole process, from initial planning, to software development, acquisition, and post-processing, took several years (and many volunteers), and produced a wealth of 3D shape and appearance modeling techniques as a result.

Even larger-scale projects have since been attempted, for example, the scanning of complete temple sites such as Angkor-Thom (Ikeuchi and Sato 2001; Ikeuchi and Miyazaki 2007; Banno, Masuda *et al.* 2008). Figure 13.14 shows details from this project, including a sample photograph, a detailed 3D (sculptural) head model scanned from ground level, and an aerial overview of the final merged 3D site model, which was acquired using a balloon.

13.3 Surface representations

In previous sections, we have seen different representations being used to integrate 3D range scans. We now look at several of these representations in more detail. Explicit surface representations, such as triangle meshes, splines (Farin 1992, 2002), and subdivision sur-

faces (Stollnitz, DeRose, and Salesin 1996; Zorin, Schröder, and Sweldens 1996; Warren and Weimer 2001; Peters and Reif 2008), enable not only the creation of highly detailed models but also processing operations, such as interpolation (Section 13.3.1), fairing or smoothing, and decimation and simplification (Section 13.3.2). We also examine discrete point-based representations (Section 13.4) and volumetric representations (Section 13.5).

13.3.1 Surface interpolation

One of the most common operations on surfaces is their reconstruction from a set of sparse data constraints, i.e., *scattered data interpolation*, which we covered in Section 4.1. When formulating such problems, surfaces may be parameterized as height fields $f(\mathbf{x})$, as 3D parametric surfaces $\mathbf{f}(\mathbf{x})$, or as non-parametric models such as collections of triangles.

In Section 4.2, we saw how two-dimensional function interpolation and approximation problems $\{d_i\} \rightarrow f(\mathbf{x})$ could be cast as energy minimization problems using regularization (4.18–4.23). Such problems can also specify the locations of discontinuities in the surface as well as local orientation constraints (Terzopoulos 1986b; Zhang, Dugas-Phocion *et al.* 2002).

One approach to solving such problems is to discretize both the surface and the energy on a discrete grid or mesh using finite element analysis (4.24–4.27) (Terzopoulos 1986b). Such problems can then be solved using sparse system solving techniques, such as multigrid (Briggs, Henson, and McCormick 2000) or hierarchically preconditioned conjugate gradient (Szeliski 2006b; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013). The surface can also be represented using a hierarchical combination of multilevel B-splines (Lee, Wolberg, and Shin 1997).

An alternative approach is to use *radial basis* (or *kernel*) functions (Boult and Kender 1986; Nielson 1993), which we covered in Section 4.1.1. As we mentioned in that section, if we want the function $\mathbf{f}(\mathbf{x})$ to exactly interpolate the data points, a dense linear system must be solved to determine the magnitude associated with each basis function (Boult and Kender 1986). It turns out that, for certain regularized problems, e.g., (4.18–4.21), there exist radial basis functions (kernels) that give the same results as a full analytical solution (Boult and Kender 1986). Unfortunately, because the dense system solving is cubic in the number of data points, basis function approaches can only be used for small problems such as feature-based image morphing (Beier and Neely 1992).

When a three-dimensional *parametric surface* is being modeled, the vector-valued function \mathbf{f} in (4.18–4.27) encodes 3D coordinates (x, y, z) on the surface and the domain $\mathbf{x} = (s, t)$ encodes the surface parameterization. One example of such surfaces are symmetry-seeking parametric models, which are elastically deformable versions of *generalized cylinder*.

ders⁷ (Terzopoulos, Witkin, and Kass 1987). In these models, s is the parameter *along* the spine of the deformable tube and t is the parameter *around* the tube. A variety of smoothness and radial symmetry forces are used to constrain the model while it is fitted to image-based silhouette curves.

It is also possible to define *non-parametric* surface models, such as general triangulated meshes, and to equip such meshes (using finite element analysis) with both internal smoothness metrics and external data fitting metrics (Sander and Zucker 1990; Fua and Sander 1992; Delingette, Hebert, and Ikeuchi 1992; McInerney and Terzopoulos 1993). While most of these approaches assume a standard *elastic* deformation model, which uses quadratic internal smoothness terms, it is also possible to use sub-linear energy models to better preserve surface creases (Diebel, Thrun, and Brüning 2006) or to use graph-convolutional neural networks (GCNNs) as an alternative to the update equations, as in Deep Active Surface Models (Wickramasinghe, Fua, and Knott 2021). Triangle meshes can also be augmented with either spline elements (Sullivan and Ponce 1998) or subdivision surfaces (Stollnitz, DeRose, and Salesin 1996; Zorin, Schröder, and Sweldens 1996; Warren and Weimer 2001; Peters and Reif 2008) to produce surfaces with better smoothness control.

Both parametric and non-parametric surface models assume that the topology of the surface is known and fixed ahead of time. For more flexible surface modeling, we can either represent the surface as a collection of oriented points (Section 13.4) or use 3D implicit functions (Section 13.5.1), which can also be combined with elastic 3D surface models (McInerney and Terzopoulos 1993).

The field of surface reconstruction from unorganized point samples continues to advance rapidly, with more recent work addressing issues with data imperfections, as described in the survey by Berger, Tagliasacchi *et al.* (2017) .

13.3.2 Surface simplification

Once a triangle mesh has been created from 3D data, it is often desirable to create a hierarchy of mesh models, for example, to control the displayed *level of detail* (LOD) in a computer graphics application. (In essence, this is a 3D analog to image pyramids (Section 3.5).) One approach to doing this is to approximate a given mesh with one that has *subdivision connectivity*, over which a set of triangular wavelet coefficients can then be computed (Eck, DeRose *et al.* 1995). A more continuous approach is to use sequential *edge collapse* operations to go from the original fine-resolution mesh to a coarse base-level mesh (Hoppe 1996; Lee,

⁷A generalized cylinder (Brooks 1981) is a *solid of revolution*, i.e., the result of rotating a (usually smooth) curve around an axis. It can also be generated by sweeping a slowly varying circular cross-section along the axis. (These two interpretations are equivalent.)

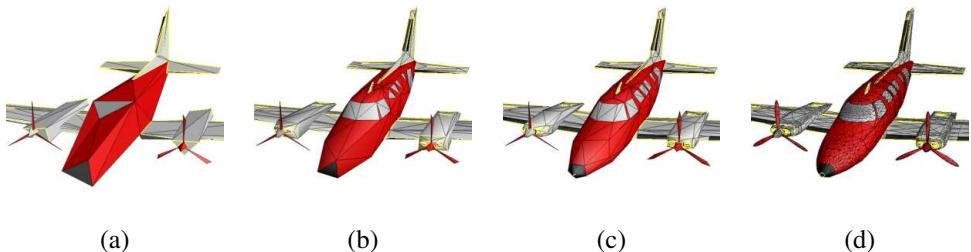


Figure 13.15 *Progressive mesh representation of an airplane model (Hoppe 1996) © 1996 ACM:* (a) base mesh M^0 (150 faces); (b) mesh M^{175} (500 faces); (c) mesh M^{425} (1,000 faces); (d) original mesh $M = M^n$ (13,546 faces).

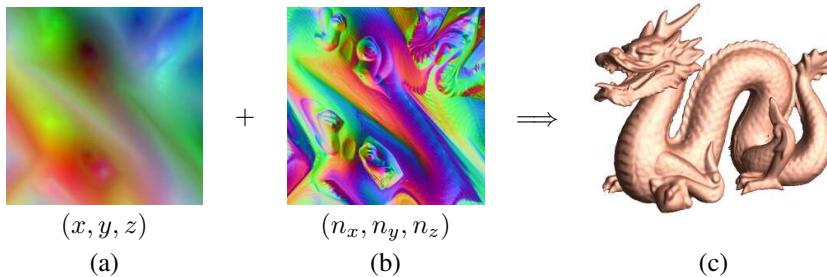


Figure 13.16 *Geometry images (Gu, Gortler, and Hoppe 2002) © 2002 ACM:* (a) the 257×257 geometry image defines a mesh over the surface; (b) the 512×512 normal map defines vertex normals; (c) final lit 3D model.

Sweldens *et al.* 1998). The resulting *progressive mesh* (PM) representation can be used to render the 3D model at arbitrary levels of detail, as shown in Figure 13.15. More recent papers on multiresolution geometric modeling can be found in the survey by Floater and Hormann (2005) and the collection of papers edited by Dodgson, Floater, and Sabin (2005).

13.3.3 Geometry images

While multi-resolution surface representations such as Eck, DeRose *et al.* (1995), Hoppe (1996), and Lee, Sweldens *et al.* (1998) support level of detail operations, they still consist of an irregular collection of triangles, which makes them more difficult to compress and store in a cache-efficient manner.⁸

⁸Subdivision triangulations, such as those in Eck, DeRose *et al.* (1995), are *semi-regular*, i.e., regular (ordered and nested) within each subdivided base triangle.

To make the triangulation completely regular (uniform and gridded), Gu, Gortler, and Hoppe (2002) describe how to create *geometry images* by cutting surface meshes along well-chosen lines and “flattening” the resulting representation into a square. Figure 13.16a shows the resulting (x, y, z) values of the surface mesh mapped over the unit square, while Figure 13.16b shows the associated (n_x, n_y, n_z) *normal map*, i.e., the surface normals associated with each mesh vertex, which can be used to compensate for loss in visual fidelity if the original geometry image is heavily compressed.

13.4 Point-based representations

As we mentioned previously, triangle-based surface models assume that the topology (and often the rough shape) of the 3D model is known ahead of time. While it is possible to re-mesh a model as it is being deformed or fitted, a simpler solution is to dispense with an explicit triangle mesh altogether and to have triangle vertices behave as *oriented points*, or particles, or *surface elements* (surfels) (Szeliski and Tonnesen 1992).

To endow the resulting particle system with internal smoothness constraints, pairwise interaction potentials can be defined that approximate the equivalent elastic bending energies that would be obtained using local finite-element analysis.⁹ Instead of defining the finite element neighborhood for each particle (vertex) ahead of time, a soft influence function is used to couple nearby particles. The resulting 3D model can change both topology and particle density as it evolves and can therefore be used to interpolate partial 3D data with holes (Szeliski, Tonnesen, and Terzopoulos 1993b). Discontinuities in both the surface orientation and crease curves can also be modeled (Szeliski, Tonnesen, and Terzopoulos 1993a).

To render the particle system as a continuous surface, local dynamic triangulation heuristics (Szeliski and Tonnesen 1992) or direct surface element *splatting* (Pfister, Zwicker *et al.* 2000) can be used. Another alternative is to first convert the point cloud into an implicit signed distance or inside–outside function, using either minimum signed distances to the oriented points (Hoppe, DeRose *et al.* 1992) or by interpolating a characteristic (inside–outside) function using radial basis functions (Turk and O’Brien 2002; Dinh, Turk, and Slabaugh 2002). Even greater precision over the implicit function fitting, including the ability to handle irregular point densities, can be obtained by computing a *moving least squares* (MLS) estimate of the signed distance function (Alexa, Behr *et al.* 2003; Pauly, Keiser *et al.* 2003), as shown in Figure 13.17. Further improvements can be obtained using local sphere fitting (Guennebaud and Gross 2007), faster and more accurate re-sampling (Guennebaud, Germann, and

⁹As mentioned before, an alternative is to use *sub-linear* interaction potentials, which encourage the preservation of surface creases (Diebel, Thrun, and Brüning 2006).

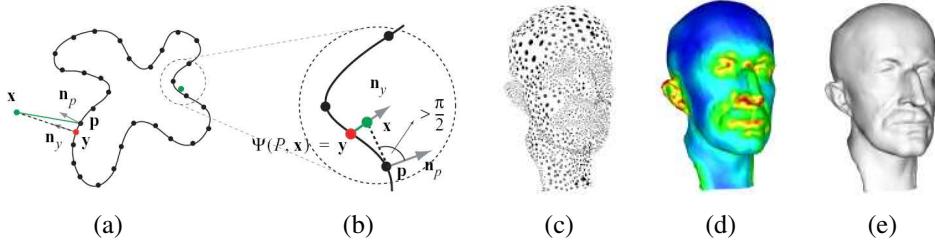


Figure 13.17 Point-based surface modeling with moving least squares (MLS) (Pauly, Keiser et al. 2003) © 2003 ACM: (a) a set of points (black dots) is turned into an implicit inside–outside function (black curve); (b) the signed distance to the nearest oriented point can serve as an approximation to the inside–outside distance; (c) a set of oriented points with variable sampling density representing a 3D surface (head model); (d) local estimate of sampling density, which is used in the moving least squares; (e) reconstructed continuous 3D surface.

Gross 2008), and kernel regression to better tolerate outliers (Oztireli, Guennebaud, and Gross 2008).

The survey by Berger, Tagliasacchi *et al.* (2017) discusses more recent work on reconstructing smooth complete surfaces from point clouds. The SurfelMeshing paper by Schöps, Sattler, and Pollefeys (2020) presents an RGB-D SLAM system based on a variable-resolution surfel representation that gets re-triangulated as more scans are integrated. Other recent approaches to 3D point clouds that use deep learning, mentioned previously in Section 5.5.1, are discussed in the survey by Guo, Wang *et al.* (2020). Even more recent algorithms to estimate better normals in 3D models are presented in Ben-Shabat and Gould (2020) and Zhu and Smith (2020).

13.5 Volumetric representations

A third alternative for modeling 3D surfaces is to construct 3D volumetric inside–outside functions. We have already seen examples of this in Section 12.7.2, where we looked at voxel coloring (Seitz and Dyer 1999), space carving (Kutulakos and Seitz 2000), and level set (Pons, Keriven, and Faugeras 2007) techniques for stereo matching, and Section 12.7.3, where we discussed using binary silhouette images to reconstruct volumes.

In this section, we look at continuous *implicit* (inside–outside) functions to represent 3D shape.

13.5.1 Implicit surfaces and level sets

While polyhedral and voxel-based representations can represent three-dimensional shapes to an arbitrary precision, they lack some of the intrinsic smoothness properties available with continuous implicit surfaces, which use an *indicator function* (or *characteristic function*) $F(x, y, z)$ to indicate which 3D points are inside $F(x, y, z) < 0$ or outside $F(x, y, z) > 0$ the object.

An early example of using implicit functions to model 3D objects in computer vision were superquadrics (Pentland 1986; Solina and Bajcsy 1990; Waithé and Ferrie 1991; Leonardis, Jaklič, and Solina 1997). To model a wider variety of shapes, superquadrics are usually combined with either rigid or non-rigid deformations (Terzopoulos and Metaxas 1991; Metaxas and Terzopoulos 2002). Superquadric models can either be fitted to range data or used directly for stereo matching.

A different kind of implicit shape model can be constructed by defining a *signed distance function* over a regular three-dimensional grid, optionally using an octree spline to represent this function more coarsely away from its surface (zero-set) (Lavallée and Szeliski 1995; Szeliski and Lavallée 1996; Frisken, Perry *et al.* 2000; Ohtake, Belyaev *et al.* 2003). We have already seen examples of signed distance functions being used to represent distance transforms (Section 3.3.3), level sets for 2D contour fitting and tracking (Section 7.3.2), volumetric stereo (Section 12.7.2), range data merging (Section 13.2.1), and point-based modeling (Section 13.4). The advantage of representing such functions directly on a grid is that it is quick and easy to look up distance function values for any (x, y, z) location and also easy to extract the isosurface using the marching cubes algorithm (Lorensen and Cline 1987). The work of Ohtake, Belyaev *et al.* (2003) is particularly notable, as it allows for several distance functions to be used simultaneously and then combined locally to produce sharp features such as creases.

Poisson surface reconstruction (Kazhdan, Bolitho, and Hoppe 2006; Kazhdan and Hoppe 2013) uses a closely related volumetric function, namely a smoothed 0/1 inside–outside (characteristic or occupancy) function, which can be thought of as a clipped signed distance function. The gradients for this function are set to lie along oriented surface normals near known surface points and 0 elsewhere. The function itself is represented using a quadratic tensor-product B-spline over an octree, which provides a compact representation with larger cells away from the surface or in regions of lower point density, and also admits the efficient solution of the related Poisson equations (4.24–4.27), e.g., Section 8.4.4 and Pérez, Gangnet, and Blake (2003).

It is also possible to replace the quadratic penalties used in the Poisson equations with L_1 (total variation) constraints and still obtain a convex optimization problem, which can be

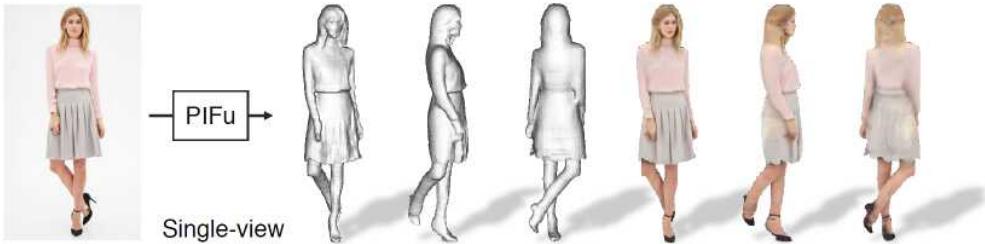


Figure 13.18 A Pixel-aligned Implicit Function (PIFu) network can recover a high-resolution 3D textured model of a clothed human from a single input image (Saito, Huang et al. 2019) © 2019 IEEE.

solved using either continuous (Zach, Pock, and Bischof 2007b; Zach 2008) or discrete graph cut (Lempitsky and Boykov 2007) techniques.

Signed distance functions also play an integral role in level-set evolution equations (Sections 7.3.2 and 12.7.2), where the values of distance transforms on the mesh are updated as the surface evolves to fit multi-view stereo photoconsistency measures (Faugeras and Keriven 1998).

As with many other areas of computer vision, deep neural networks have started being applied to the construction and modeling of volumetric object representations. Some neural networks construct 3D surface or volumetric occupancy grid models from single images (Choy, Xu *et al.* 2016; Tatarchenko, Dosovitskiy, and Brox 2017; Groueix, Fisher *et al.* 2018; Richter and Roth 2018), although more recent experiments suggest that these networks may just be recognizing the general object category and doing a small amount of fitting (Tatarchenko, Richter *et al.* 2019). DeepSDFs (Park, Florence *et al.* 2019), IM-NET (Chen and Zhang 2019), Occupancy Networks (Mescheder, Oechsle *et al.* 2019), Deep Implicit Surface (DISN) networks (Xu, Wang *et al.* 2019), and UCLID-Net (Guillard, Remelli, and Fua 2020) train networks to transform continuous (x, y, z) inputs into signed distance or $[0, 1]$ occupancy values and sometimes combine convolutional image encoders with MLPs to represent color and surface details (Oechsle, Mescheder *et al.* 2019), while MeshSDF can continuously transform SDFs into deformable meshes (Remelli, Lukoianov *et al.* 2020). All of these networks use latent codes to represent individual instances from a generic class (e.g., car or chair) from the ShapeNet dataset (Chang, Funkhouser *et al.* 2015), although they use the codes in a different part of the network (either in the input or through conditional batch normalization). This allows them to reconstruct 3D models from just a single image.

Pixel-aligned Implicit function (PIFu) networks combine fully convolutional image features with neural implicit functions to better preserve local shape and color details (Saito,

Huang *et al.* 2019; Saito, Simon *et al.* 2020). They are trained specifically on clothed humans and can hallucinate full 3D models from just a single color image (Figure 13.18). Neural Radiance Fields (NeRF) extend this to also use pixel ray directions as inputs and also output continuous valued opacities and radiance values, enabling ray-traced rendering of shiny 3D models constructed from multiple input images (Mildenhall, Srinivasan *et al.* 2020). This representation is related to Lumigraphs and Surface Light Fields, which we study in Section 14.3. Both of these systems are examples of *neural rendering* approaches to generating photorealistic novel views, which we discuss in more detail in Section 14.6.

To deal with larger (e.g., building-scale) scenes, Convolutional Occupancy Networks (Peng, Niemeyer *et al.* 2020) first retrieve local features from a 2D, multiplane, or 3D grid, and then use a trained MLP (fully connected network) to decode these into local occupancy volumes. Instead of modeling a complete 3D scene, Local Implicit Grid Representations (Jiang, Sud *et al.* 2020) model small local sub-volumes, allowing them to be used as a kind of prior for other shape reconstruction methods.

13.6 Model-based reconstruction

When we know something ahead of time about the objects we are trying to model, we can construct more detailed and reliable 3D models using specialized techniques and representations. For example, architecture is usually made up of large planar regions and other parametric forms (such as surfaces of revolution), usually oriented perpendicular to gravity and to each other (Section 13.6.1). Heads and faces can be represented using low-dimensional, non-rigid shape models, because the variability in shape and appearance of human faces, while extremely large, is still bounded (Section 13.6.2). Human bodies or parts, such as hands, form highly articulated structures, which can be represented using kinematic chains of piecewise rigid skeletal elements linked by joints (Section 13.6.4).

In this section, we highlight some of the main ideas, representations, and modeling algorithms used for these three cases. Additional details and references can be found in specialized conferences and workshops devoted to these topics, e.g., the International Conference on 3D Vision (3DV) and the IEEE International Conference on Automatic Face and Gesture Recognition (FG).

13.6.1 Architecture

Architectural modeling, especially from aerial photography, has been one of the longest studied problems in both photogrammetry and computer vision (Walker and Herman 1988). In the

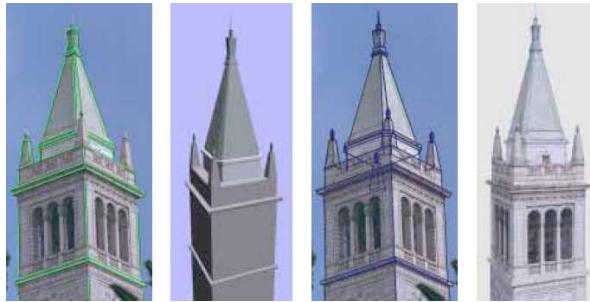


Figure 13.19 *Interactive architectural modeling using the Façade system (Debevec, Taylor, and Malik 1996) © 1996 ACM:* (a) *input image with user-drawn edges shown in green;* (b) *shaded 3D solid model;* (c) *geometric primitives overlaid onto the input image;* (d) *final view-dependent, texture-mapped 3D model.*

last two decades, the development of reliable image-based modeling techniques, as well as the prevalence of digital cameras and 3D computer games, has led to widespread deployment of such systems.

The work by Debevec, Taylor, and Malik (1996) was one of the earliest hybrid geometry-and image-based modeling and rendering systems. Their Façade system combines an interactive image-guided geometric modeling tool with model-based (local plane plus parallax) stereo matching and view-dependent texture mapping. During the interactive photogrammetric modeling phase, the user selects block elements and aligns their edges with visible edges in the input images (Figure 13.19a). The system then automatically computes the dimensions and locations of the blocks along with the camera positions using constrained optimization (Figure 13.19b–c). This approach is intrinsically more reliable than general feature-based structure from motion, because it exploits the strong geometry available in the block primitives. Related work by Becker and Bove (1995), Horry, Anjyo, and Arai (1997), Criminisi, Reid, and Zisserman (2000), and Holynski, Geraghty *et al.* (2020) exploits similar information available from vanishing points. In the interactive, image-based modeling system of Sinha, Steedly *et al.* (2008), vanishing point directions are used to guide the user drawing of polygons, which are then automatically fitted to sparse 3D points recovered using structure from motion.

Once the rough geometry has been estimated, more detailed offset maps can be computed for each planar face using a local plane sweep, which Debevec, Taylor, and Malik (1996) call *model-based stereo*. Finally, during rendering, images from different viewpoints are warped and blended together as the camera moves around the scene, using a process (related to light

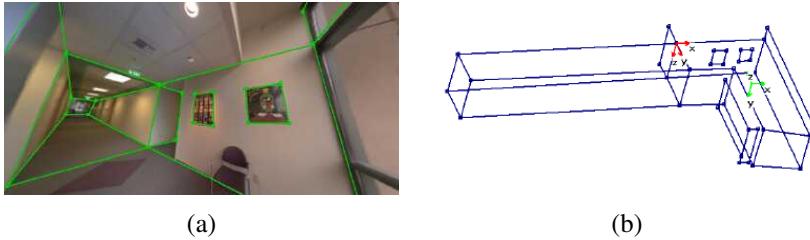


Figure 13.20 *Interactive 3D modeling from panoramas (Shum, Han, and Szeliski 1998)*
 © 1998 IEEE: (a) wide-angle view of a panorama with user-drawn vertical and horizontal (axis-aligned) lines; (b) single-view reconstruction of the corridors.

field and Lumigraph rendering; see Section 14.3) called *view-dependent texture mapping* (Figure 13.19d).

For interior modeling, instead of working with single pictures, it is more useful to work with panoramas, as you can see larger extents of walls and other structures. The 3D modeling system developed by Shum, Han, and Szeliski (1998) first constructs calibrated panoramas from multiple images (Section 11.4.2) and then has the user draw vertical and horizontal lines in the image to demarcate the boundaries of planar regions. The lines are initially used to establish an absolute rotation for each panorama and are later used (along with the inferred vertices and planes) to optimize the 3D structure, which can be recovered up to scale from one or more images (Figure 13.20). Recent advances in deep networks now make it possible to both automatically infer the lines and their junctions (Huang, Wang *et al.* 2018; Zhang, Li *et al.* 2019) and to build complete 3D wireframe models (Zhou, Qi, and Ma 2019; Zhou, Qi *et al.* 2019b). 360° high dynamic range panoramas can also be used for outdoor modeling, because they provide highly reliable estimates of relative camera orientations as well as vanishing point directions (Antone and Teller 2002; Teller, Antone *et al.* 2003).

While earlier image-based modeling systems required some user authoring, Werner and Zisserman (2002) present a fully automated line-based reconstruction system. As described in Section 11.4.8, they first detect lines and vanishing points and use them to calibrate the camera; then they establish line correspondences using both appearance matching and trifocal tensors, which enables them to reconstruct families of 3D line segments. They then generate plane hypotheses, using both co-planar 3D lines and a plane sweep (Section 12.1.2) based on cross-correlation scores evaluated at interest points. Intersections of planes are used to determine the extent of each plane, i.e., an initial coarse geometry, which is then refined with the addition of rectangular or wedge-shaped indentations and extrusions. Note that when top-down maps of the buildings being modeled are available, these can be used to further

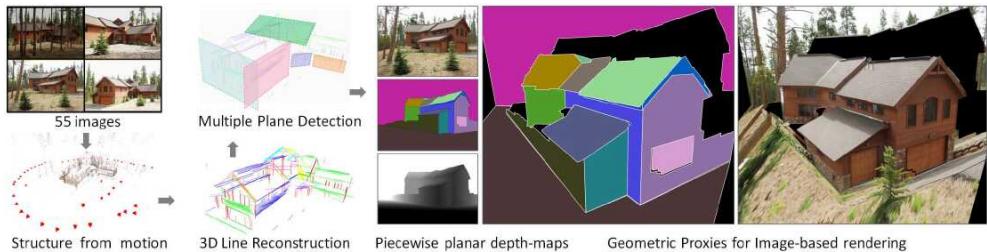


Figure 13.21 Automated architectural reconstruction using 3D lines and planes (Sinha, Steedly, and Szeliski 2009) © 2009 IEEE.

constrain the 3D modeling process (Robertson and Cipolla 2002, 2009). The idea of using matched 3D lines for estimating vanishing point directions and dominant planes is used in a number of fully automated image-based architectural modeling systems (Zebedin, Bauer *et al.* 2008; Mičušk and Košecká 2009; Furukawa, Curless *et al.* 2009b; Sinha, Steedly, and Szeliski 2009; Holynski, Geraghty *et al.* 2020) as well as SLAM systems (Zhou, Zou *et al.* 2015; Li, Yao *et al.* 2018; Yang and Scherer 2019). Figure 13.21 shows some of the processing stages in the system developed by Sinha, Steedly, and Szeliski (2009).

