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 *Application*: 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 *Application*: 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

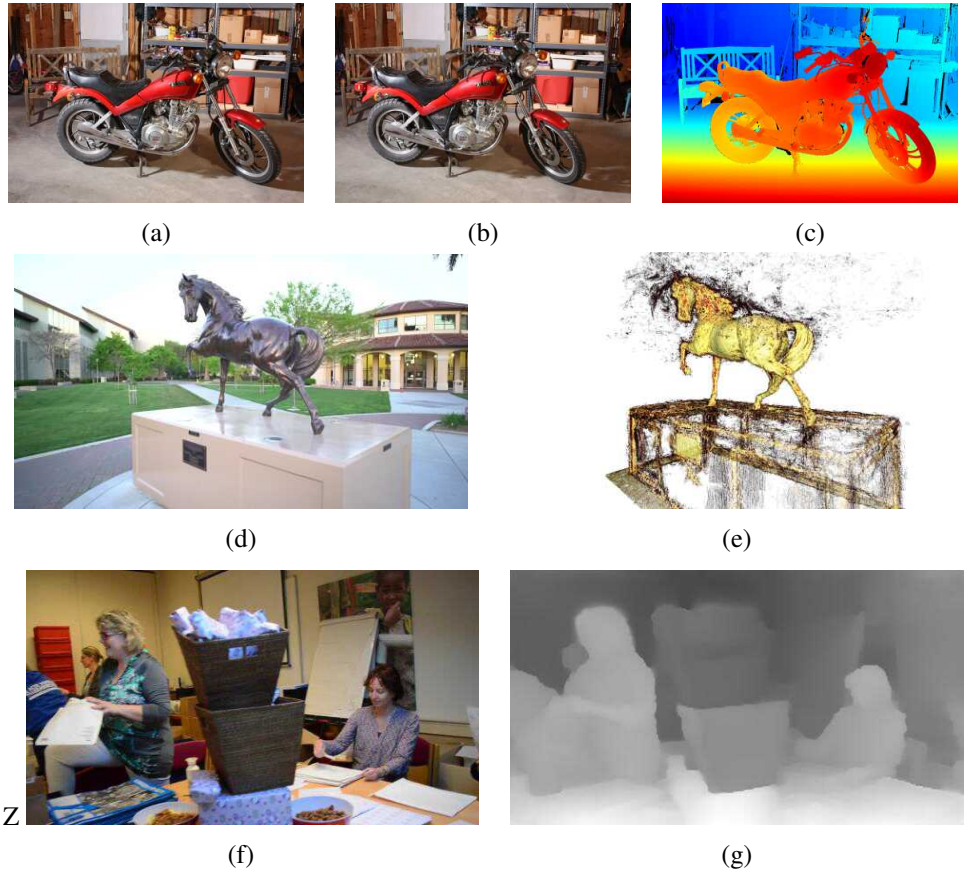


Figure 12.1 *Depth estimation algorithms can convert a pair of color images (a–b) into a depth map (c) (Scharstein, Hirschmüller* et al. *2014) © 2014 Springer, a sequence of images (d) into a 3D model (e) (Knapitsch, Park* et al. *2017) © 2017 ACM, or a single image (f) into a depth map (g) (Li, Dekel* 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,

[1] The word *stereo* comes from the Greek for *solid*; stereo vision is how we perceive solid shape (Koenderink 1990).

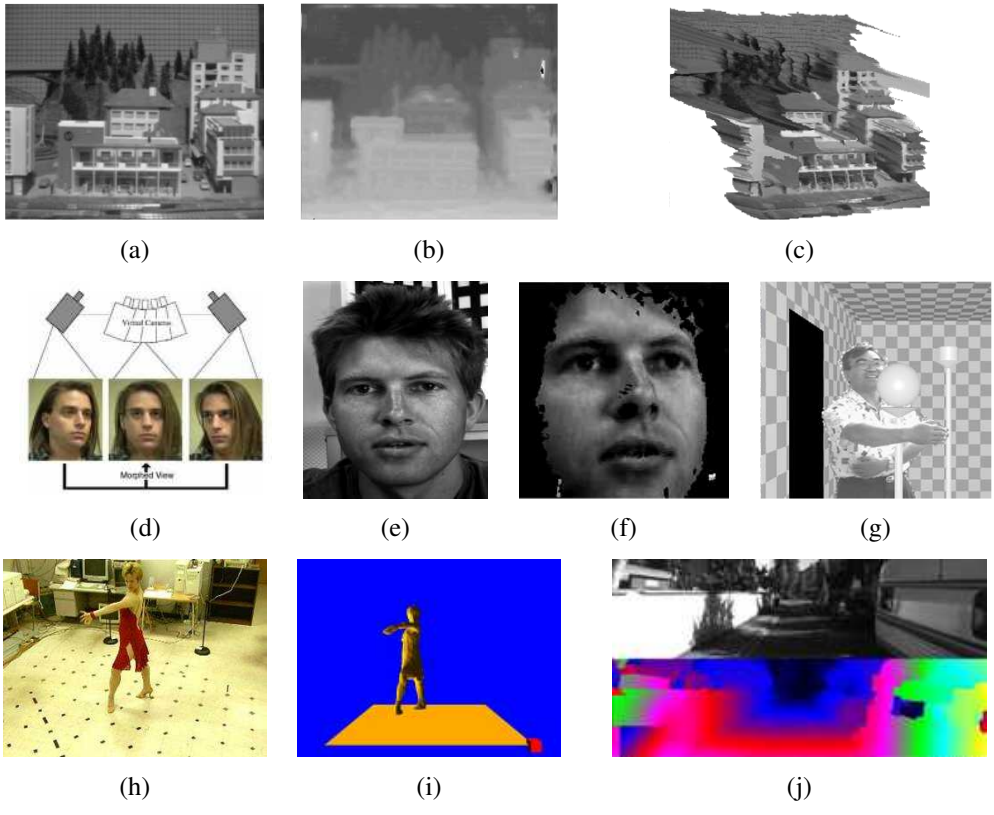


Figure 12.2 *Applications of stereo vision: (a) input image, (b) computed depth map, and (c) new view generation from multi-view stereo (Matthies, Kanade, and Szeliski 1989) © 1989 Springer; (d) view morphing between two images (Seitz and Dyer 1996) © 1996 ACM; (e–f) 3D face modeling (images courtesy 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 *et al.* 2014)
Middlebury multi-view https://vision.middlebury.edu/mview	6 3D objects scanned from 300+ views (Seitz, Curless *et al.* 2006)
EPFL (no longer active)	6 outdoor multi-view sets of images (Strecha, von Hansen *et al.* 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 *et al.* 2014)
Freiburg Scene Flow https://lmb.informatik.uni-freiburg.de/resources/datasets	39k synthetic stereo pairs (Mayer, Ilg *et al.* 2018)
ETH3D https://www.eth3d.net	13 training + 12 test high-res scenes (Schöps, Schönberger *et al.* 2017)
Tanks and Temples https://www.tanksandtemples.org	7 training + 14 test 4K video scenes (Knapitsch, Park *et al.* 2017)
BlendedMVS https://github.com/YoYo000/BlendedMVS	17k MVS images covering 113 scenes (Yao, Luo *et al.* 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 $\mathbf{x}_0$ 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 $\mathbf{p}_\infty$ and at the other end by the projection of the original camera center $\mathbf{c}_0$ into the second camera, which is known as the *epipole* $\mathbf{e}_1$. 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 $\mathbf{e}_0$. 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 $\mathbf{c}_0$ and $\mathbf{c}_1$ as well as the point of interest $\mathbf{p}$ (Faugeras and Luong 2001; Hartley and Zisserman 2004).

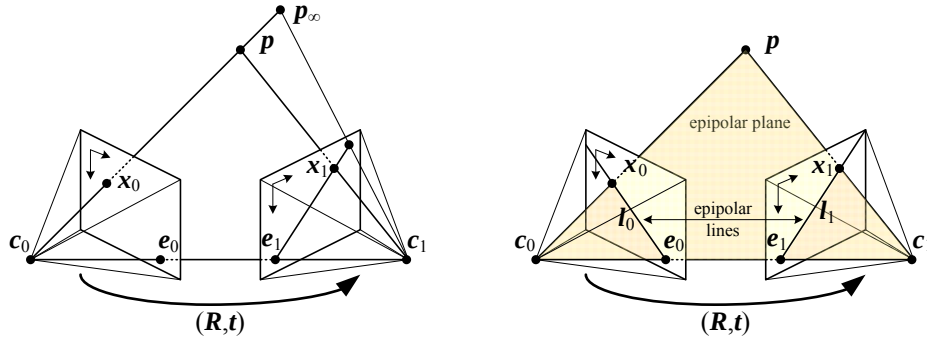


Figure 12.3 *Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.*

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 $\mathbf{c}_0$ and $\mathbf{c}_1$. As there is a degree of freedom in the *tilt*, the smallest rotations that achieve this should be used. Next, to

[2]This 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 (Goesele, Snavely *et al.* 2007) may be preferable.