Another common characteristic of architecture is the repeated use of primitives such as windows, doors, and colonnades. Architectural modeling systems can be designed to search for such repeated elements and to use them as part of the structure inference process (Dick, Torr, and Cipolla 2004; Mueller, Zeng *et al.* 2007; Schindler, Krishnamurthy *et al.* 2008; Pauly, Mitra *et al.* 2008; Sinha, Steedly *et al.* 2008). The combination of structured elements such as parallel lines, junctions, and rectangles with full axis-aligned 3D models for the modeling of architectural environments has recently been called *holistic 3D reconstruction*. More details can be found in the recent tutorial by Zhou, Furukawa, and Ma (2019), workshop (Zhou, Furukawa *et al.* 2020), and state-of-the-art report by Pintore, Mura *et al.* (2020).

The combination of all these techniques now makes it possible to reconstruct the structure of large 3D scenes (Zhu and Kanade 2008). For example, the *Urbanscan* system of Pollefeys, Nistér *et al.* (2008) reconstructs texture-mapped 3D models of city streets from videos acquired with a GPS-equipped vehicle. To obtain real-time performance, they use both optimized online structure-from-motion algorithms, as well as GPU implementations of plane-sweep stereo aligned to dominant planes and depth map fusion. Cornelis, Leibe *et al.* (2008) present a related system that also uses plane-sweep stereo (aligned to vertical building façades) combined with object recognition and segmentation for vehicles. Mičušk and Košecká (2009) build on these results using omni-directional images and superpixel-based stereo matching along dominant plane orientations. Reconstruction directly from active range

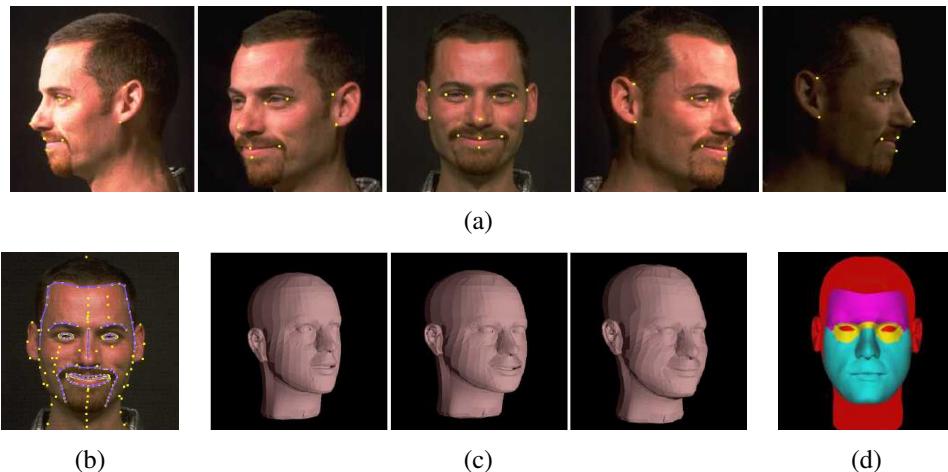


Figure 13.22 3D model fitting to a collection of images: (Pighin, Hecker et al. 1998) © 1998 ACM: (a) set of five input images along with user-selected keypoints; (b) the complete set of keypoints and curves; (c) three meshes—the original, adapted after 13 keypoints, and after an additional 99 keypoints; (d) the partition of the image into separately animatable regions.

scanning data combined with color imagery that has been compensated for exposure and lighting variations is also possible (Chen and Chen 2008; Stamos, Liu et al. 2008; Troccoli and Allen 2008).

Numerous photogrammetric reconstruction systems that produce detailed texture-mapped 3D models have been developed based on these computer vision techniques.¹⁰ Examples of commercial software that can be used to reconstruct large-scale 3D models from aerial drone and ground level photography include Pix4D,¹¹ Metashape,¹² and RealityCapture.¹³ Another example is Occipital's Canvas mobile phone app¹⁴ (Stein 2020), which appears to use a combination of photogrammetry (3D point and line matching and reconstruction, as discussed above) and depth map fusion.

13.6.2 Facial modeling and tracking

Another area in which specialized shape and appearance models are extremely helpful is in the modeling of heads and faces. Even though the appearance of people seems at first glance to be infinitely variable, the actual shape of a person’s head and face can be described reasonably well using a few dozen parameters (Pighin, Hecker *et al.* 1998; Guenter, Grimm *et al.* 1998; DeCarlo, Metaxas, and Stone 1998; Blanz and Vetter 1999; Shan, Liu, and Zhang 2001; Zollhöfer, Thies *et al.* 2018; Egger, Smith *et al.* 2020).

Figure 13.22 shows an example of an image-based modeling system, where user-specified keypoints in several images are used to fit a generic head model to a person’s face. As you can see in Figure 13.22c, after specifying just over 100 keypoints, the shape of the face has become quite adapted and recognizable. Extracting a texture map from the original images and then applying it to the head model results in an animatable model with striking visual fidelity (Figure 13.23a).

A more powerful system can be built by applying *principal component analysis* (PCA) to a collection of 3D scanned faces, which is a topic we discuss in Section 13.6.3. As you can see in Figure 13.25, it is then possible to fit morphable 3D models to single images and to use such models for a variety of animation and visual effects (Blanz and Vetter 1999; Egger, Smith *et al.* 2020). It is also possible to design stereo matching algorithms that optimize directly for the head model parameters (Shan, Liu, and Zhang 2001; Kang and Jones 2002) or to use the output of real-time stereo with active illumination (Zhang, Snavely *et al.* 2004) (Figures 13.10 and 13.23b).

As the sophistication of 3D facial capture systems evolved, so did the detail and realism in the reconstructed models. Modern systems can capture (in real-time) not only surface details such as wrinkles and creases, but also accurate models of skin reflection, translucency, and sub-surface scattering (Debevec, Hawkins *et al.* 2000; Weyrich, Matusik *et al.* 2006; Golovinskiy, Matusik *et al.* 2006; Bickel, Botsch *et al.* 2007; Igarashi, Nishino, and Nayar 2007; Meka, Haene *et al.* 2019).

Once a 3D head model has been constructed, it can be used in a variety of applications, such as head tracking (Toyama 1998; Lepetit, Pilet, and Fua 2004; Matthews, Xiao, and Baker 2007), as shown in Figures 7.30 and face transfer, i.e., replacing one person’s face with another in a video (Bregler, Covell, and Slaney 1997; Vlasic, Brand *et al.* 2005). Additional applications include face beautification by warping face images toward a more attrac-

¹⁰<https://all3dp.com/1/best-photogrammetry-software>

¹¹<https://www.pix4d.com>

¹²<https://www.agisoft.com/>

¹³<https://www.capturingreality.com/>

¹⁴<https://canvas.io>

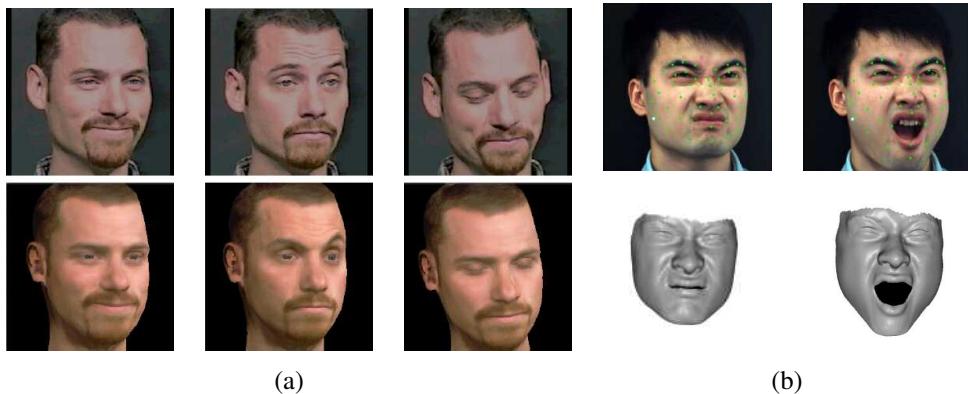


Figure 13.23 Head and expression tracking and re-animation using deformable 3D models. (a) Models fitted directly to five input video streams (Pighin, Szeliski, and Salesin 2002) © 2002 Springer: The bottom row shows the results of re-animating a synthetic texture-mapped 3D model with pose and expression parameters fitted to the input images in the top row. (b) Models fitted to frame-rate spacetime stereo surface models (Zhang, Snavely et al. 2004) © 2004 ACM: The top row shows the input images with synthetic green markers overlaid, while the bottom row shows the fitted 3D surface model.

tive “standard” (Leyvand, Cohen-Or *et al.* 2008), face de-identification for privacy protection (Gross, Sweeney *et al.* 2008), and face swapping (Bitouk, Kumar *et al.* 2008).

More recent applications of 3D head models include photorealistic avatars for video conferencing (Chu, Ma *et al.* 2020), 3D unwarping for better selfies (Fried, Shechtman *et al.* 2016; Zhao, Huang *et al.* 2019; Ma, Lin *et al.* 2020), and single image portrait relighting (Sun, Barron *et al.* 2019; Zhou, Hadap *et al.* 2019; Zhang, Barron *et al.* 2020), an example of which is shown in Figure 13.24. This last application is available as the Portrait Light feature in Google Photos.¹⁵ Additional applications can be found in the survey papers by Zollhöfer, Thies *et al.* (2018) and Egger, Smith *et al.* (2020).

13.6.3 Application: Facial animation

Perhaps the most widely used application of 3D head modeling is facial animation (Zollhöfer, Thies *et al.* 2018). Once a parameterized 3D model of the shape and appearance (surface texture) of a person’s head has been constructed, it can be used directly to track a person’s facial motions (Figure 13.23a) and to animate a different character with these same motions

¹⁵<https://blog.google/products/photos/new-helpful-editor>



Figure 13.24 *Portrait shadow removal and manipulation (Zhang, Barron et al. 2020) © 2020 ACM. The top row shows the original photographs and the bottom row the corresponding enhanced photographs after more flattering lighting has been simulated.*

and expressions (Pighin, Szeliski, and Salesin 2002).

An improved version of such a system can be constructed by first applying principal component analysis (PCA) to the space of possible head shapes and facial appearances. Blanz and Vetter (1999) describe a system where they first capture a set of 200 colored range scans of faces (Figure 13.25a), which can be represented as a large collection of (X, Y, Z, R, G, B) samples (vertices).¹⁶ For 3D morphing to be meaningful, corresponding vertices in different people's scans must first be put into correspondence (Pighin, Hecker *et al.* 1998). Once this is done, PCA can be applied to more naturally parameterize the 3D morphable model. The flexibility of this model can be increased by performing separate analyses in different subregions, such as the eyes, nose, and mouth, just as in modular eigenspaces (Moghaddam and Pentland 1997).

After computing a subspace representation, different directions in this space can be associated with different characteristics such as gender, facial expressions, or facial features (Figure 13.25a). As in the work of Rowland and Perrett (1995), faces can be turned into caricatures by exaggerating their displacement from the mean image.

3D morphable models can be fitted to a single image using gradient descent on the error between the input image and the re-synthesized model image, after an initial manual placement of the model in an approximately correct pose, scale, and location (Figures 13.25b–c). The efficiency of this fitting process can be increased using inverse compositional image

¹⁶A cylindrical coordinate system provides a natural two-dimensional embedding for this collection, but such an embedding is not necessary to perform PCA.



Figure 13.25 3D morphable face model (Blanz and Vetter 1999) © 1999 ACM: (a) original 3D face model with the addition of shape and texture variations in specific directions: deviation from the mean (caricature), gender, expression, weight, and nose shape; (b) a 3D morphable model is fitted to a single image, after which its weight or expression can be manipulated; (c) another example of a 3D reconstruction along with a different set of 3D manipulations, such as lighting and pose change.

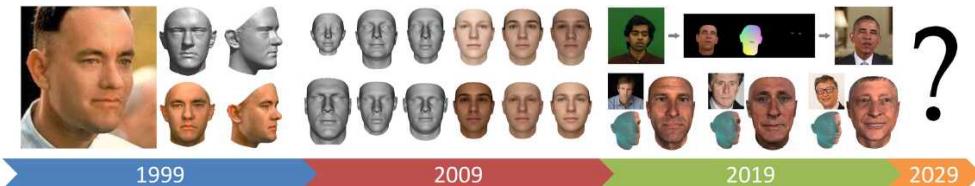


Figure 13.26 A timeline of twenty years of 3D morphable head models (Egger, Smith et al. 2020) © 2020 ACM, including results from the original paper by Blanz and Vetter (1999), the first publicly available morphable model (Paysan, Knothe et al. 2009), facial re-enactment results (Kim, Garrido et al. 2018), and GAN-based models (Gecer, Ploumpis et al. 2019).

alignment (Baker and Matthews 2004) as described by Romdhani and Vetter (2003).

The resulting texture-mapped 3D model can then be modified to produce a variety of visual effects, including changing a person’s weight or expression, or three-dimensional effects such as re-lighting or 3D video-based animation (Section 14.5.1). Such models can also be used for video compression, e.g., by only transmitting a small number of facial expression and pose parameters to drive a synthetic avatar (Eisert, Wiegand, and Girod 2000; Gao, Chen et al. 2003; Lombardi, Saragih et al. 2018; Wei, Saragih et al. 2019) or to bring a still portrait image to life (Averbuch-Elor, Cohen-Or et al. 2017). The survey paper on 3D morphable face models by Egger, Smith et al. (2020) (Figure 13.26) discusses additional research and applications in this area.

3D facial animation is often matched to the performance of an actor, in what is known as *performance-driven animation* (Section 7.1.6) (Williams 1990). Traditional performance-driven animation systems use marker-based motion capture (Ma, Jones et al. 2008), while some newer systems use depth cameras or regular video to control the animation (Buck, Finkelstein et al. 2000; Pighin, Szeliski, and Salesin 2002; Zhang, Snavely et al. 2004; Vlasic, Brand et al. 2005; Weise, Bouaziz et al. 2011; Thies, Zollhofer et al. 2016; Thies, Zollhöfer et al. 2018).

An example of the latter approach is the system developed for the film *The Curious Case of Benjamin Button*, in which Digital Domain used the CONTOUR system from Mova¹⁷ to capture actor Brad Pitt’s facial motions and expressions (Roble and Zafar 2009). CONTOUR uses a combination of phosphorescent paint and multiple high-resolution video cameras to capture real-time 3D range scans of the actor. These 3D models were then translated into Facial Action Coding System (FACS) shape and expression parameters (Ekman and Friesen 1978) to drive a different (older) synthetically animated computer-generated imagery (CGI)

¹⁷<http://www.mova.com>.

character. More recent examples of performance-driven facial animation can be found in the state of the art report by Zollhöfer, Thies *et al.* (2018).

13.6.4 Human body modeling and tracking

The topics of tracking humans, modeling their shape and appearance, and recognizing their activities, are some of the most actively studied areas of computer vision. Annual conferences¹⁸ and special journal issues (Hilton, Fua, and Ronfard 2006) are devoted to this subject, and two surveys (Forsyth, Arikān *et al.* 2006; Moeslund, Hilton, and Krüger 2006) each list over 400 papers devoted to these topics.¹⁹ The HumanEva database of articulated human motions contains multi-view video sequences of human actions along with corresponding motion capture data, evaluation code, and a reference 3D tracker based on particle filtering. The companion paper by Sigal, Balan, and Black (2010) not only describes the database and evaluation but also has a nice survey of important work in this field. The more recent MPI FAUST dataset (Bogo, Romero *et al.* 2014) has 300 real, high-resolution human scans with automatically computed ground-truth correspondences, while the even newer AMASS dataset (Mahmood, Ghorbani *et al.* 2019) has more than 40 hours of motion data, spanning over 300 subjects and 11,000 motions.²⁰

Given the breadth of this area, it is difficult to categorize all of this research, especially as different techniques usually build on each other. Moeslund, Hilton, and Krüger (2006) divide their survey into initialization, tracking (which includes background modeling and segmentation), pose estimation, and action (activity) recognition. Forsyth, Arikān *et al.* (2006) divide their survey into sections on tracking (background subtraction, deformable templates, flow, and probabilistic models), recovering 3D pose from 2D observations, and data association and body parts. They also include a section on motion synthesis, which is more widely studied in computer graphics (Arikān and Forsyth 2002; Kovar, Gleicher, and Pighin 2002; Lee, Chai *et al.* 2002; Li, Wang, and Shum 2002; Pullen and Bregler 2002): see Section 14.5.2. Another potential taxonomy for work in this field would be along the lines of whether 2D or 3D (or multi-view) images are used as input and whether 2D or 3D kinematic models are used.

In this section, we briefly review some of the more seminal and widely cited papers in the areas of background subtraction, initialization and detection, tracking with flow, 3D kinematic

¹⁸International Conference on Automatic Face and Gesture Recognition (FG) and IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG).

¹⁹Older surveys include those by Gavrila (1999) and Moeslund and Granum (2001). Some surveys on gesture recognition, which we do not cover in this book, include those by Pavlović, Sharma, and Huang (1997) and Yang, Ahuja, and Tabb (2002).

²⁰Additional datasets from the MPI Perceiving Systems group can be found at <https://ps.is.mpg.de/code>.

models, probabilistic models, adaptive shape modeling, and activity recognition. We refer the reader to the previously mentioned surveys for other topics and more details.

Background subtraction. One of the first steps in many human tracking systems is to model the background to extract the moving foreground objects (silhouettes) corresponding to people. Toyama, Krumm *et al.* (1999) review several *difference matting* and *background maintenance* (modeling) techniques and provide a good introduction to this topic. Stauffer and Grimson (1999) describe some techniques based on mixture models, while Sidenbladh and Black (2003) develop a more comprehensive treatment, which models not only the background image statistics but also the appearance of the foreground objects, e.g., their edge and motion (frame difference) statistics. More recent techniques for video background matting, such as those of Sengupta, Jayaram *et al.* (2020) and Lin, Ryabtsev *et al.* (2021) are discussed in Section 10.4.5 on video matting.

Once silhouettes have been extracted from one or more cameras, they can then be modeled using deformable templates or other contour models (Baumberg and Hogg 1996; Wren, Azarbayejani *et al.* 1997). Tracking such silhouettes over time supports the analysis of multiple people moving around a scene, including building shape and appearance models and detecting if they are carrying objects (Haritaoglu, Harwood, and Davis 2000; Mittal and Davis 2003; Dimitrijevic, Lepetit, and Fua 2006).

Initialization and detection. To track people in a fully automated manner, it is necessary to first detect (or re-acquire) their presence in individual video frames. This topic is closely related to *pedestrian detection*, which is often considered as a kind of object recognition (Mori, Ren *et al.* 2004; Felzenszwalb and Huttenlocher 2005; Felzenszwalb, McAllester, and Ramanan 2008; Dollár, Wojek *et al.* 2012; Dollár, Appel *et al.* 2014; Sermanet, Kavukcuoglu *et al.* 2013; Ouyang and Wang 2013; Tian, Luo *et al.* 2015; Zhang, Lin *et al.* 2016), and is therefore treated in more depth in Section 6.3.2. Additional techniques for initializing 3D trackers based on 2D images include those described by Howe, Leventon, and Freeman (2000), Rosales and Sclaroff (2000), Shakhnarovich, Viola, and Darrell (2003), Sminchisescu, Kanaujia *et al.* (2005), Agarwal and Triggs (2006), Lee and Cohen (2006), Sigal and Black (2006b), and Stenger, Thayananthan *et al.* (2006).

Single-frame human detection and pose estimation algorithms can be used by themselves to perform tracking (Ramanan, Forsyth, and Zisserman 2005; Rogez, Rihan *et al.* 2008; Bourdev and Malik 2009; Güler, Neverova, and Kokkinos 2018; Cao, Hidalgo *et al.* 2019), as described in Section 6.3.2 (Figure 6.25) and Section 6.4.5 (Figure 6.42–6.43). They are often combined, however, with frame-to-frame tracking techniques to provide better reliability

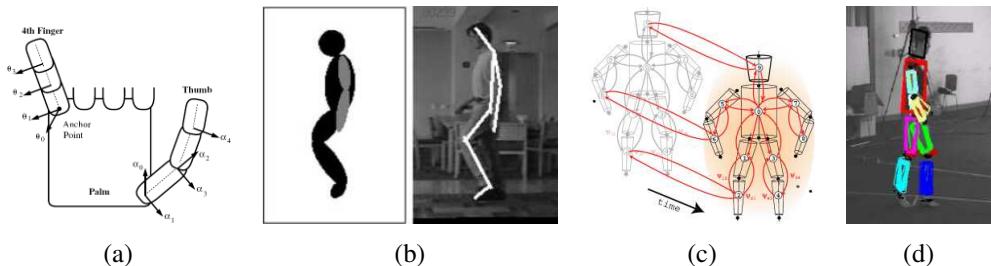


Figure 13.27 Tracking 3D human motion: (a) kinematic chain model for a human hand (Rehg, Morris, and Kanade 2003) © 2003, reprinted by permission of SAGE; (b) tracking a kinematic chain blob model in a video sequence (Bregler, Malik, and Pullen 2004) © 2004 Springer; (c–d) probabilistic loose-limbed collection of body parts (Sigal, Bhatia et al. 2004) © 2004 IEEE.

(Fossati, Dimitrijevic *et al.* 2007; Andriluka, Roth, and Schiele 2008; Ferrari, Marin-Jimenez, and Zisserman 2008).

Tracking with flow. The tracking of people and their pose from frame to frame can be enhanced by computing optical flow or matching the appearance of their limbs from one frame to another. For example, the *cardboard people* model of Ju, Black, and Yacoob (1996) models the appearance of each leg portion (upper and lower) as a moving rectangle, and uses optical flow to estimate their location in each subsequent frame. Cham and Rehg (1999) and Sidenbladh, Black, and Fleet (2000) track limbs using optical flow and templates, along with techniques for dealing with multiple hypotheses and uncertainty. Bregler, Malik, and Pullen (2004) use a full 3D model of limb and body motion, as described below. It is also possible to match the estimated motion field itself to some prototypes in order to identify the particular phase of a running motion or to match two low-resolution video portions to perform video replacement (Efros, Berg *et al.* 2003). Flow-based tracking can also be used to track non-rigidly deforming objects such as T-shirts (White, Crane, and Forsyth 2007; Pilet, Lepetit, and Fua 2008; Furukawa and Ponce 2008; Salzmann and Fua 2010; Božić, Zollhöfer *et al.* 2020; Božić, Palafox *et al.* 2020, 2021). It is also possible to use inter-frame motion to estimate an evolving textured 3D mesh model of a moving person (de Aguiar, Stoll *et al.* 2008).

3D kinematic models. The effectiveness of human modeling and tracking can be greatly enhanced using a more accurate 3D model of a person’s shape and motion. Underlying such representations, which are ubiquitous in 3D computer animation in games and special effects,

is a *kinematic model* or *kinematic chain*, which specifies the length of each limb in a skeleton as well as the 2D or 3D rotation angles between the limbs or segments (Figure 13.27a–b). Inferring the values of the joint angles from the locations of the visible surface points is called *inverse kinematics* (IK) and is widely studied in computer graphics.

Figure 13.27a shows the kinematic model for a human hand used by Rehg, Morris, and Kanade (2003) to track hand motion in a video. As you can see, the attachment points between the fingers and the thumb have two degrees of freedom, while the finger joints themselves have only one. Using this kind of model can greatly enhance the ability of an edge-based tracker to cope with rapid motion, ambiguities in 3D pose, and partial occlusions.

One of the biggest advances in reliable real-time hand tracking and modeling was the introduction of the Kinect consumer RGB-D camera (Sharp, Keskin *et al.* 2015; Taylor, Bordeau *et al.* 2016). Since then, regular RGB tracking and modeling has also improved significantly, with newer techniques using neural networks for reliability and speed (Zimmermann and Brox 2017; Mueller, Bernard *et al.* 2018; Hasson, Varol *et al.* 2019; Shan, Geng *et al.* 2020; Moon, Shiratori, and Lee 2020; Moon, Yu *et al.* 2020; Spurr, Iqbal *et al.* 2020; Taheri, Ghorbani *et al.* 2020). Several systems also combine body and hand tracking to more accurately capture human expressions and activities (Romero, Tzionas, and Black 2017; Joo, Simon, and Sheikh 2018; Pavlakos, Choutas *et al.* 2019; Rong, Shiratori, and Joo 2020).

In addition to hands, kinematic chain models are even more widely used for whole body modeling and tracking (O'Rourke and Badler 1980; Hogg 1983; Rohr 1994). One popular approach is to associate an ellipsoid or superquadric with each rigid limb in the kinematic model, as shown in Figure 13.27b. This model can then be fitted to each frame in one or more video streams either by matching silhouettes extracted from known backgrounds or by matching and tracking the locations of occluding edges (Gavrila and Davis 1996; Kakadiaris and Metaxas 2000; Bregler, Malik, and Pullen 2004; Kehl and Van Gool 2006).

One of the big breakthroughs in real-time skeletal tracking was the introduction of the Kinect consumer depth camera for interactive video game control (Shotton, Fitzgibbon *et al.* 2011; Taylor, Shotton *et al.* 2012; Shotton, Girshick *et al.* 2013) as shown in Figure 13.28. In the current landscape of skeletal tracking, some techniques use 2D models coupled to 2D measurements, some use 3D measurements (range data or multi-view video) with 3D models (Baak, Mueller *et al.* 2011), and some use monocular video to infer and track 3D models directly (Mehta, Sridhar *et al.* 2017; Habermann, Xu *et al.* 2019).

It is also possible to use temporal models to improve the tracking of periodic motions, such as walking, by analyzing the joint angles as functions of time (Polana and Nelson 1997; Seitz and Dyer 1997; Cutler and Davis 2000). The generality and applicability of such techniques can be improved by learning typical motion patterns using principal component anal-

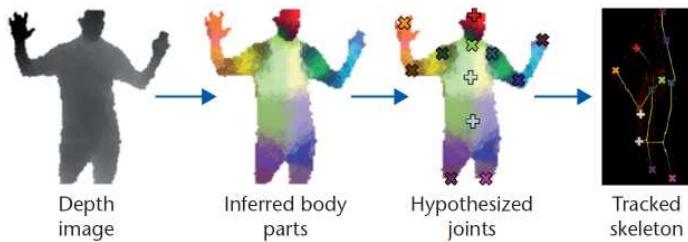


Figure 13.28 The Kinect skeletal tracking pipeline, which consists of per-pixel body-part classification, body joint hypotheses, and then mapping to a skeleton using temporal continuity and prior knowledge (Shotton, Girshick et al. 2013). This figure is taken from (Zhang 2012) © 2012 IEEE.

ysis (Sidenbladh, Black, and Fleet 2000; Urtasun, Fleet, and Fua 2006).

Probabilistic models. Because tracking can be such a difficult task, sophisticated probabilistic inference techniques are often used to estimate the likely states of the person being tracked. One popular approach, called *particle filtering* (Isard and Blake 1998), was originally developed for tracking the outlines of people and hands, as described in Section 7.3.1. It was subsequently applied to whole-body tracking (Deutscher, Blake, and Reid 2000; Sidenbladh, Black, and Fleet 2000; Deutscher and Reid 2005) and continues to be used in modern trackers (Ong, Micilotta et al. 2006). Alternative approaches to handling the uncertainty inherent in tracking include multiple hypothesis tracking (Cham and Rehg 1999) and inflated covariances (Sminchisescu and Triggs 2001).

Figure 13.27c–d shows an example of a sophisticated spatio-temporal probabilistic graphical model called *loose-limbed people*, which models not only the geometric relationship between various limbs, but also their likely temporal dynamics (Sigal, Bhatia et al. 2004). The conditional probabilities relating various limbs and time instances are learned from training data, and particle filtering is used to perform the final pose inference.