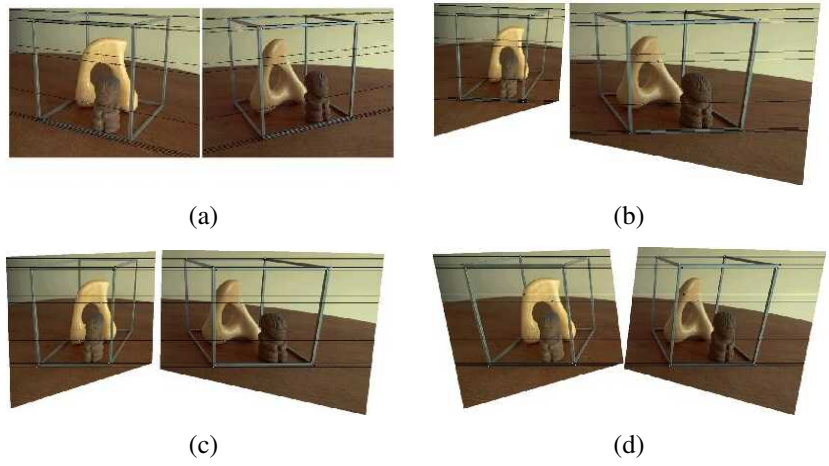


Figure 12.4 *The multi-stage stereo rectification algorithm of Loop and Zhang (1999) © 1999 IEEE. (a) Original image pair overlaid with several epipolar lines; (b) images transformed so that epipolar lines are parallel; (c) images rectified so that epipolar lines are horizontal and in vertical correspondence; (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 Fusiello, 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,

$$d = f\frac{B}{Z}, \tag{12.1}$$

where f is the focal length (measured in pixels), B is the baseline, and

$$x' = x + d(x, y), \quad y' = y \tag{12.2}$$

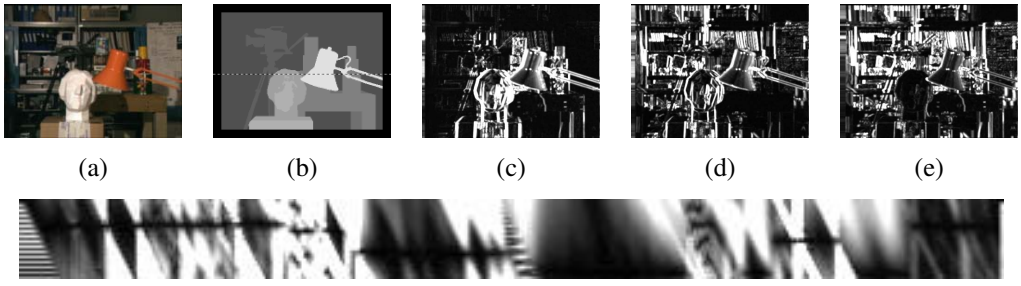


Figure 12.5 *Slices through a typical disparity space image (DSI) (Scharstein and Szeliski 2002) © 2002 Springer: (a) original color image; (b) ground truth disparities; (c–e) three (x, y) slices for $d = 10, 16, 21$; (f) an (x, d) slice for $y = 151$ (the dashed line in (b)). Various dark (matching) 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).*

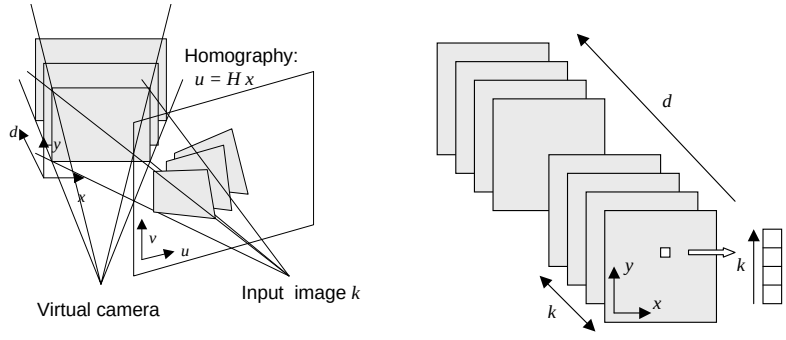
describes the relationship between corresponding pixel coordinates in the left and right images (Bolles, Baker, and Marimont 1987; Okutomi and Kanade 1993; Scharstein and Szeliski 2002).[3] The task of extracting depth from a set of images then becomes one of estimating the *disparity map* $d(x, y)$.

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).

[3] The 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



(a) (b)

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),

$$\tilde{\mathbf{x}}_k \sim \tilde{\mathbf{P}}_k \tilde{\mathbf{P}}^{-1} \mathbf{x}_s = \tilde{\mathbf{H}}_k \tilde{\mathbf{x}} + \mathbf{t}_k d = (\tilde{\mathbf{H}}_k + \mathbf{t}_k [0\ 0\ d]) \tilde{\mathbf{x}}, \tag{12.3}$$

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

$$C(x, y, d) = \sum_k \rho(\tilde{I}(x, y, d, k) - I_r(x, y)). \tag{12.4}$$

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

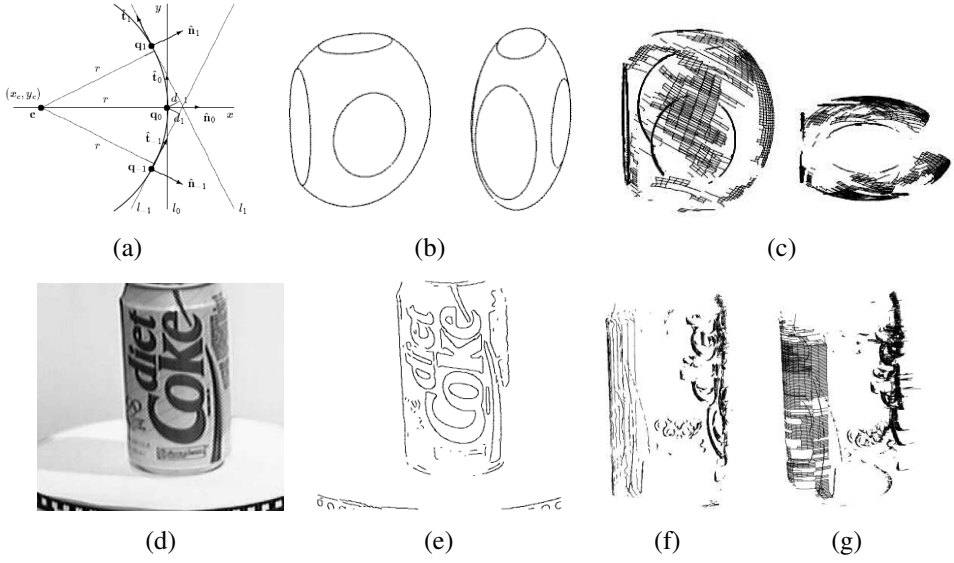


Figure 12.7 *Surface reconstruction from occluding contours (Szeliski and Weiss 1998) © 2002 Springer: (a) circular arc fitting in the epipolar plane; (b) synthetic example of an ellipsoid with a truncated side and elliptic surface markings; (c) partially reconstructed surface mesh seen from an oblique 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

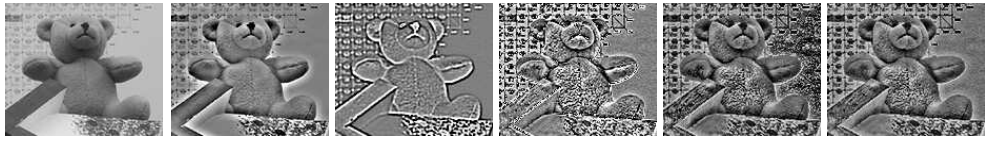
The first component of any dense stereo matching algorithm is a similarity measure that compares pixel values in order to determine how likely they are to be in correspondence. In this section, we briefly review the similarity measures introduced in Section 9.1 and mention a few others that have been 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



(a) Intensity image (b) Mean filter (c) LOG filter (d) BilSub filter (e) Rank filter (f) SoftRank filter

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,

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d), \tag{12.5}$$

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

[4] For two surveys and comparisons of such techniques, please see the work of Gong, Yang *et al.* (2007) and Tombari, Mattoccia *et al.* (2008).