Adaptive shape modeling. Another essential component of whole body modeling and tracking is the fitting of parameterized shape models to visual data. As we saw in Section 13.6.3 (Figure 13.25), the availability of large numbers of registered 3D range scans can be used to create *morphable models* of shape and appearance (Allen, Curless, and Popović 2003). Building on this work, Anguelov, Srinivasan et al. (2005) develop a sophisticated system called SCAPE (Shape Completion and Animation for PEople), which first acquires a large number of range scans of different people in varied poses, and then registers these

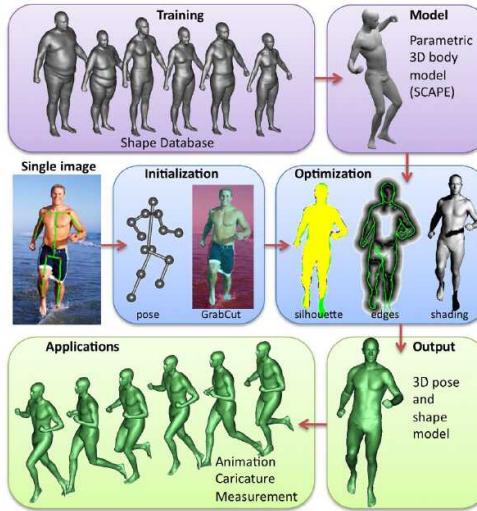


Figure 13.29 Estimating human shape and pose from a single image using a parametric 3D model (Guan, Weiss et al. 2009) © 2009 IEEE.

scans using semi-automated marker placement. The registered datasets are used to model the variation in shape as a function of personal characteristics and skeletal pose, e.g., the bulging of muscles as certain joints are flexed (Figure 13.29, top row). The resulting system can then be used for *shape completion*, i.e., the recovery of a full 3D mesh model from a small number of captured markers, by finding the best model parameters in both shape and pose space that fit the measured data.

Because it is constructed completely from scans of people in close-fitting clothing and uses a parametric shape model, the SCAPE system cannot cope with people wearing loose-fitting clothing. Bălan and Black (2008) overcome this limitation by estimating the body shape that fits within the visual hull of the same person observed in multiple poses, while Vlasic, Baran et al. (2008) adapt an initial surface mesh fitted with a parametric shape model to better match the visual hull.

While the preceding body fitting and pose estimation systems use multiple views to estimate body shape, Guan, Weiss et al. (2009) fit a human shape and pose model to a single image of a person on a natural background. Manual initialization is used to estimate a rough pose (skeleton) and height model, and this is then used to segment the person's outline using the Grab Cut segmentation algorithm (Section 4.3.2). The shape and pose estimate are then refined using a combination of silhouette edge cues and shading information (Figure 13.29). The resulting 3D model can be used to create novel animations.

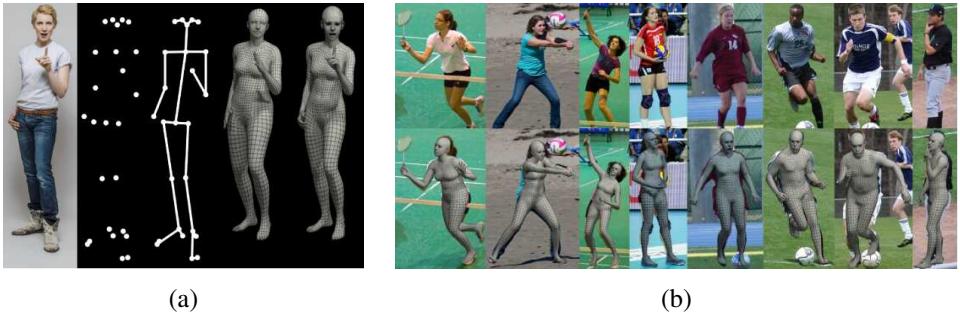


Figure 13.30 Whole body, expression, and gesture fitting from a single image using the SMPL-X model from Pavlakos, Choutas et al. (2019) © 2019 IEEE: (a) estimating the major joints, skeleton, SMPL, and SMPL-X models from a single image; (b) qualitative results of SMPL-X for some in-the-wild images.

While some of the original work on 3D body and pose fitting was done using the SCAPE and BlendSCAPE (Hirshberg, Loper *et al.* 2012) models, the Skinned Multi-Person Linear model (SMPL) developed by Loper, Mahmood *et al.* (2015) introduced a skinned vertex-based model that accurately represents a wide variety of body shapes in natural human poses. The model consists of a rest pose template, pose-dependent blend shapes, and identity-dependent blend shapes, and is built by training on a large collection of aligned 3D human scans. Bogo, Kanazawa *et al.* (2016) show how the parameters of this 3D model can be estimated from just a single image using their SMPLfy method.

In subsequent work Romero, Tzionas, and Black (2017) extend this model by adding a hand Model with Articulated and Non-rigid defOrmations (MANO). Joo, Simon, and Sheikh (2018) stitch together the SMPL body model with a face and a hand model to create the 3D Frank and Adam models that can track multiple people in a social setting. And Pavlakos, Choutas *et al.* (2019) use thousands of 3D scans to train a new, unified, 3D model of the human body (SMPL-X) that extends SMPL with gender-specific models and includes fully articulated hands and an expressive face, as shown in Figure 13.30. They also replace the mixture of Gaussians prior in SMPL with a variational autoencoder (VAE) and develop a new VPoser prior trained on the large-scale AMASS motion capture dataset collected by Mahmood, Ghorbani *et al.* (2019).

In more recent work, Kocabas, Athanasiou, and Black (2020) introduce VIBE, a system for video inference of human body pose and shape that makes use of AMASS. Choutas, Pavlakos *et al.* (2020) develop a system they call ExPose (EXpressive POse and Shape rEgression), which directly regresses the body, face, and hands SMPL-X parameters from an

RGB image. The more recent STAR (Sparse Trained Articulated human body Regressor) model (Osman, Bolkart, and Black 2020), has many fewer parameters than SMPL and removes spurious long-range correlations between vertices. It also includes shape-dependent pose-corrective blend shapes that depend on both body pose and BMI and also models a much wider range of variation in the human population by training STAR with an additional 10,000 scans of male and female subjects. GHUM and GHUML (Xu, Bazavan *et al.* 2020) rely on non-linear shape spaces constructed from deep variational autoencoders for body and facial deformation and on normalizing flow representations for skeleton (body and hand) kinematics. Recent papers that continue to improve the accuracy and speed of single-image model fitting on the challenging 3D Poses in the Wild (3DPW) benchmark and dataset (von Marcard, Henschel *et al.* 2018) include Song, Chen, and Hilliges (2020), Joo, Neverova, and Vedaldi (2020), and Rong, Shiratori, and Joo (2020).

Activity recognition. The final widely studied topic in human modeling is motion, activity, and action recognition (Bobick 1997; Hu, Tan *et al.* 2004; Hilton, Fua, and Ronfard 2006). Examples of actions that are commonly recognized include walking and running, jumping, dancing, picking up objects, sitting down and standing up, and waving. Papers on these topics include Robertson and Reid (2006), Sminchisescu, Kanaujia, and Metaxas (2006), Weinland, Ronfard, and Boyer (2006), Yilmaz and Shah (2006), and Gorelick, Blank *et al.* (2007), as well as more recent video understanding papers such as the ones we covered in Section 6.5, e.g., Carreira and Zisserman (2017), Tran, Wang *et al.* (2018), Tran, Wang *et al.* (2019), Wu, Feichtenhofer *et al.* (2019), and Feichtenhofer, Fan *et al.* (2019).

13.7 Recovering texture maps and albedos

After a 3D model of an object or person has been acquired, the final step in modeling is usually to recover a *texture map* to describe the object’s surface appearance. This first requires establishing a parameterization for the (u, v) texture coordinates as a function of 3D surface position.²¹ One simple way to do this is to associate a separate texture map with each triangle (or pair of triangles). More space-efficient techniques involve unwrapping the surface onto one or more maps, e.g., using a subdivision mesh (Section 13.3.2) (Eck, DeRose *et al.* 1995) or a geometry image (Section 13.3.3) (Gu, Gortler, and Hoppe 2002).

Once the (u, v) coordinates for each triangle have been fixed, the perspective projection equations mapping from texture (u, v) to an image j ’s pixel (u_j, v_j) coordinates can be

²¹Although a few recent papers have directly constructed a mapping from (x, y, z) to color values (Saito, Huang *et al.* 2019; Saito, Simon *et al.* 2020; Mildenhall, Srinivasan *et al.* 2020)—see Section 14.6.

obtained by concatenating the affine $(u, v) \rightarrow (X, Y, Z)$ mapping with the perspective homography $(X, Y, Z) \rightarrow (u_j, v_j)$ (Szeliski and Shum 1997). The color values for the (u, v) texture map can then be re-sampled and stored, or the original image can itself be used as the texture source using projective texture mapping (OpenGL-ARB 1997).

The situation becomes more involved when more than one source image is available for appearance recovery, which is the usual case. One possibility is to use a *view-dependent texture map* (Section 14.1.1), in which a different source image (or combination of source images) is used for each polygonal face based on the angles between the virtual camera, the surface normals, and the source images (Debevec, Taylor, and Malik 1996; Pighin, Hecker *et al.* 1998). An alternative approach is to estimate a complete Surface Light Field for each surface point (Wood, Azuma *et al.* 2000), as described in Section 14.3.2.

In some situations, e.g., when using models in traditional 3D games, it is preferable to merge all of the source images into a single coherent texture map during pre-processing (Weinhaus and Devarajan 1997). Ideally, each surface triangle should select the source image where it is seen most directly (perpendicular to its normal) and at the resolution best matching the texture map resolution.²² This can be posed as a graph cut optimization problem, where the smoothness term encourages adjacent triangles to use similar source images, followed by blending to compensate for exposure differences (Lempitsky and Ivanov 2007; Sinha, Steedly *et al.* 2008). Even better results can be obtained by explicitly modeling geometric and photometric misalignments between the source images (Shum and Szeliski 2000; Gal, Wexler *et al.* 2010; Waechter, Moehrle, and Goesele 2014; Zhou and Koltun 2014; Huang, Dai *et al.* 2017; Fu, Yan *et al.* 2018; Schöps, Sattler, and Pollefeys 2019b; Lee, Ha *et al.* 2020). “Neural” texture map representations can also be used as an alternative to RGB color fields (Oechsle, Mescheder *et al.* 2019; Mihajlovic, Weder *et al.* 2021). Zollhöfer, Stotko *et al.* (2018, Section 4.1) discuss related techniques in more detail.

These kinds of approaches produce good results when the lighting stays fixed with respect to the object, i.e., when the camera moves around the object or space. When the lighting is strongly directional, however, and the object is being moved relative to this lighting, strong shading effects or specularities may be present, which will interfere with the reliable recovery of a texture (albedo) map. In this case, it is preferable to explicitly undo the shading effects (Section 13.1) by modeling the light source directions and estimating the surface reflectance properties while recovering the texture map (Sato and Ikeuchi 1996; Sato, Wheeler, and Ikeuchi 1997; Yu and Malik 1998; Yu, Debevec *et al.* 1999). Figure 13.31 shows the results of one such approach, where the specularities are first removed while estimating the

²²When surfaces are seen at oblique viewing angles, it may be necessary to blend different images together to obtain the best resolution (Wang, Kang *et al.* 2001).



Figure 13.31 *Estimating the diffuse albedo and reflectance parameters for a scanned 3D model (Sato, Wheeler, and Ikeuchi 1997) © 1997 ACM:* (a) set of input images projected onto the model; (b) the complete diffuse reflection (albedo) model; (c) rendering from the reflectance model including the specular component.

matte reflectance component (albedo) and then later re-introduced by estimating the specular component k_s in a Torrance–Sparrow reflection model (2.92).

13.7.1 Estimating BRDFs

A more ambitious approach to the problem of view-dependent appearance modeling is to estimate a general bidirectional reflectance distribution function (BRDF) for each point on an object’s surface. Dana, van Ginneken *et al.* (1999), Jensen, Marschner *et al.* (2001), and Lensch, Kautz *et al.* (2003) present different techniques for estimating such functions, while Dorsey, Rushmeier, and Sillion (2007) and Weyrich, Lawrence *et al.* (2009) provide surveys of the topics of BRDF modeling, recovery, and rendering.

As we saw in Section 2.2.2 (2.82), the BRDF can be written as

$$f_r(\theta_i, \phi_i, \theta_r, \phi_r; \lambda), \quad (13.6)$$

where (θ_i, ϕ_i) and (θ_r, ϕ_r) are the angles the incident $\hat{\mathbf{v}}_i$ and reflected $\hat{\mathbf{v}}_r$ light ray directions make with the local surface coordinate frame $(\hat{\mathbf{d}}_x, \hat{\mathbf{d}}_y, \hat{\mathbf{n}})$ shown in Figure 2.15. When modeling the appearance of an object, as opposed to the appearance of a patch of material, we need to estimate this function at every point (x, y) on the object’s surface, which gives us the *spatially varying* BRDF, or SVBRDF (Weyrich, Lawrence *et al.* 2009),

$$f_v(x, y, \theta_i, \phi_i, \theta_r, \phi_r; \lambda). \quad (13.7)$$

If sub-surface scattering effects are being modeled, such as the long-range transmission of light through materials such as alabaster, the eight-dimensional bidirectional scattering-surface reflectance-distribution function (BSSRDF) is used instead,

$$f_e(x_i, y_i, \theta_i, \phi_i, x_e, y_e, \theta_e, \phi_e; \lambda), \quad (13.8)$$

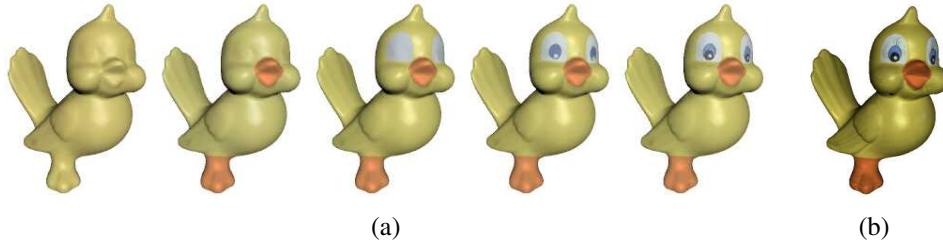


Figure 13.32 *Image-based reconstruction of appearance and detailed geometry (Lensch, Kautz et al. 2003) © 2003 ACM.* (a) Appearance models (BRDFs) are re-estimated using divisive clustering. (b) To model detailed spatially varying appearance, each lumitexel is projected onto the basis formed by the clustered materials.

where the e subscript now represents the *emitted* rather than the *reflected* light directions.

Weyrich, Lawrence *et al.* (2009) provide a nice survey of these and related topics, including basic photometry, BRDF models, traditional BRDF acquisition using *gonio reflectometry*, i.e., the precise measurement of visual angles and reflectances (Marschner, Westin *et al.* 2000; Dupuy and Jakob 2018), multiplexed illumination (Schechner, Nayar, and Belhumeur 2009), skin modeling (Debevec, Hawkins *et al.* 2000; Weyrich, Matusik *et al.* 2006), and image-based acquisition techniques, which simultaneously recover an object’s 3D shape and reflectometry from multiple photographs.

A nice example of this latter approach is the system developed by Lensch, Kautz *et al.* (2003), who estimate locally varying BRDFs and refine their shape models using local estimates of surface normals. To build up their models, they first associate a *lumitexel*, which contains a 3D position, a surface normal, and a set of sparse radiance samples, with each surface point. Next, they cluster such lumitexels into materials that share common properties, using a Lafontaine reflectance model (Lafontaine, Foo *et al.* 1997) and a divisive clustering approach (Figure 13.32a). Finally, to model detailed spatially varying appearance, each lumitexel (surface point) is projected onto the basis of clustered appearance models (Figure 13.32b). A more accurate system for estimating normals can be obtained using polarized lighting, as described by Ma, Hawkins *et al.* (2007).

More recent approaches to recovering spatially varying BRDFs (SVBRDFs) either start with RGB-D scanners (Park, Newcombe, and Seitz 2018; Schmitt, Donne *et al.* 2020), flash/no-flash image pairs (Aittala, Weyrich, and Lehtinen 2015), or use deep learning approaches to simultaneously estimate surface normals and appearance models (Li, Sunkavalli, and Chandraker 2018; Li, Xu *et al.* 2018). Even more sophisticated systems can also estimate shape and environmental lighting from range scanner sequences (Park, Holynski, and Seitz 2020) or

single monocular images (Boss, Jampani *et al.* 2020; Li, Shafiei *et al.* 2020; Chen, Nobuhara, and Nishino 2020) and even perform relighting on such scenes (Bi, Xu *et al.* 2020a,b; Sang and Chandraker 2020; Bi, Xu *et al.* 2020c). A more in-depth review of techniques for capturing the 3D shape and appearance of objects with RGB-D cameras can be found in the state of the art report by Zollhöfer, Stotko *et al.* (2018).

While most of the techniques discussed in this section require large numbers of views to estimate surface properties, an interesting challenge is to take these techniques out of the lab and into the real world, and to combine them with regular and internet photo image-based modeling approaches.

13.7.2 Application: 3D model capture

The techniques described in this chapter for building complete 3D models from multiple images and then recovering their surface appearance have opened up a whole new range of applications that often go under the name *3D photography*. Pollefeys and Van Gool (2002) and Pollefeys, Van Gool *et al.* (2004) provide nice introductions to such systems, including the processing steps of feature matching, structure from motion recovery, dense depth map estimation, 3D model building, and texture map recovery. A complete web-based system for automatically performing all of these tasks, called ARC3D, is described by Vergauwen and Van Gool (2006) and Moons, Van Gool, and Vergauwen (2010). The latter paper provides not only an in-depth survey of this whole field but also a detailed description of their complete end-to-end system.

An example of a more recent commercial photogrammetric modeling system that can be used for both object and scene capture is Pix4D, whose website shows a wonderful example of a 3D texture-mapped castle reconstructed from both regular and aerial drone photographs.²³ Examples of casual 3D photography enabled by the advent of smartphones include Hedman, Alsisan *et al.* (2017), Hedman and Kopf (2018), and Kopf, Matzen *et al.* (2020) and are described in more detail in Section 14.2.2.

An alternative to such fully automated systems is to put the user in the loop in what is sometimes called *interactive computer vision*. An early example of this was the Façade architectural modeling system developed by Debevec, Taylor, and Malik (1996). van den Hengel, Dick *et al.* (2007) describe their VideoTrace system, which performs automated point tracking and 3D structure recovery from video and then lets the user draw triangles and surfaces on top of the resulting point cloud, as well as interactively adjusting the locations of model vertices. Sinha, Steedly *et al.* (2008) describe a related system that uses matched vanishing

²³<https://www.pix4d.com/blog/mapping-chillon-castle-with-drone>

points in multiple images (Figure 7.50) to infer 3D line orientations and plane normals. These are then used to guide the user drawing axis-aligned planes, which are automatically fitted to the recovered 3D point cloud. Fully automated variants on these ideas are described by Zebedin, Bauer *et al.* (2008), Furukawa, Curless *et al.* (2009a), Furukawa, Curless *et al.* (2009b), Mičušík and Košecká (2009), and Sinha, Steedly, and Szeliski (2009).

As the sophistication and reliability of these techniques continues to improve, we can expect to see even more user-friendly applications for photorealistic 3D modeling from images (Exercise 13.8).

13.8 Additional reading

Shape from shading is one of the classic problems in computer vision (Horn 1975). Some representative papers in this area include those by Horn (1977), Ikeuchi and Horn (1981), Pentland (1984), Horn and Brooks (1986), Horn (1990), Szeliski (1991a), Mancini and Wolff (1992), Dupuis and Oliensis (1994), and Fua and Leclerc (1995). The collection of papers edited by Horn and Brooks (1989) is a great source of information on this topic, especially the chapter on variational approaches. The survey by Zhang, Tsai *et al.* (1999) reviews such techniques and also provides some comparative results.

Woodham (1981) wrote the seminal paper of photometric stereo. Shape from texture techniques include those by Witkin (1981), Ikeuchi (1981), Blostein and Ahuja (1987), Gårding (1992), Malik and Rosenholtz (1997), Liu, Collins, and Tsin (2004), Liu, Lin, and Hays (2004), Hays, Leordeanu *et al.* (2006), Lin, Hays *et al.* (2006), Lobay and Forsyth (2006), White and Forsyth (2006), White, Crane, and Forsyth (2007), and Park, Brocklehurst *et al.* (2009). Good papers and books on depth from defocus have been written by Pentland (1987), Nayar and Nakagawa (1994), Nayar, Watanabe, and Noguchi (1996), Watanabe and Nayar (1998), Chaudhuri and Rajagopalan (1999), and Favaro and Soatto (2006). Additional techniques for recovering shape from various kinds of illumination effects, including inter-reflections (Nayar, Ikeuchi, and Kanade 1991), are discussed in the book on shape recovery edited by Wolff, Shafer, and Healey (1992b). A more recent survey on photometric stereo is Ackermann and Goesele (2015) and recent papers include Logothetis, Mecca, and Cipolla (2019), Haefner, Ye *et al.* (2019), and Santo, Waechter, and Matsushita (2020).

Active rangefinding systems, which use laser or natural light illumination projected into the scene, have been described by Besl (1989), Rioux and Bird (1993), Kang, Webb *et al.* (1995), Curless and Levoy (1995), Curless and Levoy (1996), Proesmans, Van Gool, and Defoort (1998), Bouguet and Perona (1999), Curless (1999), Hebert (2000), Iddan and Yahav (2001), Goesele, Fuchs, and Seidel (2003), Scharstein and Szeliski (2003), Davis, Ra-

mamoothi, and Rusinkiewicz (2003), Zhang, Curless, and Seitz (2003), Zhang, Snavely *et al.* (2004), and Moons, Van Gool, and Vergauwen (2010), and in the more recent reviews by Zhang (2018) and Ikeuchi, Matsushita *et al.* (2020). Individual range scans can be aligned using 3D correspondence and distance optimization techniques such as *iterative closest points* and its variants (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Johnson and Kang 1997; Gold, Rangarajan *et al.* 1998; Johnson and Hebert 1999; Pulli 1999; David, DeMenthon *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009; Pomerleau, Colas, and Siegwart 2015; Rusinkiewicz 2019). Once they have been aligned, range scans can be merged using techniques that model the signed distance of surfaces to volumetric sample points (Hoppe, DeRose *et al.* 1992; Curless and Levoy 1996; Hilton, Stoddart *et al.* 1996; Wheeler, Sato, and Ikeuchi 1998; Kazhdan, Bolitho, and Hoppe 2006; Lempitsky and Boykov 2007; Zach, Pock, and Bischof 2007b; Zach 2008; Newcombe, Izadi *et al.* 2011; Zhou, Miller, and Koltun 2013; Newcombe, Fox, and Seitz 2015; Zollhöfer, Stotko *et al.* 2018).

Once constructed, 3D surfaces can be modeled and manipulated using a variety of three-dimensional representations, which include triangle meshes (Eck, DeRose *et al.* 1995; Hoppe 1996), splines (Farin 1992; Lee, Wolberg, and Shin 1997; Farin 2002), subdivision surfaces (Stollnitz, DeRose, and Salesin 1996; Zorin, Schröder, and Sweldens 1996; Warren and Weimer 2001; Peters and Reif 2008), and geometry images (Gu, Gortler, and Hoppe 2002). Alternatively, they can be represented as collections of point samples with local orientation estimates (Hoppe, DeRose *et al.* 1992; Szeliski and Tonnesen 1992; Turk and O'Brien 2002; Pfister, Zwicker *et al.* 2000; Alexa, Behr *et al.* 2003; Pauly, Keiser *et al.* 2003; Diebel, Thrun, and Brünig 2006; Guennebaud and Gross 2007; Guennebaud, Germann, and Gross 2008; Oztireli, Guennebaud, and Gross 2008; Berger, Tagliasacchi *et al.* 2017). They can also be modeled using implicit inside–outside characteristic or signed distance functions sampled on regular or irregular (octree) volumetric grids (Lavallée and Szeliski 1995; Szeliski and Lavallée 1996; Frisken, Perry *et al.* 2000; Dinh, Turk, and Slabaugh 2002; Kazhdan, Bolitho, and Hoppe 2006; Lempitsky and Boykov 2007; Zach, Pock, and Bischof 2007b; Zach 2008; Kazhdan and Hoppe 2013).

The literature on model-based 3D reconstruction is extensive. For modeling architecture and urban scenes, both interactive and fully automated systems have been developed. A special journal issue devoted to the reconstruction of large-scale 3D scenes (Zhu and Kanade 2008) is a good source of references and Robertson and Cipolla (2009) give a nice description of a complete system. Lots of additional references can be found in Section 13.6.1.

Face and whole body modeling and tracking is a very active sub-field of computer vision, with its own conferences and workshops, e.g., the International Conference on Auto-

matic Face and Gesture Recognition (FG) and IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG). Two recent survey papers on 3D face modeling and tracking are Zollhöfer, Thies *et al.* (2018) and Egger, Smith *et al.* (2020), while surveys on the topic of whole body modeling and tracking include Forsyth, Arikán *et al.* (2006), Moeslund, Hilton, and Krüger (2006), and Sigal, Balan, and Black (2010).

Some representative papers on recovering texture maps from multiple color and RGB-D images include Gal, Wexler *et al.* (2010), Waechter, Moehrle, and Goesele (2014), Zhou and Koltun (2014), and Lee, Ha *et al.* (2020) as well as Zollhöfer, Stotko *et al.* (2018, Section 4.1). The more complex process of recovering spatially varying BRDFs is covered in surveys by Dorsey, Rushmeier, and Sillion (2007) and Weyrich, Lawrence *et al.* (2009). More recent techniques that can do this using fewer images and RGB-D images include Aittala, Weyrich, and Lehtinen (2015), Li, Sunkavalli, and Chandraker (2018), Schmitt, Donne *et al.* (2020), and Boss, Jampani *et al.* (2020) and the survey by Zollhöfer, Stotko *et al.* (2018).

13.9 Exercises

Ex 13.1: Shape from focus. Grab a series of focused images with a digital SLR set to manual focus (or get one that allows for programmatic focus control) and recover the depth of an object.

1. Take some calibration images, e.g., of a checkerboard, so that you can compute a mapping between the amount of defocus and the focus setting.
2. Try both a fronto-parallel planar target and one which is slanted so that it covers the working range of the sensor. Which one works better?
3. Now put a real object in the scene and perform a similar focus sweep.
4. For each pixel, compute the local sharpness and fit a parabolic curve over focus settings to find the most in-focus setting.
5. Map these focus settings to depth and compare your result to ground truth. If you are using a known simple object, such as a sphere or cylinder (a ball or a soda can), it's easy to measure its true shape.
6. (Optional) See if you can recover the depth map from just two or three focus settings.
7. (Optional) Use an LCD projector to project artificial texture onto the scene. Use a pair of cameras to compare the accuracy of your shape from focus and shape from stereo techniques.

8. (Optional) Create an all-in-focus image using the technique of Agarwala, Dontcheva *et al.* (2004).

Ex 13.2: Shadow striping. Implement the handheld shadow striping system of Bouguet and Perona (1999). The basic steps include the following:

1. Set up two background planes behind the object of interest and calculate their orientation relative to the viewer, e.g., with fiducial marks.
2. Cast a moving shadow with a stick across the scene; record the video or capture the data with a webcam.
3. Estimate each light plane equation from the projections of the cast shadow against the two backgrounds.
4. Triangulate to the remaining points on each curve to get a 3D stripe and display the stripes using a 3D graphics engine.
5. (Optional) remove the requirement for a known second (vertical) plane and infer its location (or that of the light source) using the techniques described by Bouguet and Perona (1999). The techniques from Exercise 10.9 may also be helpful here.

Ex 13.3: Range data registration. Register two or more 3D datasets using either iterative closest points (ICP) (Besl and McKay 1992; Zhang 1994; Gold, Rangarajan *et al.* 1998) or octree signed distance fields (Szeliski and Lavallée 1996) (Section 13.2.1).

Apply your technique to narrow-baseline stereo pairs, e.g., obtained by moving a camera around an object, using structure from motion to recover the camera poses, and using a standard stereo matching algorithm.

Ex 13.4: Range data merging. Merge the datasets that you registered in the previous exercise using signed distance fields (Curless and Levoy 1996; Hilton, Stoddart *et al.* 1996) or one of their newer variants (Newcombe, Izadi *et al.* 2011; Hornung, Wurm *et al.* 2013; Nießner, Zollhöfer *et al.* 2013; Klingensmith, Dryanovski *et al.* 2015; Dai, Nießner *et al.* 2017; Zollhöfer, Stotko *et al.* 2018). Extract a meshed surface model from the signed distance field using marching cubes and display the resulting model.

Ex 13.5: Surface simplification. Use progressive meshes (Hoppe 1996) or some other technique from Section 13.3.2 to create a hierarchical simplification of your surface model.

Ex 13.6: Architectural modeler. Build a 3D interior or exterior model of some architectural structure, such as your house, from a series of handheld wide-angle photographs.