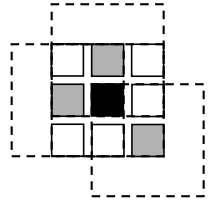


Figure 12.9 *Shiftable window (Scharstein and Szeliski 2002) © 2002 Springer. The effect of trying all 3 × 3 shifted windows around the black pixel is the same as taking the minimum matching score across all* centered *(non-shifted) windows in the same neighborhood. (For clarity, only three of the neighboring shifted windows are shown here.)*

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

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-

[5] Extensive 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

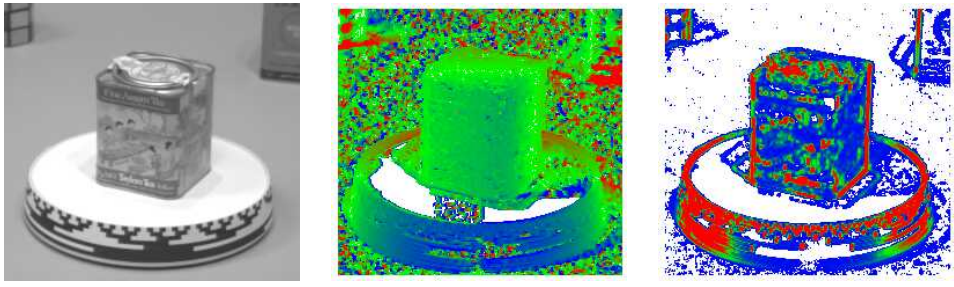
$$Var(d) = \frac{\sigma_I^2}{a}, \tag{12.6}$$

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



(a) (b) (c)

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,

$$E(d) = E_{\mathrm{D}}(d) + \lambda E_{\mathrm{S}}(d). \tag{12.7}$$

The data term, $E_{\mathrm{D}}(d)$, 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

$$E_{\mathrm{D}}(d) = \sum_{(x,y)} C(x, y, d(x, y)), \tag{12.8}$$

where C is the (initial or aggregated) matching cost DSI.

The smoothness term $E_{\mathrm{S}}(d)$ 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,

$$E_{\mathrm{S}}(d) = \sum_{(x,y)} \rho(d(x, y) - d(x + 1, y)) + \rho(d(x, y) - d(x, y + 1)), \tag{12.9}$$

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.,

$$\rho_D(d(x,y) - d(x+1,y)) \cdot \rho_I(\|I(x,y) - I(x+1,y)\|), \tag{12.10}$$

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 KITTI 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

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

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 PatchMatch 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).

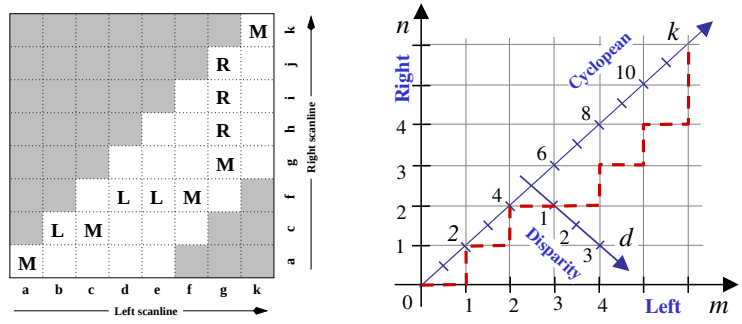


Figure 12.13 *Stereo matching using dynamic programming, as illustrated by (a) Scharstein and Szeliski (2002) © 2002 Springer and (b) Kolmogorov, Criminisi* et al. *(2006) © 2006 IEEE. For each pair of corresponding scanlines, a minimizing path through the matrix of all pairwise matching costs (DSI) 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.

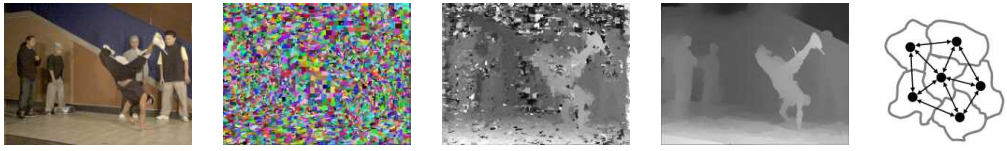
Problems with dynamic programming stereo include the selection of the right cost for occluded pixels and the difficulty of enforcing inter-scanline consistency, although several methods propose ways of addressing the latter (Ohta and Kanade 1985; Belhumeur 1996; Cox, Hingorani *et al.* 1996; Bobick and 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.

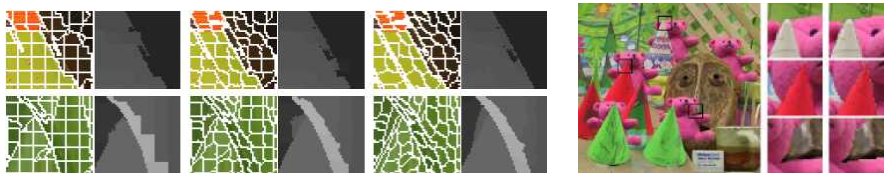
12.5.2 Segmentation-based techniques

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.



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

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.*



(a) (b)

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.

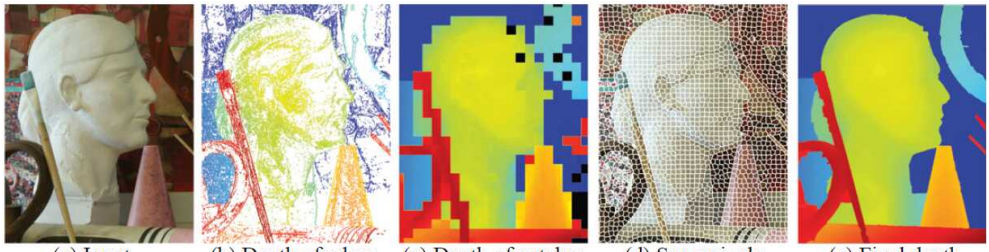


Figure 12.16 *Multiframe matching using edges, planes, and superpixels (Xue, Owens* et al. *2019) © 2019 Elsevier.*

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). Žbontar 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 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 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.

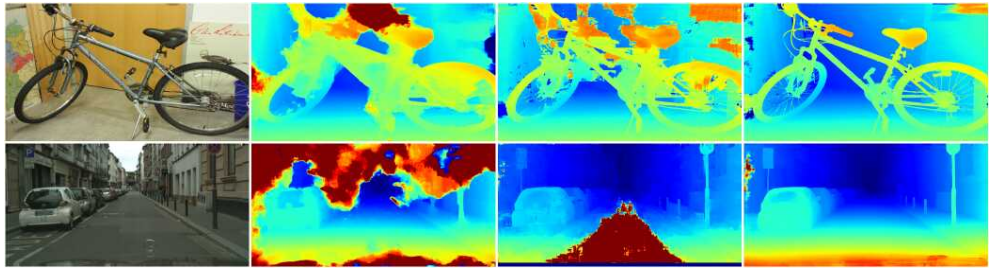


Figure 12.18 *Disparity maps computed by three different DNN stereo matchers trained on synthetic data and applied to real-world image pairs (Zhang, Qi* et al. *2020)* © *2020 Springer.*

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

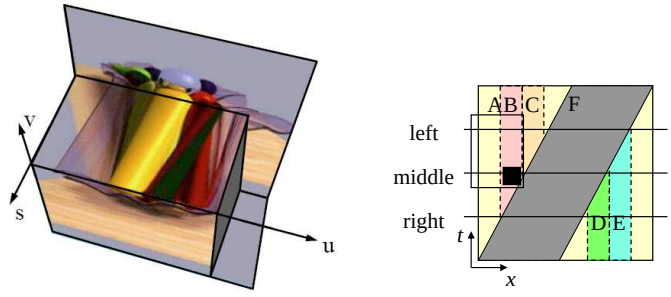
geometric inconsistencies (Chabra, Straub *et al.* 2019). While resource constraints often mean that 3D DNN-based stereo methods operate at fairly low resolutions, the Hierarchical Stereo Matching (HSM) network (Yang, Manela *et al.* 2019) uses a pyramid approach that selectively restricts the search 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, Rhemann, 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 HD^3 (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



(a) (b)