1. Extract lines and vanishing points (Exercises 7.11–7.14) to estimate the dominant directions in each image.
2. Use structure from motion to recover all of the camera poses and match up the vanishing points.
3. Let the user sketch the locations of the walls by drawing lines corresponding to wall bottoms, tops, and horizontal extents onto the images (Sinha, Steedly *et al.* 2008)—see also Exercise 11.4. Do something similar for openings (doors and windows) and simple furniture (tables and countertops).
4. Convert the resulting polygonal meshes into a 3D model (e.g., VRML) and optionally texture-map these surfaces from the images.

Ex 13.7: Body tracker. Download some human body movement sequences from one of the datasets such as HumanEva, MPI FAUST, or AMASS discussed in Section 13.6.4. Either implement a human motion tracker from scratch or extend existing code in some interesting way.

Ex 13.8: 3D photography. Combine all of your previously developed techniques to produce a system that takes a series of photographs or a video and constructs a photorealistic texture-mapped 3D model.

Chapter 14

Image-based rendering

14.1	View interpolation	863
14.1.1	View-dependent texture maps	865
14.1.2	<i>Application:</i> Photo Tourism	867
14.2	Layered depth images	868
14.2.1	Impostors, sprites, and layers	869
14.2.2	<i>Application:</i> 3D photography	872
14.3	Light fields and Lumigraphs	875
14.3.1	Unstructured Lumigraph	879
14.3.2	Surface light fields	880
14.3.3	<i>Application:</i> Concentric mosaics	882
14.3.4	<i>Application:</i> Synthetic re-focusing	883
14.4	Environment mattes	883
14.4.1	Higher-dimensional light fields	885
14.4.2	The modeling to rendering continuum	886
14.5	Video-based rendering	887
14.5.1	Video-based animation	888
14.5.2	Video textures	889
14.5.3	<i>Application:</i> Animating pictures	892
14.5.4	3D and free-viewpoint Video	893
14.5.5	<i>Application:</i> Video-based walkthroughs	896
14.6	Neural rendering	899
14.7	Additional reading	908
14.8	Exercises	910

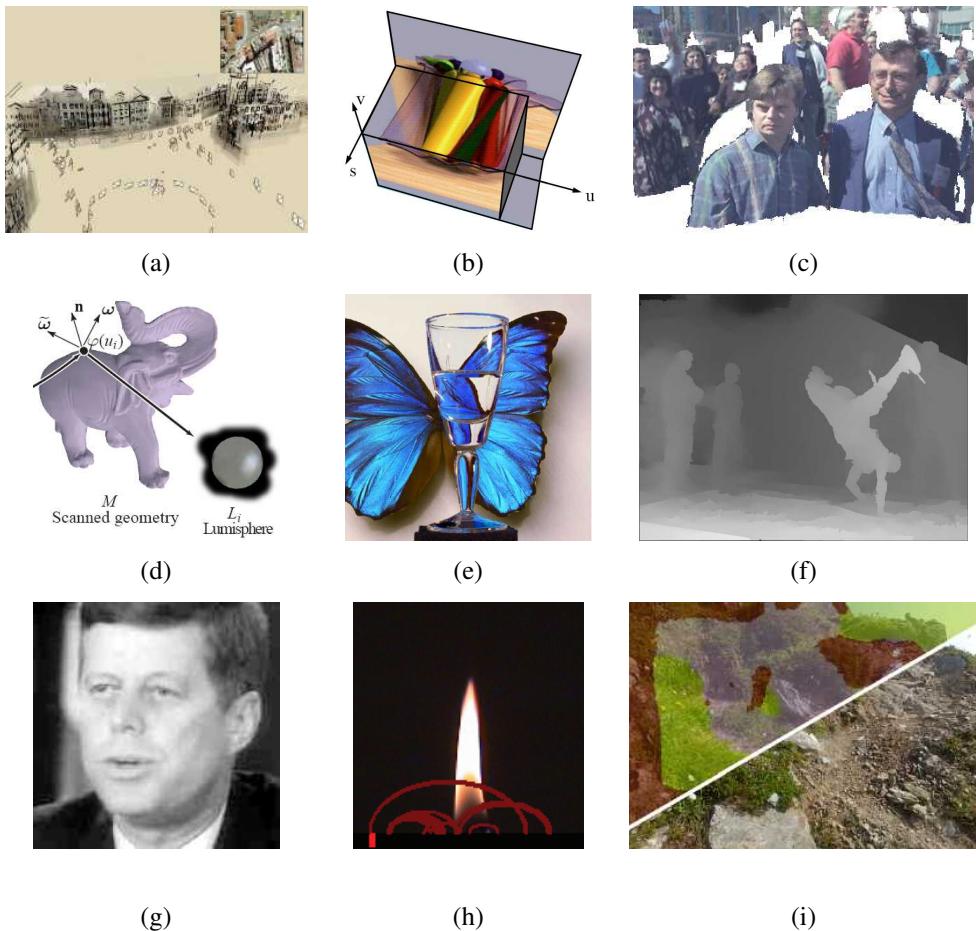


Figure 14.1 Image-based and video-based rendering: (a) a 3D view of a Photo Tourism reconstruction (Snavely, Seitz, and Szeliski 2006) © 2006 ACM; (b) a slice through a 4D light field (Gortler, Grzeszczuk et al. 1996) © 1996 ACM; (c) sprites with depth (Shade, Gortler et al. 1998) © 1998 ACM; (d) surface light field (Wood, Azuma et al. 2000) © 2000 ACM; (e) environment matte in front of a novel background (Zongker, Werner et al. 1999) © 1999 ACM; (f) video view interpolation (Zitnick, Kang et al. 2004) © 2004 ACM; (g) Video Rewrite used to re-animate old video (Bregler, Covell, and Slaney 1997) © 1997 ACM; (h) video texture of a candle flame (Schödl, Szeliski et al. 2000) © 2000 ACM; (i) hyperlapse video, stitching multiple frames with 3D proxies (Kopf, Cohen, and Szeliski 2014) © 2014 ACM.

Over the last few decades, image-based rendering has emerged as one of the most exciting applications of computer vision (Kang, Li *et al.* 2006; Shum, Chan, and Kang 2007; Gallo, Troccoli *et al.* 2020). In image-based rendering, 3D reconstruction techniques from computer vision are combined with computer graphics rendering techniques that use multiple views of a scene to create interactive photo-realistic experiences such as the Photo Tourism system shown in Figure 14.1a. Commercial versions of such systems include immersive street-level navigation in online mapping systems such as Google Maps and the creation of 3D Photo-synths from large collections of casually acquired photographs.

In this chapter, we explore a variety of image-based rendering techniques, such as those illustrated in Figure 14.1. We begin with *view interpolation* (Section 14.1), which creates a seamless transition between a pair of reference images using one or more precomputed depth maps. Closely related to this idea are *view-dependent texture maps* (Section 14.1.1), which blend multiple texture maps on a 3D model’s surface. The representations used for both the color imagery and the 3D geometry in view interpolation include a number of clever variants such as *layered depth images* (Section 14.2) and *sprites with depth* (Section 14.2.1).

We continue our exploration of image-based rendering with the *light field* and *Lumigraph* four-dimensional representations of a scene’s appearance (Section 14.3), which can be used to render the scene from any arbitrary viewpoint. Variants on these representations include the *unstructured Lumigraph* (Section 14.3.1), *surface light fields* (Section 14.3.2), *concentric mosaics* (Section 14.3.3), and *environment mattes* (Section 14.4).

We then explore the topic of *video-based rendering*, which uses one or more videos To create novel video-based experiences (Section 14.5). The topics we cover include video-based facial animation (Section 14.5.1), as well as *video textures* (Section 14.5.2), in which short video clips can be seamlessly looped to create dynamic real-time video-based renderings of a scene.

We continue with a discussion of *3D videos* created from multiple video streams (Section 14.5.4), as well as *video-based walkthroughs* of environments (Section 14.5.5), which have found widespread application in immersive outdoor mapping and driving direction systems. We finish this chapter with a review of recent work in *neural rendering* (Section 14.6), where generative neural networks are used to create more realistic reconstructions of both static scenes and objects as well as people.

14.1 View interpolation

While the term *image-based rendering* first appeared in the papers by Chen (1995) and McMillan and Bishop (1995), the work on *view interpolation* by Chen and Williams (1993)

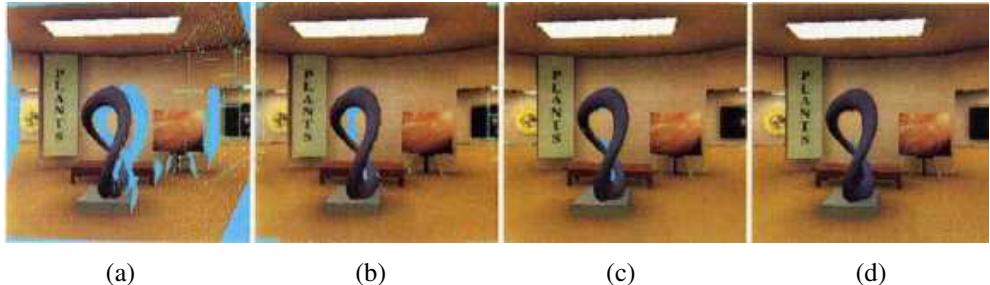


Figure 14.2 View interpolation (Chen and Williams 1993) © 1993 ACM: (a) holes from one source image (shown in blue); (b) holes after combining two widely spaced images; (c) holes after combining two closely spaced images; (d) after interpolation (hole filling).

is considered as the seminal paper in the field. In view interpolation, pairs of rendered images are combined with their precomputed depth maps to generate interpolated views that mimic what a virtual camera would see in between the two reference views. Since its original introduction, the whole field of *novel view synthesis* from captured images has continued to be a very active area. A good historical overview and recent results can be found in the CVPR tutorial on this topic (Gallo, Troccoli *et al.* 2020).

View interpolation combines two ideas that were previously used in computer vision and computer graphics. The first is the idea of pairing a recovered depth map with the reference image used in its computation and then using the resulting texture-mapped 3D model to generate novel views (Figure 12.1). The second is the idea of *morphing* (Section 3.6.3) (Figure 3.51), where correspondences between pairs of images are used to warp each reference image to an in-between location while simultaneously cross-dissolving between the two warped images.

Figure 14.2 illustrates this process in more detail. First, both source images are warped to the novel view, using both the knowledge of the reference and virtual 3D camera pose along with each image's depth map (2.68–2.70). In the paper by Chen and Williams (1993), a *forward warping* algorithm (Algorithm 3.1 and Figure 3.45) is used. The depth maps are represented as quadtrees for both space and rendering time efficiency (Samet 1989).

During the forward warping process, multiple pixels (which occlude one another) may land on the same destination pixel. To resolve this conflict, either a *z-buffer* depth value can be associated with each destination pixel or the images can be warped in back-to-front order, which can be computed based on the knowledge of epipolar geometry (Chen and Williams 1993; Laveau and Faugeras 1994; McMillan and Bishop 1995).

Once the two reference images have been warped to the novel view (Figure 14.2a–b), they

can be merged to create a coherent composite (Figure 14.2c). Whenever one of the images has a *hole* (illustrated as a cyan pixel), the other image is used as the final value. When both images have pixels to contribute, these can be blended as in usual morphing, i.e., according to the relative distances between the virtual and source cameras. Note that if the two images have very different exposures, which can happen when performing view interpolation on real images, the hole-filled regions and the blended regions will have different exposures, leading to subtle artifacts.

The final step in view interpolation (Figure 14.2d) is to fill any remaining holes or cracks due to the forward warping process or lack of source data (scene visibility). This can be done by copying pixels from the *further* pixels adjacent to the hole. (Otherwise, foreground objects are subject to a “fattening effect”.)

The above process works well for rigid scenes, although its visual quality (lack of aliasing) can be improved using a two-pass, forward–backward algorithm (Section 14.2.1) (Shade, Gortler *et al.* 1998) or full 3D rendering (Zitnick, Kang *et al.* 2004). In the case where the two reference images are views of a non-rigid scene, e.g., a person smiling in one image and frowning in the other, *view morphing*, which combines ideas from view interpolation with regular morphing, can be used (Seitz and Dyer 1996). A depth map fitted to a face can also be used to synthesize a view from a longer distance, removing the enlarged nose and other facial features common to “selfie” photography (Fried, Shechtman *et al.* 2016).

While the original view interpolation paper describes how to generate novel views based on similar precomputed (linear perspective) images, the *plenoptic modeling* paper of McMillan and Bishop (1995) argues that cylindrical images should be used to store the precomputed rendering or real-world images. Chen (1995) also proposes using environment maps (cylindrical, cubic, or spherical) as source images for view interpolation.

14.1.1 View-dependent texture maps

View-dependent texture maps (Debevec, Taylor, and Malik 1996) are closely related to view interpolation. Instead of associating a separate depth map with each input image, a single 3D model is created for the scene, but different images are used as texture map sources depending on the virtual camera’s current position (Figure 14.3a).¹

In more detail, given a new virtual camera position, the similarity of this camera’s view of each polygon (or pixel) is compared to that of potential source images. The images are then blended using a weighting that is inversely proportional to the angles α_i between the

¹The term *image-based modeling*, which is now commonly used to describe the creation of texture-mapped 3D models from multiple images, appears to have first been used by Debevec, Taylor, and Malik (1996), who also used the term *photogrammetric modeling* to describe the same process.

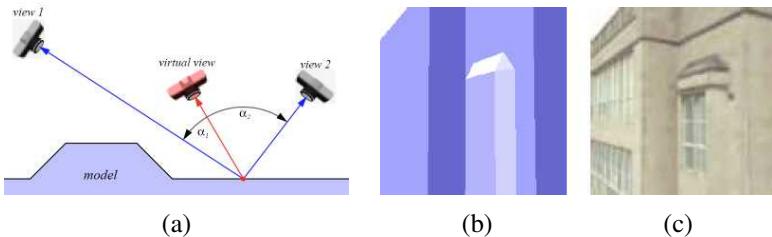


Figure 14.3 *View-dependent texture mapping (Debevec, Taylor, and Malik 1996) © 1996 ACM.* (a) The weighting given to each input view depends on the relative angles between the novel (virtual) view and the original views; (b) simplified 3D model geometry; (c) with view-dependent texture mapping, the geometry appears to have more detail (recessed windows).

virtual view and the source views (Figure 14.3a).² Even though the geometric model can be fairly coarse (Figure 14.3b), blending different views gives a strong sense of more detailed geometry because of the visual motion between corresponding pixels. While the original paper performs the weighted blend computation separately at each pixel or coarsened polygon face, follow-on work by Debevec, Yu, and Borshukov (1998) presents a more efficient implementation based on precomputing contributions for various portions of viewing space and then using projective texture mapping (OpenGL-ARB 1997).

The idea of view-dependent texture mapping has been used in a large number of subsequent image-based rendering systems, including facial modeling and animation (Pighin, Hecker *et al.* 1998) and 3D scanning and visualization (Pulli, Abi-Rached *et al.* 1998). Closely related to view-dependent texture mapping is the idea of blending between light rays in 4D space, which forms the basis of the Lumigraph and unstructured Lumigraph systems (Section 14.3) (Gortler, Grzeszczuk *et al.* 1996; Buehler, Bosse *et al.* 2001).

To provide even more realism in their Façade system, Debevec, Taylor, and Malik (1996) also include a *model-based stereo* component, which computes an offset (parallax) map for each coarse planar facet of their model. They call the resulting analysis and rendering system a *hybrid geometry- and image-based* approach, as it uses traditional 3D geometric modeling to create the global 3D model, but then uses local depth offsets, along with view interpolation, to add visual realism. Instead of warping per-pixel depth maps or coarser triangulated geometry (as in unstructured Lumigraphs, Section 14.3.1), it is also possible to use superpixels as the basic primitives being warped (Chaurasia, Duchene *et al.* 2013). Fixed rules for view-dependent blending can also be replaced with deep neural networks, as in the *deep blending* system by Hedman, Philip *et al.* (2018).

²More sophisticated blending weights are discussed in Section 14.3.1 on unstructured Lumigraph rendering.

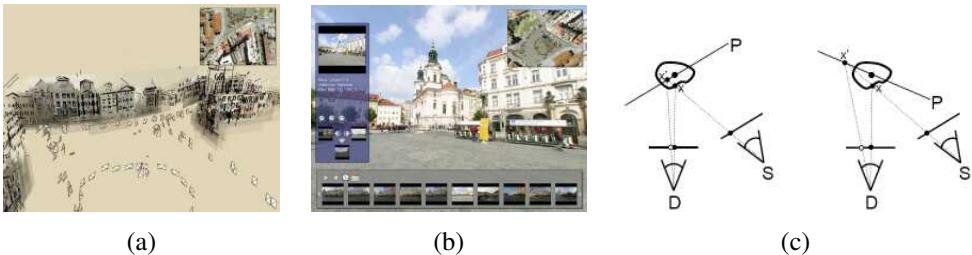


Figure 14.4 *Photo Tourism* (Snavely, Seitz, and Szeliski 2006) © 2006 ACM: (a) a 3D overview of the scene, with translucent washes and lines painted onto the planar impostors; (b) once the user has selected a region of interest, a set of related thumbnails is displayed along the bottom; (c) planar proxy selection for optimal stabilization (Snavely, Garg et al. 2008) © 2008 ACM.

14.1.2 Application: Photo Tourism

While view interpolation was originally developed to accelerate the rendering of 3D scenes on low-powered processors and systems without graphics acceleration, it turns out that it can be applied directly to large collections of casually acquired photographs. The *Photo Tourism* system developed by Snavely, Seitz, and Szeliski (2006) uses structure from motion to compute the 3D locations and poses of all the cameras taking the images, along with a sparse 3D point-cloud model of the scene (Section 11.4.6, Figure 11.17).

To perform an image-based exploration of the resulting *sea of images* (Aliaga, Funkhouser *et al.* 2003), Photo Tourism first associates a 3D proxy with each image. While a triangulated mesh obtained from the point cloud can sometimes form a suitable proxy, e.g., for outdoor terrain models, a simple dominant plane fit to the 3D points visible in each image often performs better, because it does not contain any erroneous segments or connections that pop out as artifacts. As automated 3D modeling techniques continue to improve, however, the pendulum may swing back to more detailed 3D geometry (Goesele, Snavely *et al.* 2007; Sinha, Steedly, and Szeliski 2009). One example is the hybrid rendering system developed by Goesele, Ackermann *et al.* (2010), who use dense per-image depth maps for the well-reconstructed portions of each image and 3D colored point clouds for the less confident regions.

The resulting image-based navigation system lets users move from photo to photo, either by selecting cameras from a top-down view of the scene (Figure 14.4a) or by selecting regions of interest in an image, navigating to nearby views, or selecting related thumbnails (Figure 14.4b). To create a background for the 3D scene, e.g., when being viewed from above, non-photorealistic techniques (Section 10.5.2), such as translucent color washes or

highlighted 3D line segments, can be used (Figure 14.4a). The system can also be used to annotate regions of images and to automatically propagate such annotations to other photographs.

The 3D planar proxies used in Photo Tourism and the related Photosynth system from Microsoft result in non-photorealistic transitions reminiscent of visual effects such as “page flips”. Selecting a stable 3D axis for all the planes can reduce the amount of swimming and enhance the perception of 3D (Figure 14.4c) (Snavely, Garg *et al.* 2008). It is also possible to automatically detect objects in the scene that are seen from multiple views and create “orbits” of viewpoints around such objects. Furthermore, nearby images in both 3D position and viewing direction can be linked to create “virtual paths”, which can then be used to navigate between arbitrary pairs of images, such as those you might take yourself while walking around a popular tourist site (Snavely, Garg *et al.* 2008). This idea has been further developed and released as a feature on Google Maps called Photo Tours (Kushal, Self *et al.* 2012).³ The quality of such synthesized virtual views has become so accurate that Shan, Adams *et al.* (2013) propose a *visual Turing test* to distinguish between synthetic and real images. Waechter, Beljan *et al.* (2017) produce higher-resolution quality assessments of image-based modeling and rendering system using what they call *virtual rephotography*. Further improvements can be obtained using even more recent *neural rendering* techniques (Hedman, Philip *et al.* 2018; Meshry, Goldman *et al.* 2019; Li, Xian *et al.* 2020), which we discuss in Section 14.6.

The spatial matching of image features and regions performed by Photo Tourism can also be used to infer more information from large image collections. For example, Simon, Snavely, and Seitz (2007) show how the match graph between images of popular tourist sites can be used to find the most *iconic* (commonly photographed) objects in the collection, along with their related tags. In follow-on work, Simon and Seitz (2008) show how such tags can be propagated to sub-regions of each image, using an analysis of which 3D points appear in the central portions of photographs. Extensions of these techniques to *all* of the world’s images, including the use of GPS tags where available, have been investigated as well (Li, Wu *et al.* 2008; Quack, Leibe, and Van Gool 2008; Crandall, Backstrom *et al.* 2009; Li, Crandall, and Huttenlocher 2009; Zheng, Zhao *et al.* 2009; Raguram, Wu *et al.* 2011).

14.2 Layered depth images

Traditional view interpolation techniques associate a single depth map with each source or reference image. Unfortunately, when such a depth map is warped to a novel view, holes and

³<https://maps.googleblog.com/2012/04/visit-global-landmarks-with-photo-tours.html>

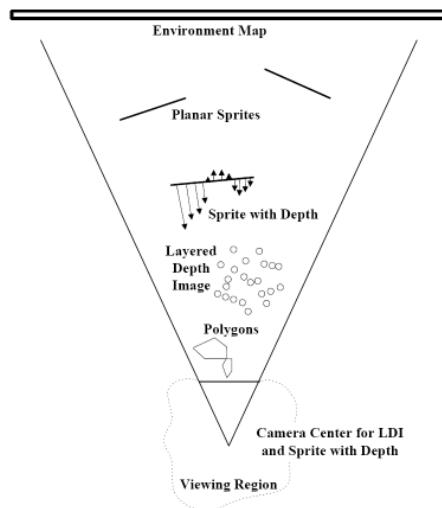


Figure 14.5 A variety of image-based rendering primitives, which can be used depending on the distance between the camera and the object of interest (Shade, Gortler et al. 1998) © 1998 ACM. Closer objects may require more detailed polygonal representations, while mid-level objects can use a layered depth image (LDI), and far-away objects can use sprites (potentially with depth) and environment maps.

cracks inevitably appear behind the foreground objects. One way to alleviate this problem is to keep several depth and color values (*depth pixels*) at every pixel in a reference image (or, at least for pixels near foreground–background transitions) (Figure 14.5). The resulting data structure, which is called a *layered depth image* (LDI), can be used to render new views using a back-to-front forward warping (splatting) algorithm (Shade, Gortler et al. 1998).

14.2.1 Impostors, sprites, and layers

An alternative to keeping lists of color-depth values at each pixel, as is done in the LDI, is to organize objects into different *layers* or *sprites*. The term sprite originates in the computer game industry, where it is used to designate flat animated characters in games such as Pac-Man or Mario Bros. When put into a 3D setting, such objects are often called *impostors*, because they use a piece of flat, alpha-matted geometry to represent simplified versions of 3D objects that are far away from the camera (Shade, Lischinski et al. 1996; Lengyel and Snyder 1997; Torborg and Kajiya 1996). In computer vision, such representations are usually called *layers* (Wang and Adelson 1994; Baker, Szeliski, and Anandan 1998; Torr, Szeliski,



Figure 14.6 *Sprites with depth* (Shade, Gortler *et al.* 1998) © 1998 ACM: (a) alpha-matted color sprite; (b) corresponding relative depth or parallax; (c) rendering without relative depth; (d) rendering with depth (note the curved object boundaries).

and Anandan 1999; Birchfield, Natarajan, and Tomasi 2007). Section 9.4.2 discusses the topics of transparent layers and reflections, which occur on specular and transparent surfaces such as glass.

While flat layers can often serve as an adequate representation of geometry and appearance for far-away objects, better geometric fidelity can be achieved by also modeling the per-pixel offsets relative to a base plane, as shown in Figures 14.5 and 14.6a–b. Such representations are called *plane plus parallax* in the computer vision literature (Kumar, Anandan, and Hanna 1994; Sawhney 1994; Szeliski and Coughlan 1997; Baker, Szeliski, and Anandan 1998), as discussed in Section 9.4 (Figure 9.14). In addition to fully automated stereo techniques, it is also possible to paint in depth layers (Kang 1998; Oh, Chen *et al.* 2001; Shum, Sun *et al.* 2004) or to infer their 3D structure from monocular image cues (Sections 6.4.4 and 12.8) (Hoiem, Efros, and Hebert 2005b; Saxena, Sun, and Ng 2009).

How can we render a sprite with depth from a novel viewpoint? One possibility, as with a regular depth map, is to just forward warp each pixel to its new location, which can cause aliasing and cracks. A better way, which we have already mentioned in Section 3.6.2, is to first warp the depth (or (u, v) displacement) map to the novel view, fill in the cracks, and then use higher-quality inverse warping to resample the color image (Shade, Gortler *et al.* 1998). Figure 14.6d shows the results of applying such a two-pass rendering algorithm. From this still image, you can appreciate that the foreground sprites look more rounded; however, to fully appreciate the improvement in realism, you would have to look at the actual animated sequence.

Sprites with depth can also be rendered using conventional graphics hardware, as described in (Zitnick, Kang *et al.* 2004). Rogmans, Lu *et al.* (2009) describe GPU implementations of both real-time stereo matching and real-time forward and inverse rendering algorithms.

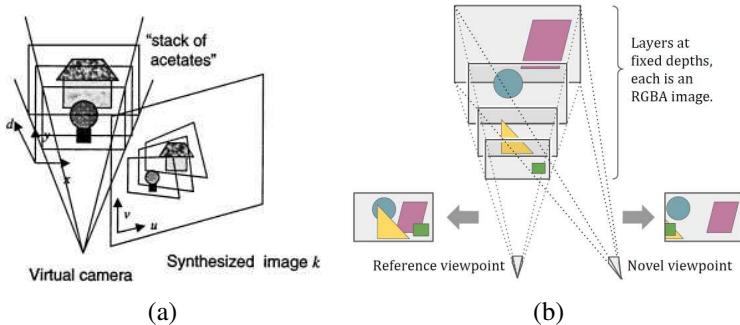


Figure 14.7 Finely sliced fronto-parallel layers: (a) stack of acetates (Szeliski and Golland 1999) © 1999 Springer and (b) multiplane images (Zhou, Tucker et al. 2018) © 2018 ACM. These representations (which are equivalent) consist of a set of fronto-parallel planes at fixed depths from a reference camera coordinate frame, with each plane encoding an RGB image and an alpha map that capture the scene appearance at the corresponding depth.

An alternative to constructing a small number of layers is to discretize the viewing frustum subtending a layered depth image into a large number of fronto-parallel planes, each of which contains RGBA values (Szeliski and Golland 1999), as shown in Figure 14.7. This is the same spatial representation we presented in Section 12.1.2 and Figure 12.6 on *plane sweep* approaches to stereo, except that here it is being used to represent a colored 3D scene instead of accumulating a matching cost volume. This representation is essentially a perspective variant of a volumetric representation containing RGB color and α opacity values (Sections 13.2.1 and 13.5).

This representation was recently rediscovered and now goes under the popular name of *multiplane images (MPI)* (Zhou, Tucker et al. 2018). Figure 14.8 shows an MPI representation derived from a stereo image pair along with a novel synthesized view. MPIs are easier to derive from pairs or collections of stereo images than true (minimal) layered representations because there is a 1:1 correspondence between pixels (actually, voxels) in a plane sweep cost volume (Figure 12.5) and an MPI. However, they are not as compact and can lead to tearing artifacts once the viewpoint exceeds a certain range. (We will talk about using inpainting to mitigate such holes in image-based representations in Section 14.2.2). MPIs are also related to the *soft 3D* volumetric representation proposed earlier by Penner and Zhang (2017).