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),

$$C(x, y, d) = \sum_k \rho(\tilde{I}(x, y, d, k) - I_r(x, y)). \tag{12.11}$$

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

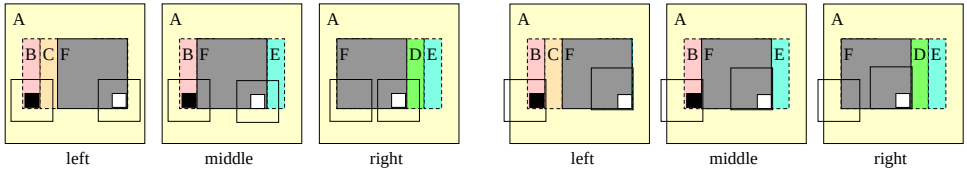


Figure 12.20 *Spatio-temporally shiftable windows (Kang, Szeliski, and Chai 2001) © 2001 IEEE: A simple three-image sequence (the middle image is the reference image), which has a moving frontal gray square (marked F) and a stationary background. Regions B, C, D, and E are partially occluded. (a) A regular 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-

[7] The four-dimensional generalization of the EPI is the *light field*, which we study in Section 14.3.

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

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.*

ure12.19b 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

PMVS SurfaceNet MVSNet Ground truth

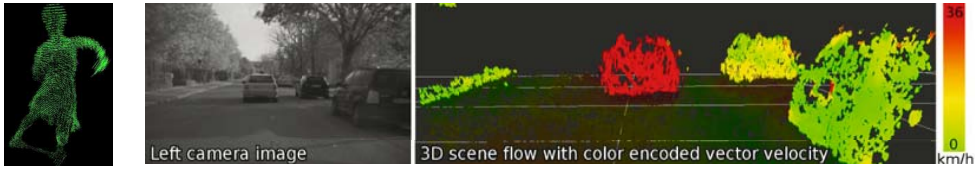
Figure 12.22 *Depth maps computed using three different multi-view stereo algorithms shown as colored point clouds (Yao, Luo* et al. *2018) © 2018 Springer. The red boxes indicate problem areas where MVSNet does better.*

(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), D^2HC-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 (Table12.1) 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),



(a) (b)

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

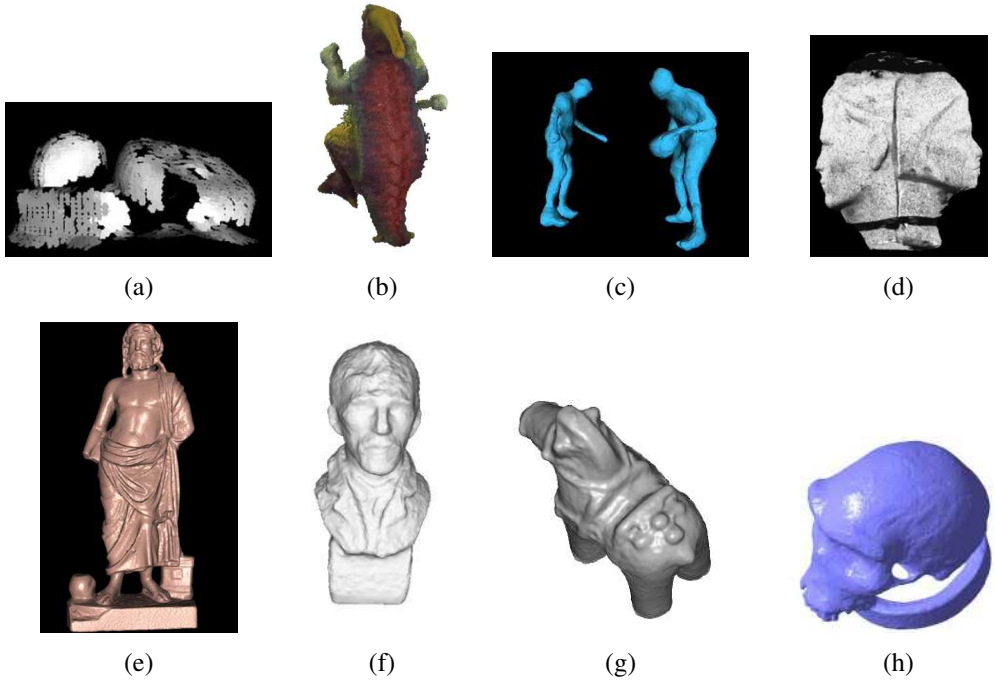


Figure 12.24 *Multi-view stereo algorithms: (a) surface-based stereo (Fua and Leclerc 1995) © 1995 Springer; (b) voxel coloring (Seitz and Dyer 1999) © 1999 Springer; (c) depth map merging (Narayanan, Rander, and Kanade 1998) © 1998 IEEE; (d) level set evolution (Faugeras and Keriven 1998) © 1998 IEEE; (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*,