Since their initial development for novel view extrapolation, i.e., ‘‘stereo magnification’’ (Zhou, Tucker et al. 2018), MPIs have found a wide range of applications in image-based rendering, including extension to multiple input images and faster inference (Flynn, Broxton et al. 2019), CNN refinement and better inpainting (Srinivasan, Tucker et al. 2019), inter-



Figure 14.8 MPI representation constructed from a stereo pair of color images, along with a novel view reconstructed from the MPI (Zhou, Tucker et al. 2018) © 2018 ACM. Note how the planes slice the 3D scene into thin layers, each of which has colors and full or partial opacities in only a small region.

polating between collections of MPIS (Mildenhall, Srinivasan et al. 2019), and large view extrapolations (Choi, Gallo et al. 2019). The planar MPI structure has also been generalized to curved surfaces for representing partial or complete 3D panoramas (Broxton, Flynn et al. 2020; Attal, Ling et al. 2020; Lin, Xu et al. 2020).⁴

Another important application of layers is in the modeling of reflections. When the reflector (e.g., a glass pane) is planar, the reflection forms a *virtual image*, which can be modeled as a separate layer (Section 9.4.2 and Figures 9.16–9.17), so long as *additive* (instead of *over*) compositing is used to combine the reflected and transmitted images (Szeliski, Avidan, and Anandan 2000; Sinha, Kopf et al. 2012; Kopf, Langguth et al. 2013). Figure 14.9 shows an example of a two-layer decomposition reconstructed from a short video clip, which can be re-rendered from novel views by adding warped versions of the two layers (each of which has its own depth map). When the reflective surface is curved, a quasi-stable virtual image may still be available, although this depends on the local variations in principal curvatures (Swaminathan, Kang et al. 2002; Criminisi, Kang et al. 2005). The modeling of reflections is one of the advantages attributed to layered representations such as MPIS (Zhou, Tucker et al. 2018; Broxton, Flynn et al. 2020), although in these papers over compositing is still used, which results in plausible but not physically correct renderings.

14.2.2 Application: 3D photography

The desire to capture and view photographs of the world in 3D prompted the development of stereo cameras and viewers in the mid-1800s (Luo, Kong et al. 2020) and more recently

⁴Exploring the interactive 3D videos on the authors’ websites, e.g., <https://augmentedperception.github.io/deepviewvideo>, is a good way to get a sense of this new medium.

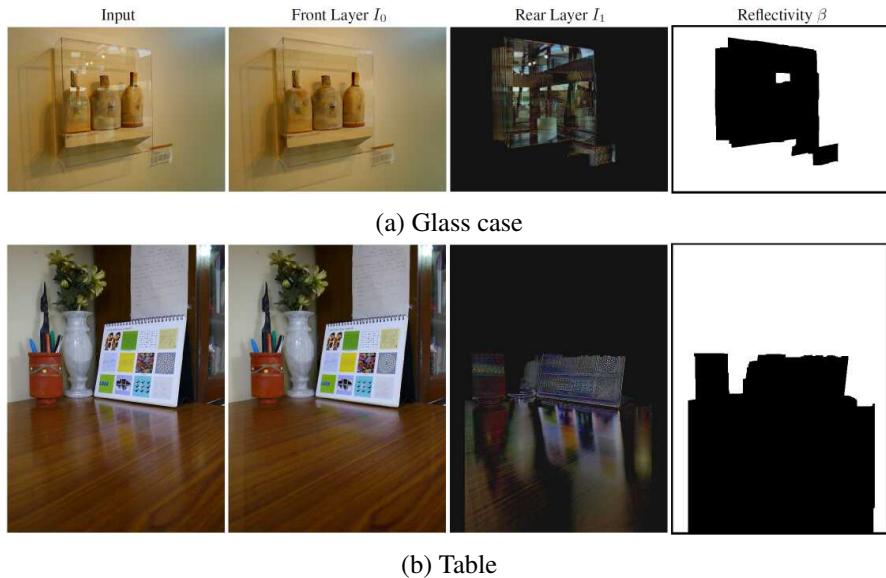
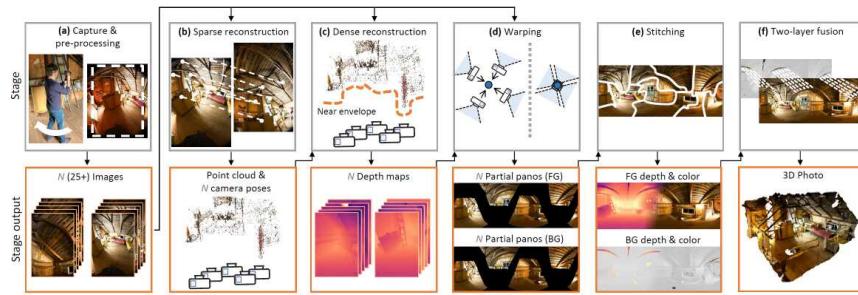


Figure 14.9 *Image-based rendering of scenes with reflections using multiple additive layers (Sinha, Kopf et al. 2012) © 2012 ACM.* The left column shows an image from the input sequence and the next two columns show the two separated layers (transmitted and reflected light). The last column is an estimate of which portions of the scene are reflective. As you can see, stray bits of reflections sometimes cling to the transmitted light layer. Note how in the table, the amount of reflected light (gloss) decreases towards the bottom of the image because of Fresnel reflection.

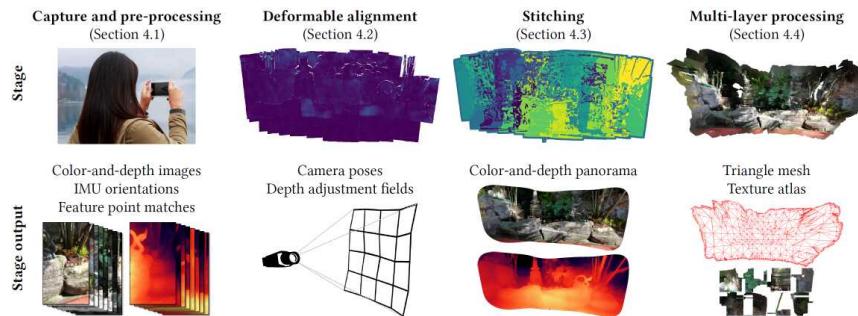
the popularity of 3D movies.⁵ It has also underpinned much of the research in 3D shape and appearance capture and modeling we studied in the previous chapter and more specifically Section 13.7.2. Until recently, however, while the required multiple images could be captured with hand-held cameras (Pollefeys, Van Gool et al. 2004; Snavely, Seitz, and Szeliski 2006), desktop or laptop computers were required to process and interactively view the images.

The ability to capture, construct, and widely share such 3D models has dramatically increased in the last few years and now goes under the name of *3D photography*. Hedman, Alsisan et al. (2017) describe their *Casual 3D Photography* system, which takes a sequence of overlapping images taken from a moving camera and then uses a combination of structure from motion, multi-view stereo, and 3D image warping and stitching to construct two-layer partial panoramas that can be viewed on a computer, as shown in Figure 14.10. The Instant

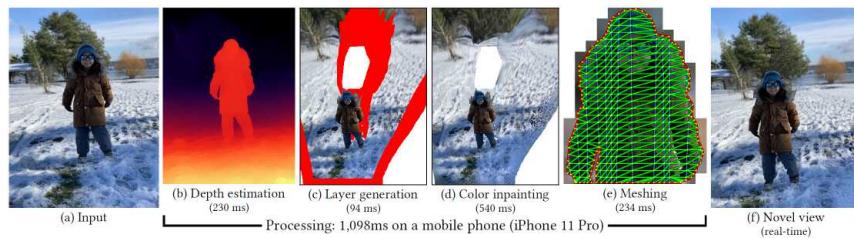
⁵It is interesting to note, however, that for now (at least), in-home 3D TV sets have failed to take off.



(a) Casual 3D Photography



(b) Instant 3D Photography



(c) One Shot 3D Photography

Figure 14.10 Systems for capturing and modeling 3D scenes from handheld photographs.

(a) *Casual 3D Photography* takes a series of overlapping images and constructs per-image depth maps, which are then warped and blended together into a two-layer representation (Hedman, Alsisan et al. 2017) © 2017 ACM. (b) *Instant 3D Photography* starts with the depth maps produced by a dual-lens smartphone and warps and registers the depth maps to create a similar representation with far less computation (Hedman and Kopf 2018) © 2018 ACM. (c) *One Shot 3D Photography* starts with a single photo, performs monocular depth estimation, layer construction and inpainting, and mesh and atlas generation, enabling phone-based reconstruction and interactive viewing (Kopf, Matzen et al. 2020) © 2020 ACM.

3D system of [Hedman and Kopf \(2018\)](#) builds a similar system, but starts with the depth images available from newer dual-camera smartphones to significantly speed up the process. Note, however, that the individual depth images are not *metric*, i.e., related to true depth with a single global scalar transformation, so must be deformably warped before being stitched together. A *texture atlas* is then constructed to compactly store the pixel color values while also supporting multiple layers.

While these systems produce beautiful wide 3D images that can create a true sense of immersion (“being there”), much more practical and fast solutions can be constructed using a single depth image. [Kopf, Alsisan et al. \(2019\)](#) describe their phone-based system, which takes a single dual-lens photograph with its estimated depth map and constructs a multi-layer 3D photograph with occluded pixels being *inpainted* from nearby background pixels (see Section 10.5.1 and [Shih, Su et al. 2020](#)).⁶ To remove the requirement for depth maps being associated with the input images [Kopf, Matzen et al. \(2020\)](#) use a monocular depth inference network (Section 12.8) to estimate the depth, thereby enabling 3D photos to be produced from any photograph in a phone’s camera roll, or even from historical photographs, as shown in Figure 14.10c.⁷ When historic stereographs are available, these can be used to create even more accurate 3D photographs, as shown by [Luo, Kong et al. \(2020\)](#). It is also possible to create a “3D Ken Burns” effect, i.e., small looming video clips, from regular images using monocular depth inference ([Niklaus, Mai et al. 2019](#)).⁸

14.3 Light fields and Lumigraphs

While image-based rendering approaches can synthesize scene renderings from novel viewpoints, they raise the following more general question:

Is it possible to capture and render the appearance of a scene from all possible viewpoints and, if so, what is the complexity of the resulting structure?

Let us assume that we are looking at a static scene, i.e., one where the objects and illuminants are fixed, and only the observer is moving around. Under these conditions, we can describe each image by the location and orientation of the virtual camera (6 dof) as well as

⁶Facebook rolled out 3D photographs for the iPhone in October 2018, <https://facebook360.fb.com/2018/10/11/3d-photos-now-rolling-out-on-facebook-and-in-vr>, along with the ability to post and interactively view the photos.

⁷In February 2020, Facebook released the ability to use regular photos, <https://ai.facebook.com/blog/powering-by-ai-turning-any-2d-photo-into-3d-using-convolutional-neural-nets>.

⁸Google released a similar feature called Cinematic photos <https://blog.google/products/photos/new-cinematic-photos-and-more-ways-relive-your-memories>.

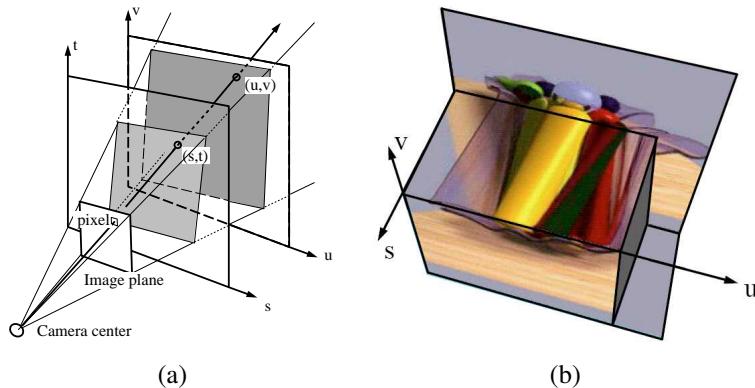


Figure 14.11 The Lumigraph (Gortler, Grzeszczuk et al. 1996) © 1996 ACM: (a) a ray is represented by its 4D two-plane parameters (s, t) and (u, v) ; (b) a slice through the 3D light field subset (u, v, s) .

its intrinsics (e.g., its focal length). However, if we capture a two-dimensional *spherical* image around each possible camera location, we can re-render any view from this information.⁹ Thus, taking the cross-product of the three-dimensional space of camera positions with the 2D space of spherical images, we obtain the 5D *plenoptic function* of Adelson and Bergen (1991), which forms the basis of the image-based rendering system of McMillan and Bishop (1995).

Notice, however, that when there is no light dispersion in the scene, i.e., no smoke or fog, all the coincident rays along a portion of free space (between solid or refractive objects) have the same color value. Under these conditions, we can reduce the 5D plenoptic function to the 4D *light field* of all possible rays (Gortler, Grzeszczuk et al. 1996; Levoy and Hanrahan 1996; Levoy 2006).¹⁰

To make the parameterization of this 4D function simpler, let us put two planes in the 3D scene roughly bounding the area of interest, as shown in Figure 14.11a. Any light ray terminating at a camera that lives in front of the *st* plane (assuming that this space is empty) passes through the two planes at (s, t) and (u, v) and can be described by its 4D coordinate (s, t, u, v) . This diagram (and parameterization) can be interpreted as describing a family of cameras living on the *st* plane with their image planes being the *uv* plane. The *uv* plane can be placed at infinity, which corresponds to all the virtual cameras looking in the same

⁹As we are counting dimensions, we ignore for now any sampling or resolution issues.

¹⁰Levoy and Hanrahan (1996) borrowed the term *light field* from a paper by Gershun (1939). Another name for this representation is the *photic field* (Moon and Spencer 1981).

direction.

In practice, if the planes are of finite extent, the finite *light slab* $L(s, t, u, v)$ can be used to generate any synthetic view that a camera would see through a (finite) *viewport* in the st plane with a view frustum that wholly intersects the far uv plane. To enable the camera to move all the way around an object, the 3D space surrounding the object can be split into multiple domains, each with its own light slab parameterization. Conversely, if the camera is moving inside a bounded volume of free space looking outward, multiple cube faces surrounding the camera can be used as (s, t) planes.

Thinking about 4D spaces is difficult, so let us drop our visualization by one dimension. If we fix the row value t and constrain our camera to move along the s axis while looking at the uv plane, we can stack all of the stabilized images the camera sees to get the (u, v, s) *epipolar volume*, which we discussed in Section 12.7. A “horizontal” cross-section through this volume is the well-known *epipolar plane image* (Bolles, Baker, and Marimont 1987), which is the us slice shown in Figure 14.11b.

As you can see in this slice, each color pixel moves along a linear track whose slope is related to its depth (parallax) from the uv plane. (Pixels exactly on the uv plane appear “vertical”, i.e., they do not move as the camera moves along s .) Furthermore, pixel tracks occlude one another as their corresponding 3D surface elements occlude. Translucent pixels, however, composite *over* background pixels (Section 3.1.3 (3.8)) rather than occluding them. Thus, we can think of adjacent pixels sharing a similar planar geometry as *EPI strips* or *EPI tubes* (Criminisi, Kang *et al.* 2005). 3D lightfields taken from a camera slowly moving through a static scene can be an excellent source for high-accuracy 3D reconstruction, as demonstrated in the papers by Kim, Zimmer *et al.* (2013), Yücer, Kim *et al.* (2016), and Yücer, Sorkine-Hornung *et al.* (2016).

The equations mapping from pixels (x, y) in a virtual camera and the corresponding (s, t, u, v) coordinates are relatively straightforward to derive and are sketched out in Exercise 14.7. It is also possible to show that the set of pixels corresponding to a regular orthographic or perspective camera, i.e., one that has a linear projective relationship between 3D points and (x, y) pixels (2.63), lie along a two-dimensional hyperplane in the (s, t, u, v) light field (Exercise 14.7).

While a light field can be used to render a complex 3D scene from novel viewpoints, a much better rendering (with less ghosting) can be obtained if something is known about its 3D geometry. The Lumigraph system of Gortler, Grzeszczuk *et al.* (1996) extends the basic light field rendering approach by taking into account the 3D location of surface points corresponding to each 3D ray.

Consider the ray (s, u) corresponding to the dashed line in Figure 14.12, which intersects

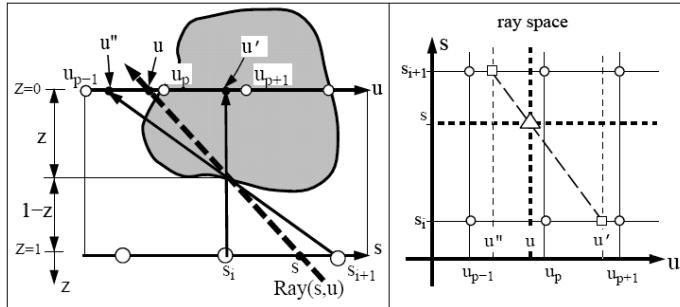


Figure 14.12 Depth compensation in the Lumigraph (Gortler, Grzeszczuk et al. 1996) © 1996 ACM. To resample the (s, u) dashed light ray, the u parameter corresponding to each discrete s_i camera location is modified according to the out-of-plane depth z to yield new coordinates u and u' ; in (u, s) ray space, the original sample (\triangle) is resampled from the (s_i, u') and (s_{i+1}, u'') samples, which are themselves linear blends of their adjacent (\circ) samples.

the object's surface at a distance z from the uv plane. When we look up the pixel's color in camera s_i (assuming that the light field is discretely sampled on a regular 4D (s, t, u, v) grid), the actual pixel coordinate is u' , instead of the original u value specified by the (s, u) ray. Similarly, for camera s_{i+1} (where $s_i \leq s \leq s_{i+1}$), pixel address u'' is used. Thus, instead of using quadri-linear interpolation of the nearest sampled (s, t, u, v) values around a given ray to determine its color, the (u, v) values are modified for each discrete (s_i, t_i) camera.

Figure 14.12 also shows the same reasoning in *ray space*. Here, the original continuous-valued (s, u) ray is represented by a triangle and the nearby sampled discrete values are shown as circles. Instead of just blending the four nearest samples, as would be indicated by the vertical and horizontal dashed lines, the modified (s_i, u') and (s_{i+1}, u'') values are sampled instead and their values are then blended.

The resulting rendering system produces images of much better quality than a proxy-free light field and is the method of choice whenever 3D geometry can be inferred. In subsequent work, Isaksen, McMillan, and Gortler (2000) show how a planar proxy for the scene, which is a simpler 3D model, can be used to simplify the resampling equations. They also describe how to create synthetic aperture photos, which mimic what might be seen by a wide-aperture lens, by blending more nearby samples (Levoy and Hanrahan 1996). A similar approach can be used to re-focus images taken with a plenoptic (microlens array) camera (Ng, Levoy et al. 2005; Ng 2005) or a light field microscope (Levoy, Ng et al. 2006). It can also be used to see through obstacles, using extremely large synthetic apertures focused on a background

that can blur out foreground objects and make them appear translucent (Wilburn, Joshi *et al.* 2005; Vaish, Szeliski *et al.* 2006).

Now that we understand how to render new images from a light field, how do we go about capturing such datasets? One answer is to move a calibrated camera with a motion control rig or *gantry*.¹¹ Another approach is to take handheld photographs and to determine the pose and intrinsic calibration of each image using either a calibrated stage or structure from motion. In this case, the images need to be *rebinned* into a regular 4D (s, t, u, v) space before they can be used for rendering (Gortler, Grzeszczuk *et al.* 1996). Alternatively, the original images can be used directly using a process called the *unstructured Lumigraph*, which we describe below.

Because of the large number of images involved, light fields and Lumigraphs can be quite voluminous to store and transmit. Fortunately, as you can tell from Figure 14.11b, there is a tremendous amount of redundancy (coherence) in a light field, which can be made even more explicit by first computing a 3D model, as in the Lumigraph. A number of techniques have been developed to compress and progressively transmit such representations (Gortler, Grzeszczuk *et al.* 1996; Levoy and Hanrahan 1996; Rademacher and Bishop 1998; Magnor and Girod 2000; Wood, Azuma *et al.* 2000; Shum, Kang, and Chan 2003; Magnor, Ramathan, and Girod 2003; Zhang and Chen 2004; Shum, Chan, and Kang 2007).

Since the original burst of research on lightfields in the mid-1990 and early 2000s, better techniques continue to be developed for analyzing and rendering such images. Some representative papers and datasets from the last decade include Wanner and Goldluecke (2014), Honauer, Johannsen *et al.* (2016), Kalantari, Wang, and Ramamoorthi (2016), Wu, Masia *et al.* (2017), and Shin, Jeon *et al.* (2018).

14.3.1 Unstructured Lumigraph

When the images in a Lumigraph are acquired in an unstructured (irregular) manner, it can be counterproductive to resample the resulting light rays into a regularly binned (s, t, u, v) data structure. This is both because resampling always introduces a certain amount of aliasing and because the resulting gridded light field can be populated very sparsely or irregularly.

The alternative is to render directly from the acquired images, by finding for each light ray in a virtual camera the closest pixels in the original images. The *unstructured Lumigraph* rendering (ULR) system of Buehler, Bosse *et al.* (2001) describes how to select such pixels

¹¹See <http://lightfield.stanford.edu/acq.html> for a description of some of the gantries and camera arrays built at the Stanford Computer Graphics Laboratory (Wilburn, Joshi *et al.* 2005). A more recent dataset was created by Honauer, Johannsen *et al.* (2016) and is available at <https://lightfield-analysis.uni-konstanz.de> Both websites provide light field datasets that are a great source of research and project material.

by combining a number of fidelity criteria, including *epipole consistency* (distance of rays to a source camera’s center), *angular deviation* (similar incidence direction on the surface), *resolution* (similar sampling density along the surface), *continuity* (to nearby pixels), and *consistency* (along the ray). These criteria can all be combined to determine a weighting function between each virtual camera’s pixel and a number of candidate input cameras from which it can draw colors. To make the algorithm more efficient, the computations are performed by discretizing the virtual camera’s image plane using a regular grid overlaid with the polyhedral object mesh model and the input camera centers of projection and interpolating the weighting functions between vertices.

The unstructured Lumigraph generalizes previous work in both image-based rendering and light field rendering. When the input cameras are gridded, the ULR behaves the same way as regular Lumigraph rendering. When fewer cameras are available but the geometry is accurate, the algorithm behaves similarly to view-dependent texture mapping (Section 14.1.1). If RGB-D depth images are available, these can be fused into lower-resolution proxies that can be combined with higher-resolution source images at rendering time (Hedman, Ritschel *et al.* 2016). And while the original ULR paper uses manually constructed rules for determining pixel weights, it is also possible to learn such blending weights using a deep neural network (Hedman, Philip *et al.* 2018; Riegler and Koltun 2020a).

14.3.2 Surface light fields

Of course, using a two-plane parameterization for a light field is not the only possible choice. (It is the one usually presented first, as the projection equations and visualizations are the easiest to draw and understand.) As we mentioned on the topic of light field compression, if we know the 3D shape of the object or scene whose light field is being modeled, we can effectively compress the field because nearby rays emanating from nearby surface elements have similar color values.

In fact, if the object is totally diffuse, ignoring occlusions, which can be handled using 3D graphics algorithms or z-buffering, all rays passing through a given surface point will have the same color value. Hence, the light field “collapses” to the usual 2D texture-map defined over an object’s surface. Conversely, if the surface is totally specular (e.g., mirrored), each surface point reflects a miniature copy of the environment surrounding that point. In the absence of inter-reflections (e.g., a convex object in a large open space), each surface point simply reflects the far-field *environment map* (Section 2.2.1), which again is two-dimensional. Therefore, it seems that re-parameterizing the 4D light field to lie on the object’s surface can be extremely beneficial.

These observations underlie the *surface light field* representation introduced by Wood,

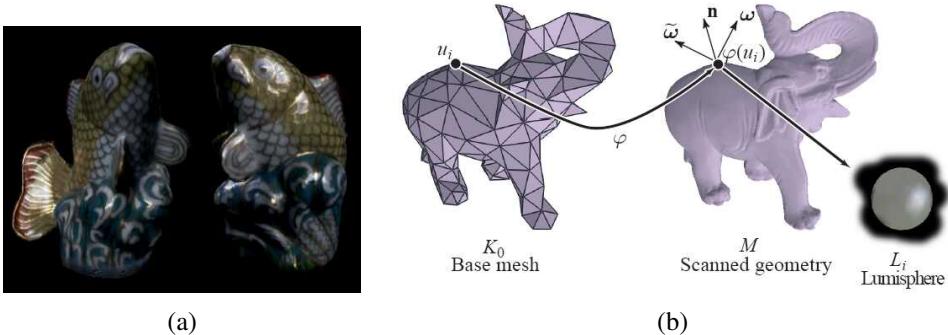


Figure 14.13 *Surface light fields* (Wood, Azuma et al. 2000) © 2000 ACM: (a) example of a highly specular object with strong inter-reflections; (b) the surface light field stores the light emanating from each surface point in all visible directions as a “Lumisphere”.

Azuma *et al.* (2000). In their system, an accurate 3D model is built of the object being represented. Then the *Lumisphere* of all rays emanating from each surface point is estimated or captured (Figure 14.13). Nearby Lumispheres will be highly correlated and hence amenable to both compression and manipulation.

To estimate the diffuse component of each Lumisphere, a median filtering over all visible exiting directions is first performed for each channel. Once this has been subtracted from the Lumisphere, the remaining values, which should consist mostly of the specular components, are *reflected* around the local surface normal (2.90), which turns each Lumisphere into a copy of the local environment around that point. Nearby Lumispheres can then be compressed using predictive coding, vector quantization, or principal component analysis.

The decomposition into a diffuse and specular component can also be used to perform editing or manipulation operations, such as re-painting the surface, changing the specular component of the reflection (e.g., by blurring or sharpening the specular Lumispheres), or even geometrically deforming the object while preserving detailed surface appearance.

In more recent work, Park, Newcombe, and Seitz (2018) use an RGB-D camera to acquire a 3D model and its diffuse reflectance layer using min compositing and iteratively reweighted least squares, as discussed in Section 9.4.2. They then estimate a simple piecewise-constant BRDF model to account for the specular components. In their follow-on Seeing the World in a Bag of Chips paper, Park, Holynski, and Seitz (2020) also estimate the *specular reflectance map*, which is a convolution of the environment map with the object’s specular BRDF. Additional techniques to estimate spatially varying BRDFs are discussed in Section 13.7.1.

In summary, surface light fields are a good representation to add realism to scanned 3D

object models by modeling their specular properties, thus avoiding the “cardboard” (matte) appearance of such models when their reflections are ignored. For larger scenes, especially those containing large planar reflectors such as glass windows or glossy tables, modeling the reflections as separate layers, as discussed in Sections 9.4.2 and 14.2.1, or as true mirror surfaces (Whelan, Goesele *et al.* 2018), may be more appropriate.

14.3.3 Application: Concentric mosaics

A useful and simple version of light field rendering is a panoramic image with parallax, i.e., a video or series of photographs taken from a camera swinging in front of some rotation point. Such panoramas can be captured by placing a camera on a boom on a tripod, or even more simply, by holding a camera at arm’s length while rotating your body around a fixed axis.

The resulting set of images can be thought of as a *concentric mosaic* (Shum and He 1999; Shum, Wang *et al.* 2002) or a *layered depth panorama* (Zheng, Kang *et al.* 2007). The term “concentric mosaic” comes from a particular structure that can be used to re-bin all of the sampled rays, essentially associating each column of pixels with the “radius” of the concentric circle to which it is tangent (Ishiguro, Yamamoto, and Tsuji 1992; Shum and He 1999; Peleg, Ben-Ezra, and Pritch 2001).

Rendering from such data structures is fast and straightforward. If we assume that the scene is far enough away, for any virtual camera location, we can associate each column of pixels in the virtual camera with the nearest column of pixels in the input image set. (For a regularly captured set of images, this computation can be performed analytically.) If we have some rough knowledge of the depth of such pixels, columns can be stretched vertically to compensate for the change in depth between the two cameras. If we have an even more detailed depth map (Peleg, Ben-Ezra, and Pritch 2001; Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007), we can perform pixel-by-pixel depth corrections.

While the virtual camera’s motion is constrained to lie in the plane of the original cameras and within the radius of the original capture ring, the resulting experience can exhibit complex rendering phenomena, such as reflections and translucencies, which cannot be captured using a texture-mapped 3D model of the world. Exercise 14.10 has you construct a concentric mosaic rendering system from a series of hand-held photos or video.

While concentric mosaics are captured by moving the camera on a (roughly) circular arc, it is also possible to construct *manifold projections* (Peleg and Herman 1997), *multiple-center-of-projection images* (Rademacher and Bishop 1998), and *multi-perspective panoramas* (Román, Garg, and Levoy 2004; Román and Lensch 2006; Agarwala, Agrawala *et al.* 2006; Kopf, Chen *et al.* 2010), which we discussed briefly in Section 8.2.5.

14.3.4 Application: Synthetic re-focusing

In addition to the interactive viewing of captured scenes and objects, light field rendering can be used to add synthetic depth of field effects to photographs (Levoy 2006). In the computational photography chapter (Section 10.3.2), we mentioned how the depth estimates produced by modern dual-lens and/or dual-pixel smartphones can be used to synthetically blur photographs (Wadhwa, Garg *et al.* 2018; Garg, Wadhwa *et al.* 2019; Zhang, Wadhwa *et al.* 2020).

When larger numbers of input images are available, e.g., when using microlens arrays, the images can be shifted and combined to simulate the effects of a larger aperture lens in what is known as *synthetic aperture photography* (Ng, Levoy *et al.* 2005; Ng 2005), which was the basis of the Lytro light field camera. Related ideas have been used for shallow depth of field in *light field microscopy* (Levoy, Chen *et al.* 2004; Levoy, Ng *et al.* 2006), obstruction removal (Wilburn, Joshi *et al.* 2005; Vaish, Szeliski *et al.* 2006; Xue, Rubinstein *et al.* 2015; Liu, Lai *et al.* 2020a), and *coded aperture photography* (Levin, Fergus *et al.* 2007; Zhou, Lin, and Nayar 2009).

14.4 Environment mattes

So far in this chapter, we have dealt with view interpolation and light fields, which are techniques for modeling and rendering complex static scenes seen from different viewpoints.

What if, instead of moving around a virtual camera, we take a complex, refractive object, such as the water goblet shown in Figure 14.14, and place it in front of a new background? Instead of modeling the 4D space of rays emanating from a scene, we now need to model how each pixel in our view of this object refracts incident light coming from its environment.

What is the intrinsic dimensionality of such a representation and how do we go about capturing it? Let us assume that if we trace a light ray from the camera at pixel (x, y) toward the object, it is reflected or refracted back out toward its environment at an angle (ϕ, θ) . If we assume that other objects and illuminants are sufficiently distant (the same assumption we made for surface light fields in Section 14.3.2), this 4D mapping $(x, y) \rightarrow (\phi, \theta)$ captures all the information between a refractive object and its environment. Zongker, Werner *et al.* (1999) call such a representation an *environment matte*, as it generalizes the process of object matting (Section 10.4) to not only cut and paste an object from one image into another but also take into account the subtle refractive or reflective interplay between the object and its environment.

Recall from Equations (3.8) and (10.29) that a foreground object can be represented by

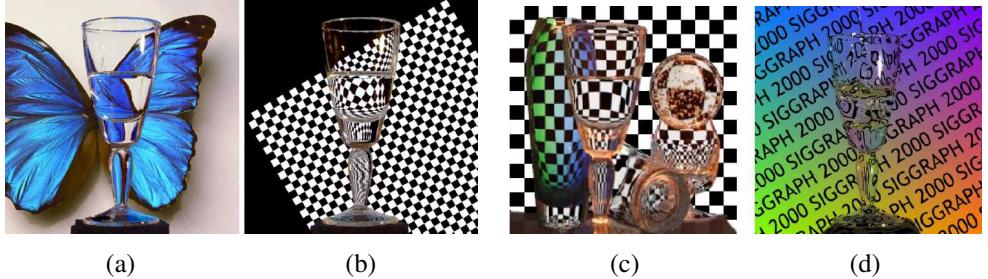


Figure 14.14 Environment mattes: (a–b) a refractive object can be placed in front of a series of backgrounds and their light patterns will be correctly refracted (Zongker, Werner et al. 1999) (c) multiple refractions can be handled using a Gaussian mixture model and (d) real-time mattes can be pulled using a single graded colored background (Chuang, Zongker et al. 2000) © 2000 ACM.

its premultiplied colors and opacities ($\alpha F, \alpha$). Such a matte can then be composited onto a new background B using

$$C_i = \alpha_i F_i + (1 - \alpha_i) B_i, \quad (14.1)$$

where i is the pixel under consideration. In environment matting, we augment this equation with a reflective or refractive term to model indirect light paths between the environment and the camera. In the original work of Zongker, Werner *et al.* (1999), this indirect component I_i is modeled as

$$I_i = R_i \int A_i(\mathbf{x}) B(\mathbf{x}) d\mathbf{x}, \quad (14.2)$$

where A_i is the rectangular *area of support* for that pixel, R_i is the colored reflectance or transmittance (for colored glossy surfaces or glass), and $B(\mathbf{x})$ is the background (environment) image, which is integrated over the area $A_i(\mathbf{x})$. In follow-on work, Chuang, Zongker *et al.* (2000) use a superposition of oriented Gaussians,

$$I_i = \sum_j R_{ij} \int G_{ij}(\mathbf{x}) B(\mathbf{x}) d\mathbf{x}, \quad (14.3)$$

where each 2D Gaussian

$$G_{ij}(\mathbf{x}) = G_{2D}(\mathbf{x}; \mathbf{c}_{ij}, \sigma_{ij}, \theta_{ij}) \quad (14.4)$$

is modeled by its center \mathbf{c}_{ij} , unrotated widths $\sigma_{ij} = (\sigma_{ij}^x, \sigma_{ij}^y)$, and orientation θ_{ij} .

Given a representation for an environment matte, how can we go about estimating it for a particular object? The trick is to place the object in front of a monitor (or surrounded by a set

of monitors), where we can change the illumination patterns $B(\mathbf{x})$ and observe the value of each composite pixel C_i .¹²

As with traditional two-screen matting (Section 10.4.1), we can use a variety of solid colored backgrounds to estimate each pixel’s foreground color $\alpha_i F_i$ and partial coverage (opacity) α_i . To estimate the area of support A_i in (14.2), Zongker, Werner *et al.* (1999) use a series of periodic horizontal and vertical solid stripes at different frequencies and phases, which is reminiscent of the structured light patterns used in active rangefinding (Section 13.2). For the more sophisticated Gaussian mixture model (14.3), Chuang, Zongker *et al.* (2000) sweep a series of narrow Gaussian stripes at four different orientations (horizontal, vertical, and two diagonals), which enables them to estimate multiple oriented Gaussian responses at each pixel.

Once an environment matte has been “pulled”, it is then a simple matter to replace the background with a new image $B(\mathbf{x})$ to obtain a novel composite of the object placed in a different environment (Figure 14.14a–c). The use of multiple backgrounds during the matting process, however, precludes the use of this technique with dynamic scenes, e.g., water pouring into a glass (Figure 14.14d). In this case, a single graded color background can be used to estimate a single 2D monochromatic displacement for each pixel (Chuang, Zongker *et al.* 2000).

14.4.1 Higher-dimensional light fields

As you can tell from the preceding discussion, an environment matte in principle maps every pixel (x, y) into a 4D distribution over light rays and is, hence, a six-dimensional representation. (In practice, each 2D pixel’s response is parameterized using a dozen or so parameters, e.g., $\{F, \alpha, B, R, A\}$, instead of a full mapping.) What if we want to model an object’s refractive properties from every potential point of view? In this case, we need a mapping from every incoming 4D light ray to every potential exiting 4D light ray, which is an 8D representation. If we use the same trick as with surface light fields, we can parameterize each surface point by its 4D BRDF to reduce this mapping back down to 6D, but this loses the ability to handle multiple refractive paths.

If we want to handle dynamic light fields, we need to add another temporal dimension. (Wenger, Gardner *et al.* (2005) gives a nice example of a dynamic appearance and illumination acquisition system.) Similarly, if we want a continuous distribution over wavelengths, this becomes another dimension.

¹²If we relax the assumption that the environment is distant, the monitor can be placed at several depths to estimate a depth-dependent mapping function (Zongker, Werner *et al.* 1999).

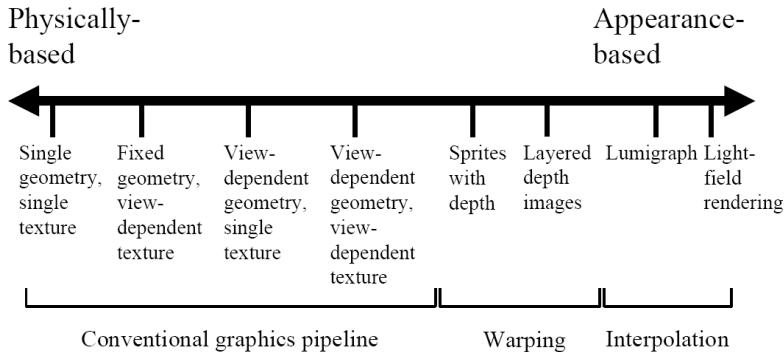


Figure 14.15 *The geometry–image continuum in image-based rendering (Kang, Szeliski, and Anandan 2000) © 2000 IEEE. Representations at the left of the spectrum use more detailed geometry and simpler image representations, while representations and algorithms on the right use more images and less geometry.*

These examples illustrate how modeling the full complexity of a visual scene through sampling can be extremely expensive. Fortunately, constructing specialized models, which exploit knowledge about the physics of light transport along with the natural coherence of real-world objects, can make these problems more tractable.

14.4.2 The modeling to rendering continuum

The image-based rendering representations and algorithms we have studied in this chapter span a continuum ranging from classic 3D texture-mapped models all the way to pure sampled ray-based representations such as light fields (Figure 14.15). Representations such as view-dependent texture maps and Lumigraphs still use a single global geometric model, but select the colors to map onto these surfaces from nearby images. View-dependent geometry, e.g., multiple depth maps, sidestep the need for coherent 3D geometry, and can sometimes better model local non-rigid effects such as specular motion (Swaminathan, Kang *et al.* 2002; Criminisi, Kang *et al.* 2005). Sprites with depth and layered depth images use image-based representations of both color and geometry and can be efficiently rendered using warping operations rather than 3D geometric rasterization.

The best choice of representation and rendering algorithm depends on both the quantity and quality of the input imagery as well as the intended application. When nearby views are being rendered, image-based representations capture more of the visual fidelity of the real world because they directly sample its appearance. On the other hand, if only a few input

images are available or the image-based models need to be manipulated, e.g., to change their shape or appearance, more abstract 3D representations such as geometric and local reflection models are a better fit. As we continue to capture and manipulate increasingly larger quantities of visual data, research into these aspects of image-based modeling and rendering will continue to evolve.

14.5 Video-based rendering

As multiple images can be used to render new images or interactive experiences, can something similar be done with video? In fact, a fair amount of work has been done in the area of *video-based rendering* and *video-based animation*, two terms first introduced by Schödl, Szeliski *et al.* (2000) to denote the process of generating new video sequences from captured video footage. An early example of such work is Video Rewrite (Bregler, Covell, and Slaney 1997), in which archival video footage is “re-animated” by having actors say new utterances (Figure 14.16). More recently, the term video-based rendering has been used by some researchers to denote the creation of virtual camera moves from a set of synchronized video cameras placed in a studio (Magnor 2005). (The terms *free-viewpoint video* and *3D video* are also sometimes used: see Section 14.5.4.)

In this section, we present a number of video-based rendering systems and applications. We start with *video-based animation* (Section 14.5.1), in which video footage is re-arranged or modified, e.g., in the capture and re-rendering of facial expressions. A special case of this is *video textures* (Section 14.5.2), in which source video is automatically cut into segments and re-looped to create infinitely long video animations. It is also possible to create such animations from still pictures or paintings, by segmenting the image into separately moving regions and animating them using stochastic motion fields (Section 14.5.3).

Next, we turn our attention to *3D video* (Section 14.5.4), in which multiple synchronized video cameras are used to film a scene from different directions. The source video frames can then be re-combined using image-based rendering techniques, such as view interpolation, to create virtual camera paths between the source cameras as part of a real-time viewing experience. Finally, we discuss capturing environments by driving or walking through them with panoramic video cameras to create interactive video-based walkthrough experiences (Section 14.5.5).



Figure 14.16 Video Rewrite (Bregler, Covell, and Slaney 1997) © 1997 ACM: the video frames are composed from bits and pieces of old video footage matched to a new audio track.

14.5.1 Video-based animation

As we mentioned above, an early example of video-based animation is Video Rewrite, in which frames from original video footage are rearranged to match them to novel spoken utterances, e.g., for movie dubbing (Figure 14.16). This is similar in spirit to the way that *concatenative speech synthesis* systems work (Taylor 2009).

In their system, Bregler, Covell, and Slaney (1997) first use speech recognition to extract phonemes from both the source video material and the novel audio stream. Phonemes are grouped into *triphones* (triplets of phonemes), as these better model the *coarticulation* effect present when people speak. Matching triphones are then found in the source footage and audio track. The mouth images corresponding to the selected video frames are then cut and pasted into the desired video footage being re-animated or dubbed, with appropriate geometric transformations to account for head motion. During the analysis phase, features corresponding to the lips, chin, and head are tracked using computer vision techniques. During synthesis, image morphing techniques are used to blend and stitch adjacent mouth shapes into a more coherent whole. In subsequent work, Ezzat, Geiger, and Poggio (2002) describe how to use a *multidimensional morphable model* (Section 13.6.2) combined with regularized trajectory synthesis to improve these results.

A more sophisticated version of this system, called *face transfer*, uses a novel source video, instead of just an audio track, to drive the animation of a previously captured video, i.e., to re-render a video of a talking head with the appropriate visual speech, expression, and head pose elements (Vlasic, Brand *et al.* 2005). This work is one of many *performance-driven animation* systems (Section 7.1.6), which are often used to animate 3D facial models (Figures 13.23–13.25). While traditional performance-driven animation systems use marker-based motion capture (Williams 1990; Litwinowicz and Williams 1994; Ma, Jones *et al.* 2008), video footage can now be used directly to control the animation (Buck, Finkelstein *et al.* 2000; Pighin, Szeliski, and Salesin 2002; Zhang, Snavely *et al.* 2004; Vlasic, Brand *et al.*

al. 2005; Roble and Zafar 2009; Thies, Zollhofer *et al.* 2016; Thies, Zollhöfer *et al.* 2018; Zollhöfer, Thies *et al.* 2018; Fried, Tewari *et al.* 2019; Egger, Smith *et al.* 2020; Tewari, Fried *et al.* 2020). More details on related techniques can also be found in Section 13.6.3 on facial animation and Section 14.6 on neural rendering.

In addition to its most common application to facial animation, video-based animation can also be applied to whole body motion (Section 13.6.4), e.g., by matching the flow fields between two different source videos and using one to drive the other (Efros, Berg *et al.* 2003; Wang, Liu *et al.* 2018; Chan, Ginosar *et al.* 2019). Another approach to video-based rendering is to use flow or 3D modeling to *unwrap* surface textures into stabilized images, which can then be manipulated and re-rendered onto the original video (Pighin, Szeliski, and Salesin 2002; Rav-Acha, Kohli *et al.* 2008).

14.5.2 Video textures

Video-based animation is a powerful means of creating photo-realistic videos by re-purposing existing video footage to match some other desired activity or script. What if, instead of constructing a special animation or narrative, we simply want the video to continue playing in a plausible manner? For example, many websites use images or videos to highlight their destinations, e.g., to portray attractive beaches with surf and palm trees waving in the wind. Instead of using a static image or a video clip that has a discontinuity when it loops, can we transform the video clip into an infinite-length animation that plays forever?

This idea is the basis of *video textures*, in which a short video clip can be arbitrarily extended by re-arranging video frames while preserving visual continuity (Schödl, Szeliski *et al.* 2000). The basic problem in creating video textures is how to perform this re-arrangement without introducing visual artifacts. Can you think of how you might do this?

The simplest approach is to match frames by visual similarity (e.g., L_2 distance) and to jump between frames that appear similar. Unfortunately, if the motions in the two frames are different, a dramatic visual artifact will occur (the video will appear to “stutter”). For example, if we fail to match the motions of the clock pendulum in Figure 14.17a, it can suddenly change direction in mid-swing.

How can we extend our basic frame matching to also match motion? In principle, we could compute optical flow at each frame and match this. However, flow estimates are often unreliable (especially in textureless regions) and it is not clear how to weight the visual and motion similarities relative to each other. As an alternative, Schödl, Szeliski *et al.* (2000) suggest matching *triplets* or larger neighborhoods of adjacent video frames, much in the same way as Video Rewrite matches triphones. Once we have constructed an $n \times n$ similarity matrix between all video frames (where n is the number of frames), a simple finite impulse



Figure 14.17 Video textures (Schödl, Szeliski et al. 2000) © 2000 ACM: (a) a clock pendulum, with correctly matched direction of motion; (b) a candle flame, showing temporal transition arcs; (c) the flag is generated using morphing at jumps; (d) a bonfire uses longer cross-dissolves; (e) a waterfall cross-dissolves several sequences at once; (f) a smiling animated face; (g) two swinging children are animated separately; (h) the balloons are automatically segmented into separate moving regions; (i) a synthetic fish tank consisting of bubbles, plants, and fish. Videos corresponding to these images can be found at <https://www.cc.gatech.edu/gvu/perception/projects/videotexture>.

response (FIR) filtering of each match sequence can be used to emphasize subsequences that match well.

The results of this match computation gives us a *jump table* or, equivalently, a transition probability between any two frames in the original video. This is shown schematically as red arcs in Figure 14.17b, where the red bar indicates which video frame is currently being displayed, and arcs light up as a forward or backward transition is taken. We can view these transition probabilities as encoding the *hidden Markov model* (HMM) that underlies a stochastic video generation process.

Sometimes, it is not possible to find exactly matching subsequences in the original video. In this case, morphing, i.e., warping and blending frames during transitions (Section 3.6.3) can be used to hide the visual differences (Figure 14.17c). If the motion is chaotic enough, as in a bonfire or a waterfall (Figures 14.17d–e), simple blending (extended cross-dissolves) may be sufficient. Improved transitions can also be obtained by performing 3D graph cuts on the spatio-temporal volume around a transition (Kwatra, Schödl *et al.* 2003).

Video textures need not be restricted to chaotic random phenomena such as fire, wind, and water. Pleasing video textures can be created of people, e.g., a smiling face (as in Figure 14.17f) or someone running on a treadmill (Schödl, Szeliski *et al.* 2000). When multiple people or objects are moving independently, as in Figures 14.17g–h, we must first segment the video into independently moving regions and animate each region separately. It is also possible to create large panoramic video textures from a slowly panning camera (Agarwala, Zheng *et al.* 2005; He, Liao *et al.* 2017).

Instead of just playing back the original frames in a stochastic (random) manner, video textures can also be used to create scripted or interactive animations. If we extract individual elements, such as fish in a fishtank (Figure 14.17i) into separate *video sprites*, we can animate them along prespecified paths (by matching the path direction with the original sprite motion) to make our video elements move in a desired fashion (Schödl and Essa 2002). A more recent example of controlling video sprites is the Vid2Player system, which models the movements and shots of tennis players to create synthetic video-realistic games (Zhang, Sciutto *et al.* 2021). In fact, work on video textures inspired research on systems that re-synthesize new motion sequences from motion capture data, which some people refer to as “mocap soup” (Arikan and Forsyth 2002; Kovar, Gleicher, and Pighin 2002; Lee, Chai *et al.* 2002; Li, Wang, and Shum 2002; Pullen and Bregler 2002).

While video textures primarily analyze the video as a sequence of frames (or regions) that can be re-arranged in time, *temporal textures* (Szummer and Picard 1996; Bar-Joseph, El-Yaniv *et al.* 2001) and *dynamic textures* (Doretto, Chiuso *et al.* 2003; Yuan, Wen *et al.* 2004; Doretto and Soatto 2006) treat the video as a 3D spatio-temporal volume with textural

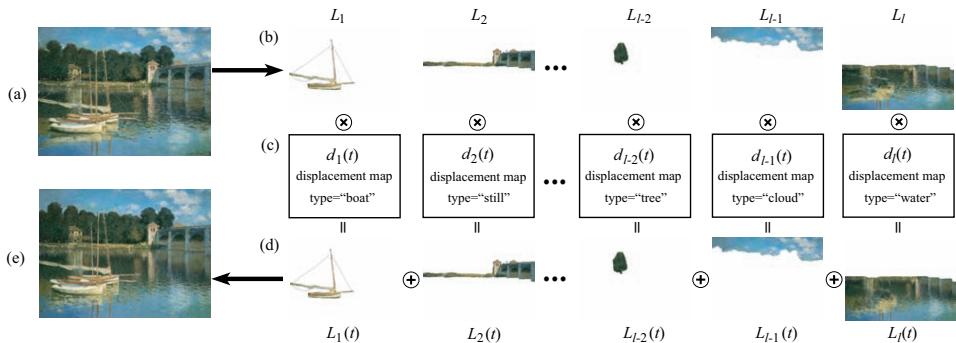


Figure 14.18 *Animating still pictures (Chuang, Goldman et al. 2005) © 2005 ACM.* (a) The input still image is manually segmented into (b) several layers. (c) Each layer is then animated with a different stochastic motion texture (d) The animated layers are then composited to produce (e) the final animation

properties, which can be described using auto-regressive temporal models and combined with layered representations (Chan and Vasconcelos 2009). In more recent work, video texture authoring systems have been extended to allow for control over the *dynamism* (amount of motion) in different regions (Joshi, Mehta et al. 2012; Liao, Joshi, and Hoppe 2013; Yan, Liu, and Furukawa 2017; He, Liao et al. 2017; Oh, Joo et al. 2017) and improved loop transitions (Liao, Finch, and Hoppe 2015).

14.5.3 Application: Animating pictures

While video textures can turn a short video clip into an infinitely long video, can the same thing be done with a single still image? The answer is yes, if you are willing to first segment the image into different layers and then animate each layer separately.

Chuang, Goldman et al. (2005) describe how an image can be decomposed into separate layers using interactive matting techniques. Each layer is then animated using a class-specific synthetic motion. As shown in Figure 14.18, boats rock back and forth, trees sway in the wind, clouds move horizontally, and water ripples, using a shaped noise displacement map. All of these effects can be tied to some global control parameters, such as the velocity and direction of a virtual wind. After being individually animated, the layers can be composited to create a final dynamic rendering.

In more recent work, Holynski, Curless et al. (2021) train a deep network to take a static photo, hallucinate a plausible motion field, encode the image as deep multi-resolution fea-

tures, and then advect these features bi-directionally in time using Eulerian motion, using an architecture inspired by [Niklaus and Liu \(2020\)](#) and [Wiles, Gkioxari *et al.* \(2020\)](#). The resulting deep features are then decoded to produce a looping video clip with synthetic stochastic fluid motions.

14.5.4 3D and free-viewpoint Video

In the last decade, the 3D movies have become an established medium. Currently, such releases are filmed using stereoscopic camera rigs and displayed in theaters (or at home) to viewers wearing polarized glasses. In the future, however, home audiences may wish to view such movies with multi-zone auto-stereoscopic displays, where each person gets his or her own customized stereo stream and can move around a scene to see it from different perspectives.

The stereo matching techniques developed in the computer vision community along with image-based rendering (view interpolation) techniques from graphics are both essential components in such scenarios, which are sometimes called *free-viewpoint video* ([Carranza, Theobalt *et al.* 2003](#)) or *virtual viewpoint video* ([Zitnick, Kang *et al.* 2004](#)). In addition to solving a series of per-frame reconstruction and view interpolation problems, the depth maps or proxies produced by the analysis phase must be temporally consistent in order to avoid flickering artifacts. Neural rendering techniques ([Tewari, Fried *et al.* 2020](#), Section 6.3) can also be used for both the reconstruction and rendering phases.

[Shum, Chan, and Kang \(2007\)](#) and [Magnor \(2005\)](#) present nice overviews of various video view interpolation techniques and systems. These include the Virtualized Reality system of [Kanade, Rander, and Narayanan \(1997\)](#) and [Vedula, Baker, and Kanade \(2005\)](#), Immersive Video ([Moezzi, Katkere *et al.* 1996](#)), Image-Based Visual Hulls ([Matusik, Buehler *et al.* 2000](#); [Matusik, Buehler, and McMillan 2001](#)), and Free-Viewpoint Video ([Carranza, Theobalt *et al.* 2003](#)), which all use global 3D geometric models (surface-based (Section 13.3) or volumetric (Section 13.5)) as their proxies for rendering. The work of [Vedula, Baker, and Kanade \(2005\)](#) also computes *scene flow*, i.e., the 3D motion between corresponding surface elements, which can then be used to perform spatio-temporal interpolation of the multi-view video stream. A more recent variant of scene flow is the *occupancy flow* work of [Niemeyer, Mescheder *et al.* \(2019\)](#).

The Virtual Viewpoint Video system of [Zitnick, Kang *et al.* \(2004\)](#), on the other hand, associates a two-layer depth map with each input image, which allows them to accurately model occlusion effects such as the mixed pixels that occur at object boundaries. Their system, which consists of eight synchronized video cameras connected to a disk array (Figure 14.19a), first uses segmentation-based stereo to extract a depth map for each input image

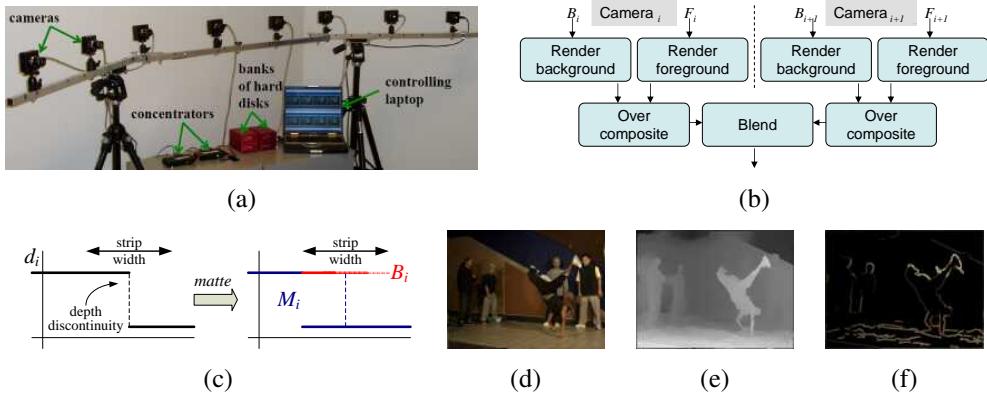


Figure 14.19 Video view interpolation (Zitnick, Kang et al. 2004) © 2004 ACM: (a) the capture hardware consists of eight synchronized cameras; (b) the background and foreground images from each camera are rendered and composited before blending; (c) the two-layer representation, before and after boundary matting; (d) background color estimates; (e) background depth estimates; (f) foreground depth estimates.

(Figure 14.19e). Near object boundaries (depth discontinuities), the background layer is extended along a strip behind the foreground object (Figure 14.19c) and its color is estimated from the neighboring images where it is not occluded (Figure 14.19d). Automated matting techniques (Section 10.4) are then used to estimate the fractional opacity and color of boundary pixels in the foreground layer (Figure 14.19f).

At render time, given a new virtual camera that lies between two of the original cameras, the layers in the neighboring cameras are rendered as texture-mapped triangles and the foreground layer (which may have fractional opacities) is then composited over the background layer (Figure 14.19b). The resulting two images are merged and blended by comparing their respective z-buffer values. (Whenever the two z-values are sufficiently close, a linear blend of the two colors is computed.) The interactive rendering system runs in real time using regular graphics hardware. It can therefore be used to change the observer's viewpoint while playing the video or to freeze the scene and explore it in 3D. Rogmans, Lu et al. (2009) subsequently developed GPU implementations of both real-time stereo matching and real-time rendering algorithms, which enable them to explore algorithmic alternatives in a real-time setting.

The depth maps computed from the eight stereo cameras using off-line stereo matching have been used in studies of 3D video compression (Smolic and Kauff 2005; Gotchev and Rosenhahn 2009; Tech, Chen et al. 2015). Active video-rate depth sensing cameras, such as the 3DV Zcam (Iddan and Yahav 2001), which we discussed in Section 13.2.1, are another

potential source of such data.

When large numbers of closely spaced cameras are available, as in the Stanford Light Field Camera (Wilburn, Joshi *et al.* 2005), it may not always be necessary to compute explicit depth maps to create video-based rendering effects, although the results are usually of higher quality if you do (Vaish, Szeliski *et al.* 2006).

The last few years have seen a revival of research into 3D video, spurred in part by the wider availability of virtual reality headsets, which can be used to view such videos with a strong sense of immersion. The Jump virtual reality capture system from Google (Anderson, Gallup *et al.* 2016) uses 16 GoPro cameras arranged on a 28cm diameter ring to capture multiple videos, which are then stitched offline into a pair of omnidirectional stereo (ODS) videos (Ishiguro, Yamamoto, and Tsuji 1992; Peleg, Ben-Ezra, and Pritch 2001; Richardt, Pritch *et al.* 2013), which can then be warped at viewing time to produce separate images for each eye. A similar system, constructed from tightly synchronized industrial vision cameras, was introduced around the same time by Cabral (2016).

As noted by Anderson, Gallup *et al.* (2016), however, the ODS representation has severe limitations in interactive viewing, e.g., it does not support head tilt, or translational motion, or produce correct depth when looking up or down. More recent systems developed by Serrano, Kim *et al.* (2019), Parra Pozo, Toksvig *et al.* (2019), and Broxton, Flynn *et al.* (2020) support full 6DoF (six degrees of freedom) video, which allows viewers to move within a bounded volume while producing perspectively correct images for each eye. However, they require multi-view stereo matching during the offline construction phase to produce the 3D proxies need to support such viewing.

While these systems are designed to capture *inside out* experiences, where a user can watch a video unfolding all around them, pointing the cameras *outside in* can be used to capture one or more actors performing an activity (Kanade, Rander, and Narayanan 1997; Joo, Liu *et al.* 2015; Tang, Dou *et al.* 2018). Such setups are often called *free-viewpoint video* or *volumetric performance capture* systems. The most recent versions of such systems use deep networks to reconstruct, represent, compress, and/or render time-evolving volumetric scenes (Martin-Brualla, Pandey *et al.* 2018; Pandey, Tkach *et al.* 2019; Lombardi, Simon *et al.* 2019; Tang, Singh *et al.* 2020; Peng, Zhang *et al.* 2021), as summarized in the recent survey on neural rendering by Tewari, Fried *et al.* (2020, Section 6.3). And while most of these systems require custom-built multi-camera rigs, it is also possible to construct 3D videos from collections of handheld videos (Bansal, Vo *et al.* 2020) or even a single moving smartphone camera (Yoon, Kim *et al.* 2020; Luo, Huang *et al.* 2020).

14.5.5 Application: Video-based walkthroughs

Video camera arrays enable the simultaneous capture of 3D dynamic scenes from multiple viewpoints, which can then enable the viewer to explore the scene from viewpoints near the original capture locations. What if, instead we wish to capture an extended area, such as a home, a movie set, or even an entire city?

In this case, it makes more sense to move the camera through the environment and play back the video as an interactive video-based walkthrough. To allow the viewer to look around in all directions, it is preferable to use a panoramic video camera (Uyttendaele, Criminisi *et al.* 2004).¹³

One way to structure the acquisition process is to capture these images in a 2D horizontal plane, e.g., over a grid superimposed inside a room. The resulting *sea of images* (Aliaga, Funkhouser *et al.* 2003) can be used to enable continuous motion between the captured locations.¹⁴ However, extending this idea to larger settings, e.g., beyond a single room, can become tedious and data-intensive.

Instead, a natural way to explore a space is often to just walk through it along some prespecified paths, just as museums or home tours guide users along a particular path, say down the middle of each room.¹⁵ Similarly, city-level exploration can be achieved by driving down the middle of each street and allowing the user to branch at each intersection. This idea dates back to the Aspen MovieMap project (Lippman 1980), which recorded analog video taken from moving cars onto videodiscs for later interactive playback.

Improvements in video technology enabled the capture of panoramic (spherical) video using a small co-located array of cameras, such as the Point Grey Ladybug camera (Figure 14.20b) developed by Uyttendaele, Criminisi *et al.* (2004) for their interactive video-based walkthrough project. In their system, the synchronized video streams from the six cameras (Figure 14.20a) are stitched together into 360° panoramas using a variety of techniques developed specifically for this project.

Because the cameras do not share the same center of projection, parallax between the cameras can lead to ghosting in the overlapping fields of view (Figure 14.20c). To remove this, a multi-perspective plane sweep stereo algorithm is used to estimate per-pixel depths at each column in the overlap area. To calibrate the cameras relative to each other, the camera is spun in place and a constrained structure from motion algorithm (Figure 11.15) is used to

¹³See <https://www.cis.upenn.edu/~kostas/omni.html> for descriptions of panoramic (omnidirectional) vision systems and associated workshops.

¹⁴The Photo Tourism system of Snavely, Seitz, and Szeliski (2006) applies this idea to less structured collections.

¹⁵In computer games, restricting a player to forward and backward motion along predetermined paths is called *rail-based gaming*.

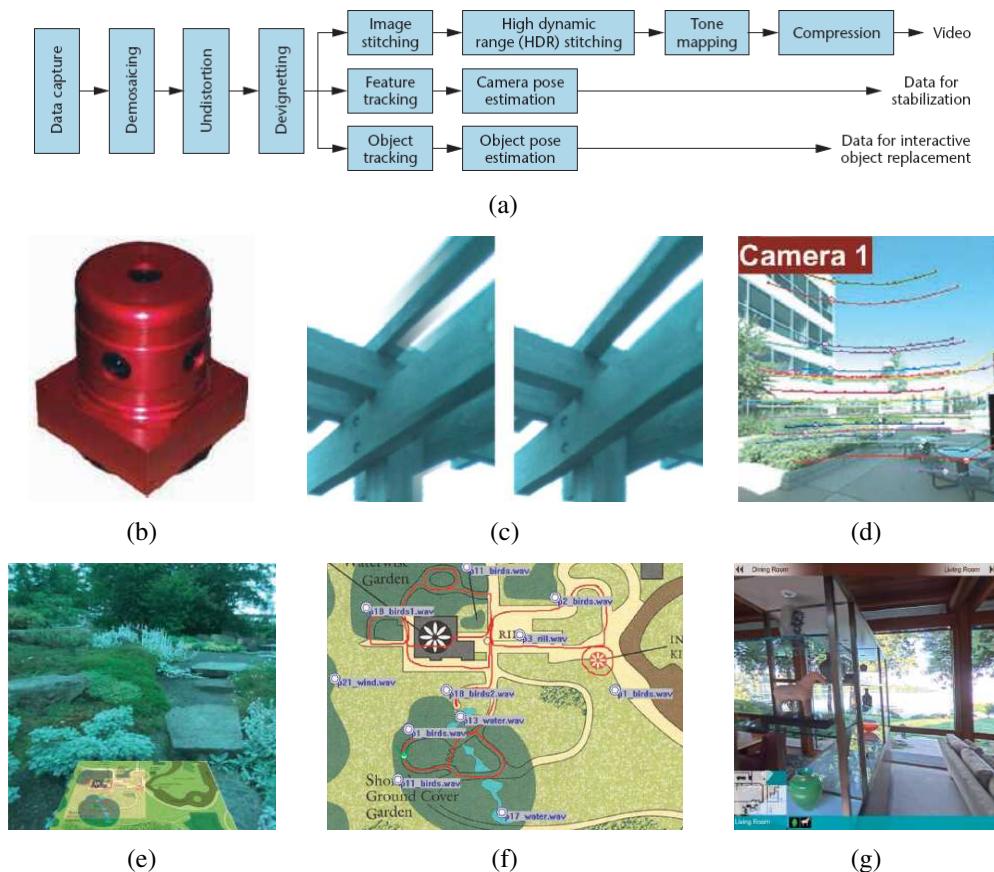


Figure 14.20 Video-based walkthroughs (Uyttendaele, Criminisi et al. 2004) © 2004 IEEE:
 (a) system diagram of video pre-processing; (b) the Point Grey Ladybug camera; (c) ghost removal using multi-perspective plane sweep; (d) point tracking, used both for calibration and stabilization; (e) interactive garden walkthrough with map below; (f) overhead map authoring and sound placement; (g) interactive home walkthrough with navigation bar (top) and icons of interest (bottom).

estimate the relative camera poses and intrinsics. Feature tracking is then run on the walk-through video to stabilize the video sequence. Liu, Gleicher *et al.* (2009), Kopf, Cohen, and Szeliski (2014), and Kopf (2016) have carried out more recent work along these lines.

Indoor environments with windows, as well as sunny outdoor environments with strong shadows, often have a dynamic range that exceeds the capabilities of video sensors. For this reason, the Ladybug camera has a programmable exposure capability that enables the bracketing of exposures at subsequent video frames. To merge the resulting video frames into high dynamic range (HDR) video, pixels from adjacent frames need to be motion-compensated before being merged (Kang, Uyttendaele *et al.* 2003).

The interactive walk-through experience becomes much richer and more navigable if an overview map is available as part of the experience. In Figure 14.20f, the map has annotations, which can show up during the tour, and localized sound sources, which play (with different volumes) when the viewer is nearby. The process of aligning the video sequence with the map can be automated using a process called *map correlation* (Levin and Szeliski 2004).

All of these elements combine to provide the user with a rich, interactive, and immersive experience. Figure 14.20e shows a walk through the Bellevue Botanical Gardens, with an overview map in perspective below the live video window. Arrows on the ground are used to indicate potential directions of travel. The viewer simply orients their view towards one of the arrows (the experience can be driven using a game controller) and “walks” forward along the desired path.

Figure 14.20g shows an indoor home tour experience. In addition to a schematic map in the lower left corner and adjacent room names along the top navigation bar, icons appear along the bottom whenever items of interest, such as a homeowner’s art pieces, are visible in the main window. These icons can then be clicked to provide more information and 3D views.

The development of interactive video tours spurred a renewed interest in 360° video-based virtual travel and mapping experiences, as evidenced by commercial sites such as Google’s Street View and 360cities. The same videos can also be used to generate turn-by-turn driving directions, taking advantage of both expanded fields of view and image-based rendering to enhance the experience (Chen, Neubert *et al.* 2009).

While initially, 360° cameras were exotic and expensive, they have more recently become widely available consumer products, such as the popular RICOH THETA camera, first introduced in 2013, and the GoPro MAX action camera. When shooting 360° videos, it is possible to stabilize the video using algorithms tailored to such videos (Kopf 2016) or proprietary algorithms based on the camera’s IMU readings. And while most of these cameras produce monocular photos and videos, VR180 cameras have two lenses and so can create

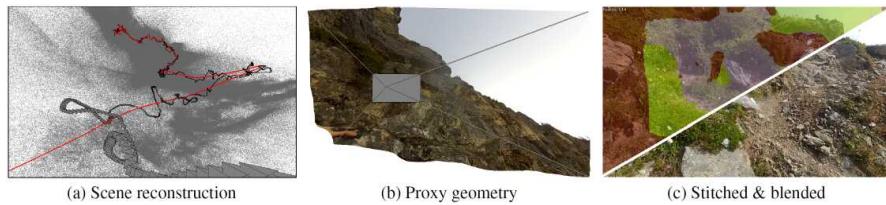


Figure 14.21 *First-person hyperlapse video creation (Kopf, Cohen, and Szeliski 2014)* © 2014 ACM: (a) 3D camera path and point cloud recovery, followed by smooth path planning; (b) 3D per-camera proxy estimation; and (c) source frame and seam selection using an MRF and Poisson blending.

wide field-of-view stereoscopic content. It is even possible to produce 3D 360° content by carefully stitching and transforming two 360° camera streams (Matzen, Cohen *et al.* 2017).

In addition to capturing immersive photos and videos of scenic locations and popular events, 360° and regular action cameras can also be worn, moved through an environment, and then sped up to create *hyperlapse* videos (Kopf, Cohen, and Szeliski 2014). Because such videos may exhibit large amounts of translational motion and parallax when heavily sped up, it is insufficient to simply compensate for camera rotations or even to warp individual input frames, because the large amounts of compensating motion may force the virtual camera to look outside the video frames. Instead, after constructing a sparse 3D model and smoothing the camera path, keyframes are selected and 3D proxies are computed for each of these by interpolating the sparse 3D point cloud, as shown in Figure 14.21. These frames are then warped and stitched together (using Poisson blending) using a Markov random field to ensure as much smoothness and visual continuity as possible. This system combines many different previously developed 3D modeling, computational photography, and image-based rendering algorithms to produce remarkably smooth high-speed tours of large-scale environments (such as cities) and activities (such as rock climbing and skiing).

As we continue to capture more and more of our real world with large amounts of high-quality imagery and video, the interactive modeling, exploration, and rendering techniques described in this chapter will play an even bigger role in bringing virtual experiences based in remote areas of the world as well as re-living special memories closer to everyone.

14.6 Neural rendering

The most recent development in image-based rendering is the introduction of deep neural networks into both the modeling (construction) and viewing parts of image-based rendering

pipelines. Neural rendering has been applied in a number of different domains, including style and texture manipulation and 2D semantic photo synthesis (Sections 5.5.4 and 10.5.3), 3D object shape and appearance modeling (Section 13.5.1), facial animation and reenactment (Section 13.6.3), 3D body capture and replay (Section 13.6.4), novel view synthesis (Section 14.1), free-viewpoint video (Section 14.5.4), and relighting (Duchêne, Riant *et al.* 2015; Meka, Haene *et al.* 2019; Philip, Gharbi *et al.* 2019; Sun, Barron *et al.* 2019; Zhou, Hadap *et al.* 2019; Zhang, Barron *et al.* 2020).

A comprehensive survey of all of these applications and techniques can be found in the state of the art report by Tewari, Fried *et al.* (2020), whose abstract states:

Neural rendering is a new and rapidly emerging field that combines generative machine learning techniques with physical knowledge from computer graphics, e.g., by the integration of differentiable rendering into network training. With a plethora of applications in computer graphics and vision, neural rendering is poised to become a new area in the graphics community...

The survey contains over 230 references and highlights 46 representative papers, grouped into six general categories, namely semantic photo synthesis, novel view synthesis, free viewpoint video, relighting, facial reenactment, and body reenactment. As you can tell, these categories overlap with the sections of the book mentioned in the previous paragraph. A set of lectures based on this content can be found in the related CVPR tutorial on neural rendering (Tewari, Zollhöfer *et al.* 2020), and several of the lectures in the TUM AI Guest Lecture Series are also on neural rendering research.¹⁶ The X-Fields paper by Bemana, Myszkowski *et al.* (2020, Table 1) also has a nice tabulation of related space, time, and illumination interpolation papers with an emphasis on deep methods, while the short bibliography by Dellaert and Yen-Chen (2021) summarizes even more recent techniques. Some neural rendering systems are implemented using differentiable rendering, which is surveyed by Kato, Beker *et al.* (2020).

As we have already seen many of these neural rendering techniques in the previous sections mentioned above, we focus here on their application to 3D image-based modeling and rendering. There are many ways to organize the last few years' worth of research in neural rendering. In this section, I have chosen to use four broad categories of underlying 3D representations, which we have studied in the last two chapters, namely: texture-mapped meshes, depth images and layers, volumetric grids, and implicit functions.

Texture-mapped meshes. As described in Chapter 13, a convenient representation for modeling and rendering a 3D scene is a triangle mesh, which can be reconstructed from

¹⁶<https://niessner.github.io/TUM-AI-Lecture-Series>

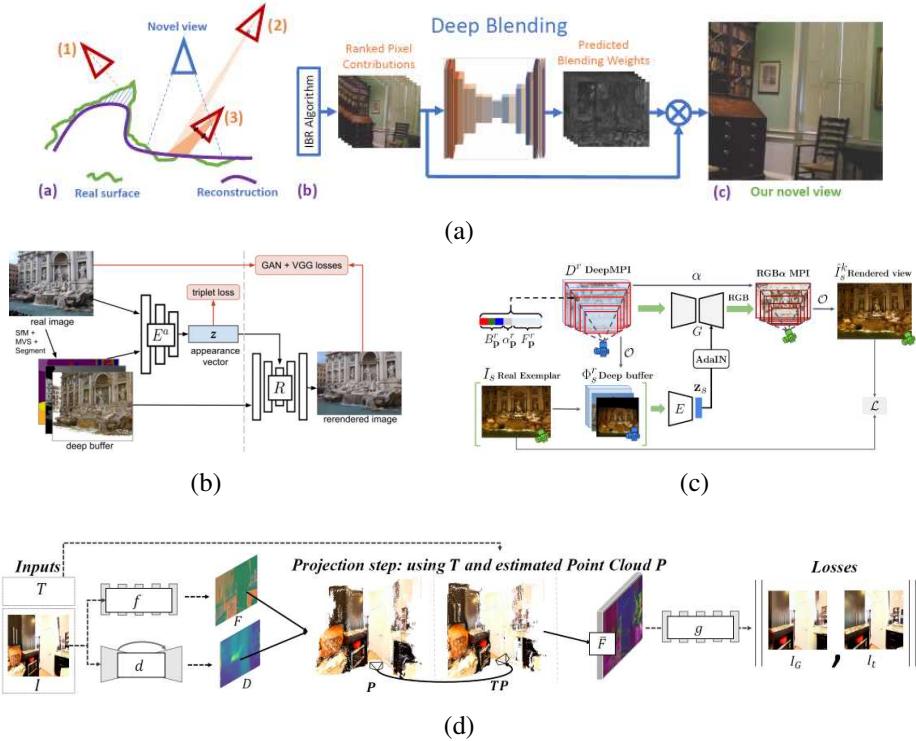


Figure 14.22 Examples of neural image-based rendering: (a) deep blending of depth-warped source images (Hedman, Philip et al. 2018) © 2018 ACM; (b) neural re-rendering in the wild with controllable view and lighting (Meshry, Goldman et al. 2019) © 2019 IEEE; (c) crowdsampling the plenoptic function with a deep MPI (Li, Xian et al. 2020) © 2020 Springer. (d) SynSin: novel view synthesis from a single image (Wiles, Gkioxari et al. 2020) © 2020 IEEE.

images using multi-view stereo. One of the earliest papers to use a neural network as part of the 3D rendering process was the deep blending system of [Hedman, Philip et al. \(2018\)](#), who augment an unstructured Lumigraph rendering pipeline ([Buehler, Bosse et al. 2001](#)) with a deep neural network that computes the per-pixel blending weights for the warped images selected for each novel view, as shown in Figure 14.22a. LookinGood ([Martin-Brualla, Pandey et al. 2018](#)) takes a single or multiple-image texture-mapped 3D upper or whole-body rendering and fills in the holes, denoises the appearance, and increases the resolution using a U-Net trained on held out views. Along a similar line, Deep Learning Super Sampling (DLSS) uses an encoder-decoder DNN implemented in GPU hardware to increase the resolution of rendered games in real time ([Burnes 2020](#)).

While these systems warp colored textures or images (i.e., view-dependent textures) and then apply a neural net post-process, it is also possible to first convert the images into a “neural” encoding and then warp and blend such representations. Free View Synthesis ([Riegler and Koltun 2020a](#)) starts by building a local 3D model for the novel view using multi-view stereo. It then encodes the source images as neural codes, reprojects these codes to the novel viewpoint, and composites them using a recurrent neural network and softmax. Instead of warping neural codes at render time and then blending and decoding them, the follow-on Stable View Synthesis system ([Riegler and Koltun 2020b](#)) collects neural codes from all incoming rays for every surface point and then combines these with an *on-surface aggregation* network to produce outgoing neural codes along the rays to the novel view camera. Deferred Neural Rendering ([Thies, Zollhöfer, and Nießner 2019](#)) uses a (u, v) parameterization over the 3D surface to learn and store a 2D texture map of neural codes, which can be sampled and decoded at rendering time.

Depth images and layers. To deal with images taken at different times of day and weather, i.e., “in the wild”, [Meshry, Goldman et al. \(2019\)](#) use a DNN to compute a latent “appearance” vector for each input image and its associated depth image (computed using traditional multi-view stereo), as shown in Figure 14.22b. At render time, the appearance can be manipulated (in addition to the 3D viewpoint) to explore the range of conditions under which the images were taken. [Li, Xian et al. \(2020\)](#) develop a related pipeline (Figure 14.22c), which instead of storing a single “deep” color/depth/appearance image or buffer uses a multiplane image (MPI). As with the previous system, an encoder-decoder modulated with the appearance vector (using Adaptive Instance Normalization) is used to render the final image, in this case through an intermediate MPI that does the view warping and over compositing. Instead of using many parallel finely sliced planes, the GeLaTO (Generative Latent Textured Objects) system uses a small number of oriented planes (“billboards”) with associated neural textures

to model thin transparent objects such as eyeglasses (Martin-Brualla, Pandey *et al.* 2020). At render time, these textures are warped and then decoded and composited using a U-Net to produce a final RGBA sprite.

While all of these previous systems use multiple images to build a 3D neural representation, SynSin (Synthesis from a Single Image) (Wiles, Gkioxari *et al.* 2020) starts with just a single color image and uses a DNN to turn this image into a neural features F and depth D buffer pair, as shown in Figure 14.22d. At render time, the neural features are warped according to their associated depths and the camera view matrix, splatted with soft weights, and composited back-to-front to obtain a neural rendered frame \tilde{F} , which is then decoded into the final color novel view I_G . In Semantic View Synthesis Huang, Tseng *et al.* (2020) start with a semantic label map and use semantic image synthesis (Section 5.5.4) to convert this into a synthetic color image and depth map. These are then used to create a multiplane image from which novel views can be rendered. Holynski, Curless *et al.* (2021) train a deep network to take a static photo, hallucinate a plausible motion field, encode the image as deep features with soft blending weights, advect these features bi-directionally in time, and decode the rendered neural feature frames to produce a looping video clip with synthetic stochastic fluid motions, as discussed in Section 14.5.3.

Voxel representations. Another 3D representation that can be used for neural rendering is a 3D voxel grid. Figure 14.23 shows the modeling and rendering pipelines from two such papers. DeepVoxels (Sitzmann, Thies *et al.* 2019) learn a 3D embedding of neural codes for a given 3D object. At render time, these are projected into 2D view, filtered through an occlusion network (similar to back-to-front alpha compositing), and then decoded into a final image. Neural Volumes (Lombardi, Simon *et al.* 2019) use an encoder-decoder to convert a set of multi-view color images into a 3D $RGB\alpha$ volume and an associated volumetric warp field that can model facial expression variation. At render time, the color volume is warped and then ray marching is used to create a final 2D $RGB\alpha$ foreground image.¹⁷ In more recent work, Weng, Curless, and Kemelmacher-Shlizerman (2020) show how deformable Neural Volumes can be constructed and animated from monocular videos of moving people, such as athletes.

Coordinate-based neural representations. The final representation we discuss in this section are implicit functions implemented using fully connected networks, which are now more

¹⁷Note that we mostly use $RGB\alpha$ in earlier parts of the book to denote three color channels with an opacity. In the remainder of this section, I use $RGB\alpha$ to be consistent with recent papers.

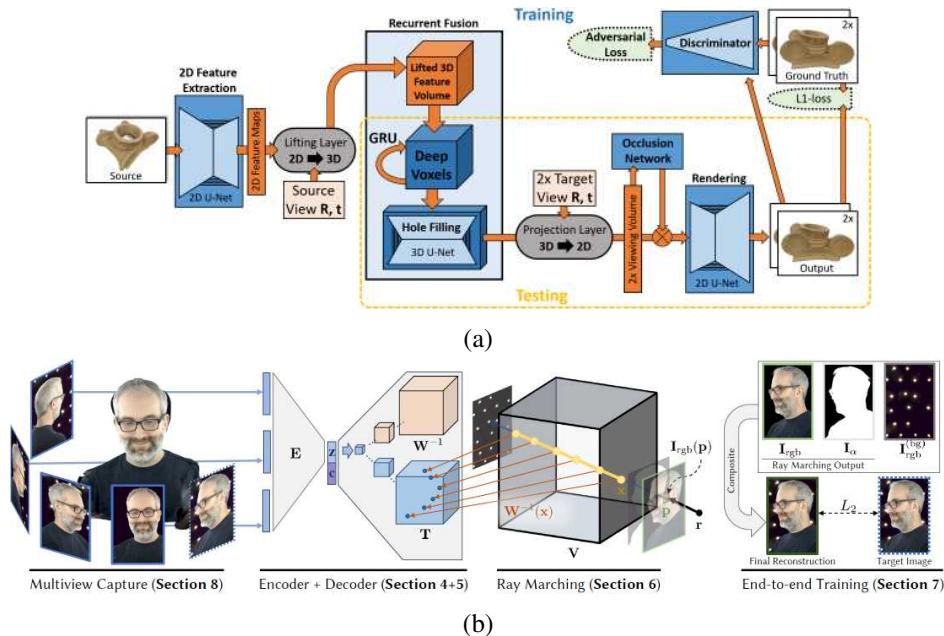


Figure 14.23 Examples of voxel grid neural rendering: (a) DeepVoxels (Sitzmann, Thies et al. 2019) © 2019 IEEE; (b) Neural Volumes (Lombardi, Simon et al. 2019) © 2019 ACM.

commonly known as *multilayer perceptrons* or *MLPs*.¹⁸ We have already seen the use of $[0, 1]$ occupancy and implicit signed distance functions for 3D shape modeling in Section 13.5.1, where we mentioned papers such as Occupancy Networks (Mescheder, Oechsle et al. 2019), IM-NET (Chen and Zhang 2019), DeepSDF (Park, Florence et al. 2019), and Convolutional Occupancy Networks (Peng, Niemeyer et al. 2020).

To render colored images, such representations also need to encode the appearance (e.g., color, texture, or light field) information at either the surface or throughout the volume. Texture Fields (Oechsle, Mescheder et al. 2019) train an MLP conditioned on both 3D shape and latent appearance (e.g., car color) to produce a 3D volumetric color field that can then be used to texture-map a 3D model, as shown in Figure 14.24a. This representation can be extended using differentiable rendering to directly compute depth gradients, as in Differential Volumetric Rendering (DVR) (Niemeyer, Mescheder et al. 2020). Pixel-aligned Implicit function (PIFu) networks (Saito, Huang et al. 2019; Saito, Simon et al. 2020) also use MLPs to com-

¹⁸As Jon Barron and others have pointed out, only signed distance functions actually encode “implicit functions” as level-sets of their volumetric values. The more general class of techniques that includes opacity models is often called *coordinate regression networks* or *coordinate-based MLPs*.

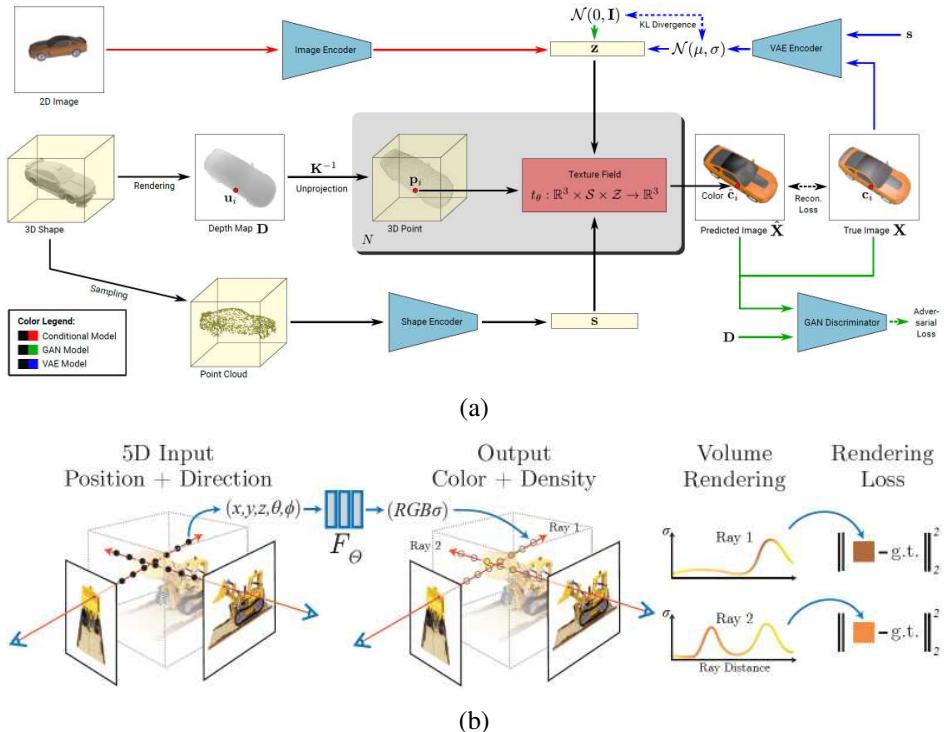


Figure 14.24 Examples of implicit function (MLP) neural rendering: (a) Texture Fields (Oechsle, Mescheder et al. 2019) © 2019 IEEE; (b) Neural Radiance Fields (Mildenhall, Srinivasan et al. 2020) © 2020 Springer.

pute volumetric inside/outside and color fields and can hallucinate full 3D models from just a single color image, as shown in Figure 13.18. Scene representation networks (Sitzmann, Zollhöfer, and Wetzstein 2019) use an MLP to map volumetric (x, y, z) coordinates to high-dimensional neural features, which are used by both a ray marching LSTM (conditioned on the 3D view and output pixel coordinate) and a 1×1 color pixel decoder to generate the final image. The network can interpolate both appearance and shape latent variables.

An interesting hybrid system that replaces a trained per-object MLP with on-the-fly multi-view stereo matching and image-based rendering is the IBRNet system of Wang, Wang et al. (2021). As with other volumetric neural renders, the network evaluates each ray in the novel viewpoint image by marching along the ray and computing a density and neural appearance feature at each sampled location. However, instead of looking up these values from a pre-trained MLP, it samples the neural features from a small number of adjacent input images,

much like in Unstructured Lumigraph (Buehler, Bosse *et al.* 2001; Hedman, Philip *et al.* 2018) and Stable View Synthesis (Riegler and Koltun 2020b), which use a precomputed 3D surface model (which IBRNet does not). The opacity and appearance values along the ray are refined using a transformer architecture, which replaces the more traditional winner-take-all module in a stereo matcher, followed by a classic volumetric compositing of the colors and densities.

To model viewpoint dependent effects such as highlights on plastic objects, i.e., to model a full light field (Section 14.3), Neural Radiance Fields (NeRF) extend the implicit mapping from (x, y, z) spatial positions to also include a viewing direction (θ, ϕ) as inputs, as shown in Figure 14.24b (Mildenhall, Srinivasan *et al.* 2020). Each (x, y, z) query is first turned into a *positional encoding* that consists of sinusoidal waves at octave frequencies before going into a 256-channel MLP. These positional codes are also injected into the fifth layer, and an encoding of the viewing direction is injected at the ninth layer, which is where the opacities are computed (Mildenhall, Srinivasan *et al.* 2020, Figure 7). It turns out that these positional encodings are essential to enabling the MLP to represent fine details, as explored in more depth by Tancik, Srinivasan *et al.* (2020), as well as in the SIREN (Sinusoidal Representation Network) paper by Sitzmann, Martel *et al.* (2020), which uses periodic (sinusoidal) activation functions.

It is also possible to pre-train these neural networks, i.e., use *meta-learning*, on a wider class of objects to speed up the optimization task for new images (Sitzmann, Chan *et al.* 2020; Tancik, Mildenhall *et al.* 2021) and also to use cone tracing together with *integrated positional encoding* to reduce aliasing and handle multi-resolution inputs and output (Bar-ron, Mildenhall *et al.* 2021). The NeRF++ paper by Zhang, Riegler *et al.* (2020) extends the original NeRF representation to handle unbounded 3D scenes by adding an “inside-out” $1/r$ *inverted sphere parameterization*, while Neural Sparse Voxel Fields build an octree with implicit neural functions inside each non-empty cell (Liu, Gu *et al.* 2020).

Instead of modeling opacities, the Implicit Differentiable Renderer (IDR) developed by Yariv, Kasten *et al.* (2020) models a signed distance function, which enables them at rendering time to extract a level-set surface with analytic normals, which are then passed to the neural renderer, which models viewpoint-dependent effects. The system also automatically adjusts input camera positions using differentiable rendering. Neural Lumigraph Rendering uses sinusoidal representation networks to produce more compact representations (Kellnhofer, Jebe *et al.* 2021). They can also export a 3D mesh for much faster view-dependent Lumigraph rendering. Takikawa, Litalien *et al.* (2021) also construct an implicit signed distance field, but instead of using a single MLP, they build a sparse octree structure that stores neural features in cells (much like neural sparse voxel fields) and supports both level of detail and fast sphere

tracing. Neural Implicit Surfaces (NeuS) also use a signed distance representation but use a rendering formula that better handles surface occlusions (Wang, Liu *et al.* 2021).

While NeRF, IDR, and NSVF require a large number of images of a static object taken under controlled (uniform lighting) conditions, NeRF in the Wild (Martin-Brualla, Radwan *et al.* 2021) takes an unstructured set of images from a landmark tourist location and not only models appearance changes such as weather and time of day but also removes transient occluders such as tourists. NeRFs can also be constructed from a single or small number of images by conditioning a class-specific neural radiance field on such inputs as in pixel-NeRF (Yu, Ye *et al.* 2020). Deformable neural radiance fields or “nerfies” (Park, Sinha *et al.* 2020), Neural Scene Flow Fields (Li, Niklaus *et al.* 2021), Dynamic Neural Radiance Fields (Pumarola, Corona *et al.* 2021), Space-time Neural Irradiance Fields (Xian, Huang *et al.* 2021), and HyperNeRF (Park, Sinha *et al.* 2021) all take as input hand-held videos taken around a person or moving through a scene. They model both the viewpoint variation and volumetric non-rigid deformations such as head or body movements and expression changes, either using a learned deformation field, adding time as an extra input variable, or embedding the representation in a higher dimension.

It is also possible to extend NeRFs to model not only the opacities and view-dependent colors of 3D coordinates, but also their interactions with potential illuminants. Neural Reflectance and Visibility Fields (NeRV) do this by also returning for each query 3D coordinate a surface normal and parametric BRDF as well as the environment visibility and expected termination depth for outgoing rays at that point (Srinivasan, Deng *et al.* 2021). Neural Reflection Decomposition (NeRD) models densities and colors using an implicit MLP that also returns an appearance vector, which is decoded into a parametric BRDF (Boss, Braun *et al.* 2020). It then uses the environmental illumination, approximated using spherical Gaussians, along with the density normal and BRDF, to render the final color sample at that voxel. PhySG uses a similar approach, using a signed distance field to represent the shape and a mixture of spherical Gaussian to represent the BRDF (Zhang, Luan *et al.* 2021).

Most of the neural rendering techniques that include view-dependent effects are quite slow to render, since they require sampling a volumetric space along each ray, using expensive MLPs to perform each location/direction lookup. To achieve real-time rendering while modeling view-dependent effects, a number of recent papers use efficient spatial data structures (octrees, sparse grids, or multiplane images) to store opacities and base colors (or potentially small MLPs) and then use factored approximations of the radiance field to model view-dependent effects (Wizadwongsa, Phongthawee *et al.* 2021; Garbin, Kowalski *et al.* 2021; Reiser, Peng *et al.* 2021; Yu, Li *et al.* 2021; Hedman, Srinivasan *et al.* 2021). While the exact details of the representations used in the various stages vary amongst these papers,

they all start with high-fidelity view-dependent models related to the original NeRF paper or its extensions and then “bake” or “distill” these into faster to evaluate spatial data structures and simplified (but still accurate) view-dependent models. The resulting systems produce the same high fidelity renderings as full Neural Radiance Fields while running often 1000x faster than pure MLP-based representations.

As you can tell from the brief discussion in this section, neural rendering is an extremely active research area with new architectures being proposed every few months (Dellaert and Yen-Chen 2021). The best place to find the latest developments, as with other topics in computer vision, is to look on arXiv and in the leading computer vision, graphics, and machine learning conferences.

14.7 Additional reading

Two good surveys of image-based rendering are by Kang, Li *et al.* (2006) and Shum, Chan, and Kang (2007), with earlier surveys available from Kang (1999), McMillan and Gortler (1999), and Debevec (1999). Today, the field often goes under the name of *novel view synthesis* (NVS), with a recent tutorial at CVPR (Gallo, Troccoli *et al.* 2020) providing a good overview of historical and current techniques.

The term *image-based rendering* was introduced by McMillan and Bishop (1995), although the seminal paper in the field is the view interpolation paper by Chen and Williams (1993). Debevec, Taylor, and Malik (1996) describe their Façade system, which not only created a variety of image-based modeling tools but also introduced the widely used technique of *view-dependent texture mapping*. Early work on planar impostors and layers was carried out by Shade, Lischinski *et al.* (1996), Lengyel and Snyder (1997), and Torborg and Kajiya (1996), while newer work based on *sprites with depth* is described by Shade, Gortler *et al.* (1998). Using a large number of parallel planes with RGBA colors and opacities (originally dubbed the “stack of acetates” model by Szeliski and Golland (1999)) was rediscovered by Zhou, Tucker *et al.* (2018) and now goes by the name of multiplane images (MPI). This representation is widely used in recent 3D capture and rendering pipelines (Mildenhall, Srinivasan *et al.* 2019; Choi, Gallo *et al.* 2019; Broxton, Flynn *et al.* 2020; Attal, Ling *et al.* 2020; Lin, Xu *et al.* 2020). To accurately model reflections, the alpha-compositing operator used in MPIS needs to be replaced with an additive model, as in Sinha, Kopf *et al.* (2012) and Kopf, Langguth *et al.* (2013).

The two foundational papers in image-based rendering are *Light field rendering* by Levoy and Hanrahan (1996) and *The Lumigraph* by Gortler, Grzeszczuk *et al.* (1996). Buehler, Bosse *et al.* (2001) generalize the Lumigraph approach to irregularly spaced collections of

images, while Levoy (2006) provides a survey and more gentle introduction to the topic of light field and image-based rendering. Wu, Masia *et al.* (2017) provide a more recent survey of this topic. More recently, neural rendering techniques have been used to improve the blending heuristics used in the Unstructured Lumigraph (Hedman, Philip *et al.* 2018; Riegler and Koltun 2020a).

Surface light fields (Wood, Azuma *et al.* 2000; Park, Newcombe, and Seitz 2018; Yariv, Kasten *et al.* 2020) provide an alternative parameterization for light fields with accurately known surface geometry and support both better compression and the possibility of editing surface properties. Concentric mosaics (Shum and He 1999; Shum, Wang *et al.* 2002) and panoramas with depth (Peleg, Ben-Ezra, and Pritch 2001; Li, Shum *et al.* 2004; Zheng, Kang *et al.* 2007), provide useful parameterizations for light fields captured with panning cameras. Multi-perspective images (Rademacher and Bishop 1998) and manifold projections (Peleg and Herman 1997), although not true light fields, are also closely related to these ideas.

Among the possible extensions of light fields to higher-dimensional structures, environment mattes (Zongker, Werner *et al.* 1999; Chuang, Zongker *et al.* 2000) are the most useful, especially for placing captured objects into new scenes.

Video-based rendering, i.e., the re-use of video to create new animations or virtual experiences, started with the seminal work of Szummer and Picard (1996), Bregler, Covell, and Slaney (1997), and Schödl, Szeliski *et al.* (2000). Important follow-on work to these basic re-targeting approaches includes Schödl and Essa (2002), Kwatra, Schödl *et al.* (2003), Doretto, Chiuso *et al.* (2003), Wang and Zhu (2003), Zhong and Sclaroff (2003), Yuan, Wen *et al.* (2004), Doretto and Soatto (2006), Zhao and Pietikäinen (2007), Chan and Vasconcelos (2009), Joshi, Mehta *et al.* (2012), Liao, Joshi, and Hoppe (2013), Liao, Finch, and Hoppe (2015), Yan, Liu, and Furukawa (2017), He, Liao *et al.* (2017), and Oh, Joo *et al.* (2017). Related techniques have also been used for performance driven video animation (Zollhöfer, Thies *et al.* 2018; Fried, Tewari *et al.* 2019; Chan, Ginosar *et al.* 2019; Egger, Smith *et al.* 2020).

Systems that allow users to change their 3D viewpoint based on multiple synchronized video streams include Moezzi, Katkere *et al.* (1996), Kanade, Rander, and Narayanan (1997), Matusik, Buehler *et al.* (2000), Matusik, Buehler, and McMillan (2001), Carranza, Theobalt *et al.* (2003), Zitnick, Kang *et al.* (2004), Magnor (2005), Vedula, Baker, and Kanade (2005), Joo, Liu *et al.* (2015), Anderson, Gallup *et al.* (2016), Tang, Dou *et al.* (2018), Serrano, Kim *et al.* (2019), Parra Pozo, Toksvig *et al.* (2019), Bansal, Vo *et al.* (2020), Broxton, Flynn *et al.* (2020), and Tewari, Fried *et al.* (2020). 3D (multi-view) video coding and compression is also an active area of research (Smolic and Kauff 2005; Gotchev and Rosenhahn 2009), and is used in 3D Blu-Ray discs and multi-view video coding (MVC) extensions to the High

Efficiency Video Coding (HEVC) standard ([Tech, Chen et al. 2015](#)).

The whole field of neural rendering is quite recent, with initial publications focusing on 2D image synthesis ([Zhu, Krähenbühl et al. 2016](#); [Isola, Zhu et al. 2017](#)) and only more recently being applied to 3D novel view synthesis ([Hedman, Philip et al. 2018](#); [Martin-Brualla, Pandey et al. 2018](#)). [Tewari, Fried et al. \(2020\)](#) provide an excellent survey of this area, with 230 references and 46 highlighted papers. Additional overviews include the related CVPR tutorial on neural rendering ([Tewari, Zollhöfer et al. 2020](#)), several of the lectures in the TUM AI Guest Lecture Series, the X-Fields paper by [Bemana, Myszkowski et al. \(2020, Table 1\)](#), and a recent bibliography by [Dellaert and Yen-Chen \(2021\)](#).

14.8 Exercises

Ex 14.1: Depth image rendering. Develop a “view extrapolation” algorithm to re-render a previously computed stereo depth map coupled with its corresponding reference color image.

1. Use a 3D graphics mesh rendering system such as OpenGL with two triangles per pixel quad and perspective (projective) texture mapping ([Debevec, Yu, and Borshukov 1998](#)).
2. Alternatively, use the one- or two-pass forward warper you constructed in Exercise 3.24, extended using ([2.68–2.70](#)) to convert from disparities or depths into displacements.
3. (Optional) Kinks in straight lines introduced during view interpolation or extrapolation are visually noticeable, which is one reason why image morphing systems let you specify line correspondences ([Beier and Neely 1992](#)). Modify your depth estimation algorithm to match and estimate the geometry of straight lines and incorporate it into your image-based rendering algorithm.

Ex 14.2: View interpolation. Extend the system you created in the previous exercise to render two reference views and then blend the images using a combination of z-buffering, hole filling, and blending (morphing) to create the final image (Section 14.1).

1. (Optional) If the two source images have very different exposures, the hole-filled regions and the blended regions will have different exposures. Can you extend your algorithm to mitigate this?
2. (Optional) Extend your algorithm to perform three-way (trilinear) interpolation between neighboring views. You can triangulate the reference camera poses and use barycentric coordinates for the virtual camera to determine the blending weights.

Ex 14.3: View morphing. Modify your view interpolation algorithm to perform morphs between views of a non-rigid object, such as a person changing expressions.

1. Instead of using a pure stereo algorithm, use a general flow algorithm to compute displacements, but separate them into a rigid displacement due to camera motion and a non-rigid deformation.
2. At render time, use the rigid geometry to determine the new pixel location but then add a fraction of the non-rigid displacement as well.
3. (Optional) Take a single image, such as the Mona Lisa or a friend’s picture, and create an animated 3D view morph (Seitz and Dyer 1996).
 - (a) Find the vertical axis of symmetry in the image and reflect your reference image to provide a virtual pair (assuming the person’s hairstyle is somewhat symmetric).
 - (b) Use structure from motion to determine the relative camera pose of the pair.
 - (c) Use dense stereo matching to estimate the 3D shape.
 - (d) Use view morphing to create a 3D animation.

Ex 14.4: View dependent texture mapping. Use a 3D model you created along with the original images to implement a view-dependent texture mapping system.

1. Use one of the 3D reconstruction techniques you developed in Exercises 11.10, 12.9, 12.10, or 13.8 to build a triangulated 3D image-based model from multiple photographs.
2. Extract textures for each model face from your photographs, either by performing the appropriate resampling or by figuring out how to use the texture mapping software to directly access the source images.
3. For each new camera view, select the best source image for each visible model face.
4. Extend this to blend between the top two or three textures. This is trickier, because it involves the use of texture blending or pixel shading (Debevec, Taylor, and Malik 1996; Debevec, Yu, and Borshukov 1998; Pighin, Hecker *et al.* 1998).

Ex 14.5: Layered depth images. Extend your view interpolation algorithm (Exercise 14.2) to store more than one depth or color value per pixel (Shade, Gortler *et al.* 1998), i.e., a layered depth image (LDI). Modify your rendering algorithm accordingly. For your data, you can use synthetic ray tracing, a layered reconstructed model, or a volumetric reconstruction.

Ex 14.6: Rendering from sprites or layers. Extend your view interpolation algorithm to handle multiple planes or sprites (Section 14.2.1) (Shade, Gortler *et al.* 1998).

1. Extract your layers using the technique you developed in Exercise 9.7.
2. Alternatively, use an interactive painting and 3D placement system to extract your layers (Kang 1998; Oh, Chen *et al.* 2001; Shum, Sun *et al.* 2004).
3. Determine a back-to-front order based on expected visibility or add a z-buffer to your rendering algorithm to handle occlusions.
4. Render and composite all of the resulting layers, with optional alpha matting to handle the edges of layers and sprites.
5. Try one of the newer multiplane image (MPI) techniques (Zhou, Tucker *et al.* 2018).

Ex 14.7: Light field transformations. Derive the equations relating regular images to 4D light field coordinates.

1. Determine the mapping between the far plane (u, v) coordinates and a virtual camera's (x, y) coordinates.
 - (a) Start by parameterizing a 3D point on the uv plane in terms of its (u, v) coordinates.
 - (b) Project the resulting 3D point to the camera pixels $(x, y, 1)$ using the usual 3×4 camera matrix \mathbf{P} (2.63).
 - (c) Derive the 2D homography relating (u, v) and (x, y) coordinates.
2. Write down a similar transformation for (s, t) to (x, y) coordinates.
3. Prove that if the virtual camera is actually on the (s, t) plane, the (s, t) value depends only on the camera's image center and is independent of (x, y) .
4. Prove that an image taken by a regular orthographic or perspective camera, i.e., one that has a linear projective relationship between 3D points and (x, y) pixels (2.63), samples the (s, t, u, v) light field along a two-dimensional hyperplane.

Ex 14.8: Light field and Lumigraph rendering. Implement a light field or Lumigraph rendering system:

1. Download one of the light field datasets from <http://lightfield.stanford.edu> or <https://lightfield-analysis.uni-konstanz.de>.

2. Write an algorithm to synthesize a new view from this light field, using quadri-linear interpolation of (s, t, u, v) ray samples.
3. Try varying the focal plane corresponding to your desired view (Isaksen, McMillan, and Gortler 2000) and see if the resulting image looks sharper.
4. Determine a 3D proxy for the objects in your scene. You can do this by running multi-view stereo over one of your light fields to obtain a depth map per image.
5. Implement the Lumigraph rendering algorithm, which modifies the sampling of rays according to the 3D location of each surface element.
6. Collect a set of images yourself and determine their pose using structure from motion.
7. Implement the unstructured Lumigraph rendering algorithm from Buehler, Bosse *et al.* (2001).

Ex 14.9: Surface light fields. Construct a surface light field (Wood, Azuma *et al.* 2000) and see how well you can compress it.

1. Acquire an interesting light field of a specular scene or object, or download one from <http://lightfield.stanford.edu> or <https://lightfield-analysis.uni-konstanz.de>.
2. Build a 3D model of the object using a multi-view stereo algorithm that is robust to outliers due to specularities.
3. Estimate the Lumisphere for each surface point on the object.
4. Estimate its diffuse components. Is the median the best way to do this? Why not use the minimum color value? What happens if there is Lambertian shading on the diffuse component?
5. Model and compress the remaining portion of the Lumisphere using one of the techniques suggested by Wood, Azuma *et al.* (2000) or invent one of your own.
6. Study how well your compression algorithm works and what artifacts it produces.
7. (Optional) Develop a system to edit and manipulate your surface light field.

Ex 14.10: Handheld concentric mosaics. Develop a system to navigate a handheld concentric mosaic.

1. Stand in the middle of a room with a camcorder held at arm's length in front of you and spin in a circle.

2. Use a structure from motion system to determine the camera pose and sparse 3D structure for each input frame.
3. (Optional) Re-bin your image pixels into a more regular concentric mosaic structure.
4. At view time, determine from the new camera's view (which should be near the plane of your original capture) which source pixels to display. You can simplify your computations to determine a source column (and scaling) for each output column.
5. (Optional) Use your sparse 3D structure, interpolated to a dense depth map, to improve your rendering ([Zheng, Kang *et al.* 2007](#)).

Ex 14.11: Video textures. Capture some videos of natural phenomena, such as a water fountain, fire, or smiling face, and loop the video seamlessly into an infinite length video ([Schödl, Szeliski *et al.* 2000](#)).

1. Compare all the frames in the original clip using an L_2 (sum of square difference) metric. (This assumes the videos were shot on a tripod or have already been stabilized.)
2. Filter the comparison table temporally to accentuate temporal sub-sequences that match well together.
3. Convert your similarity table into a jump probability table through some exponential distribution. Be sure to modify transitions near the end so you do not get “stuck” in the last frame.
4. Starting with the first frame, use your transition table to decide whether to jump forward, backward, or continue to the next frame.
5. (Optional) Add any of the other extensions to the original video textures idea, such as multiple moving regions, interactive control, or graph cut spatio-temporal texture seaming.

Ex 14.12: Neural rendering. Most of the recent neural rendering papers come with open source code as well as carefully acquired datasets.

Try downloading more than one of these and run different algorithms on different datasets. Compare the quality of the renderings you obtain and list the visual artifacts you detect and how you might improve them.

Try capturing your own dataset, if this is feasible, and describe additional breaking points of the current algorithms.

Chapter 15

Conclusion

In this book, we have covered a broad range of computer vision topics. We started with a review of basic geometry and optics, as well as mathematical tools such as image and signal processing, continuous and discrete optimization, statistical modeling, and machine learning. We then used these to develop computer vision algorithms such as image enhancement and segmentation, object detection and classification, motion estimation, and 3D shape reconstruction. These components, in turn, enabled us to build more complex applications, such as large-scale image retrieval, converting images to descriptions, stitching multiple images into wider and higher dynamic range composites, tracking people and objects, navigating in new unseen environments, and augmenting video with embedded 3D overlays.

In the decade since the publication of the first edition of this book, the computer vision field has exploded, both in the maturity and reliability of vision algorithms, as well as the number of practitioners and commercial applications. The most notable advance has been in deep learning, which now enables visual recognition at a level of performance that has eclipsed what we could do ten years ago. Deep learning has also found widespread application in basic vision algorithms such as image enhancement, motion estimation, and 3D shape recovery. Other advances, such as reliable real-time tracking and reconstruction have enabled applications such as autonomous navigation and phone-based augmented reality. And advances in sophisticated image processing have produced computational photography algorithms that run in every mobile phone producing images that surpass the quality available with much more expensive traditional photographic equipment.

You may ask: Why is our field so broad and aren't there any unifying principles that can be used to simplify our study? Part of the answer lies in the expansive definition of computer vision, which is the capture, analysis, and interpretation of our 3D environment using images

and video, as well as the incredible complexity inherent in the formation of visual imagery. In some ways, our field is as complex as the study of automotive engineering, which requires an understanding of internal combustion, mechanics, aerodynamics, ergonomics, electrical circuitry, and control systems, among other topics. Computer vision similarly draws on a wide variety of sub-disciplines, which makes it challenging to cover in a one-semester course, or even to achieve mastery during a course of graduate studies. Conversely, the incredible breadth and technical complexity of computer vision is what draws many people to this research field.

Because of this richness and the difficulty in making and measuring progress, I attempt to instill in my students, and hopefully in the readers of this book, a discipline founded on principles from engineering, science, statistics, and machine learning.

The engineering approach to problem solving is to first carefully define the overall problem being tackled and to question the basic assumptions and goals inherent in this process. Once this has been done, a number of alternative solutions or approaches are implemented and carefully tested, paying attention to issues such as reliability and computational cost. Finally, one or more solutions are deployed and evaluated in real-world settings. For this reason, this book contains different alternatives for solving vision problems, many of which are sketched out in the exercises for students to implement and test on their own.

The scientific approach builds upon a basic understanding of physical principles. In the case of computer vision, this includes the physics of natural and artificial structures, image formation, including lighting and atmospheric effects, optics, and noisy sensors. The task is to then invert this formation using stable and efficient algorithms to obtain reliable descriptions of the scene and other quantities of interest. The scientific approach also encourages us to formulate and test hypotheses, which is similar to the extensive testing and evaluation inherent in engineering disciplines.

Because so much about the image formation process is inherently uncertain and ambiguous, a statistical approach, which models both the uncertainty and prior distributions in the world, as well as the degradations in the image formation process, is often essential. Bayesian inference techniques can then be used to combine prior and measurement models to estimate the unknowns and to model their uncertainty. Efficient learning and inference algorithms, such as dynamic programming, graph cuts, and belief propagation, often play a crucial role in this process.

Finally, machine learning techniques, driven by large amounts of training data—both labeled (supervised) and unlabeled (unsupervised)—enable the development of models that can discover hard to describe regularities and patterns in the world, which can make inference more reliable. However, despite the incredible advances enabled by learning techniques, we

must still remain cautious about the inherent limitations of learning-based approaches, and not just slough off problems due to “insufficient or biased training data” as someone else’s problem.

Along these lines, I was inspired by a segment from Shree Nayar’s *First Principles of Computer Vision* online lecture series:¹

Since deep learning is very popular today, you may be wondering if it is worth knowing the first principles of vision, or for that matter, the first principles of any field. Given a task, why not just train a neural network with tons of data to solve the task?

Indeed, there are applications where such an approach may suffice. But there are several reasons to embrace the basics. First, it would be laborious and unnecessary to train a network to learn a phenomenon that can be precisely and concisely described using first principles. Second, when a network does not perform well enough, first principles are your only hope for understanding why. Third, collecting data to train a network can be tedious, and sometimes even impractical. In such cases, models based on first principles can be used to synthesize the data, instead of collecting it. And finally, the most compelling reason to learn the first principles of any field is curiosity. What makes humans unique is that innate desire to know why things work the way they do.

Given the breadth of material we have covered in this book, what new developments are we likely to see in the future? It seems fairly obvious from the tremendous advances in the last decade that machine learning, including the ability to fine-tune architectures and algorithm to optimize continuous criteria and metrics, will continue to evolve and produce significant improvements. The current dominance of feedforward convolutional architectures, mostly using weighted linear summation and simple non-linearities, is likely to evolve to include more complex architectures with attention and top-down feedback, as we are already starting to see. Sophisticated application-specific imaging sensors will likely start being used more often, displacing and enhancing the use of visible light imaging sensors originally developed for photography. Integration with additional sensors, such as IMUs and potentially active sensing (where power permits) will make classic problems such as real-time localization and 3D reconstruction much more reliable and ubiquitous.

The most challenging applications of computer vision will likely remain in the realm of artificial general intelligence (AGI), which aims to create systems that exhibit the same range of understanding and behaviors as people. Since progress here depends on concurrent

¹<https://fpcv.cs.columbia.edu>, Introduction:Overview video

progress in many other aspects of artificial intelligence, it will be interesting to see how these different AI modalities and capabilities leverage each other for improved performance.

Whatever the outcome of these research endeavors, computer vision is already having a tremendous impact in many areas, including digital photography, visual effects, medical imaging, safety and surveillance, image search, product recommendations, and aids for the visually impaired. The breadth of the problems and techniques inherent in this field, combined with the richness of the mathematics and the utility of the resulting algorithms, will ensure that this remains an exciting area of study for years to come.