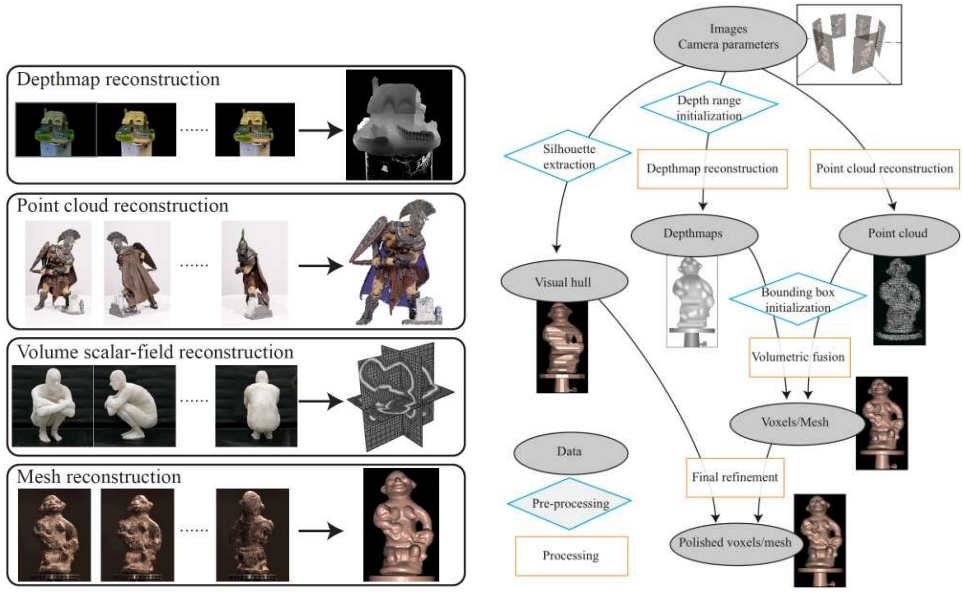


Figure 12.25 *Multi-view stereo (a) scene representations and (b) processing pipelines, from Furukawa and Hernández (2015) © 2015 now publishers.*

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

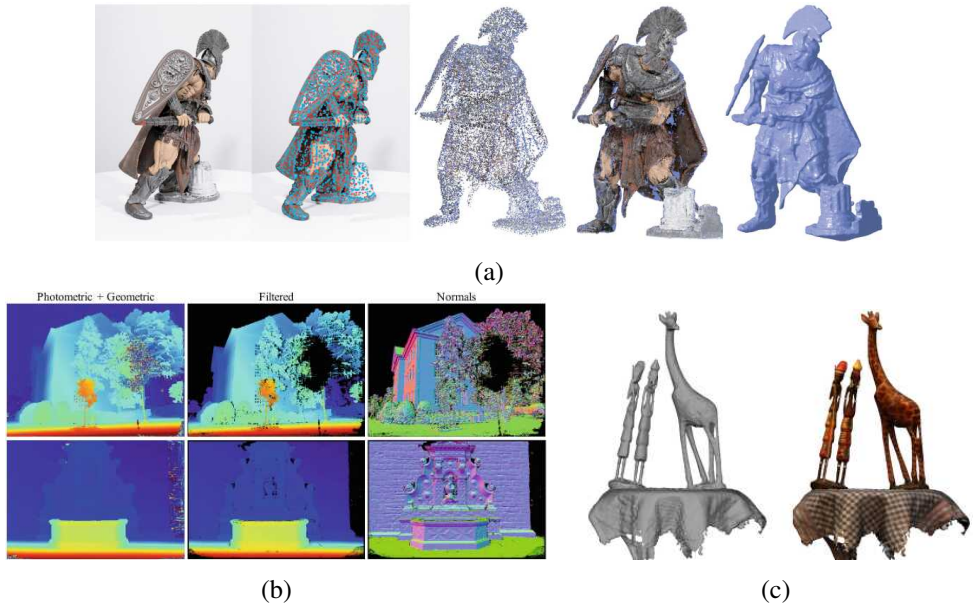


Figure 12.26 *Point cloud reconstruction: (a) the PMVS pipeline, showing a sample input image, detected features, initial reconstructed patches, patches after expansion and filtering, and the final mesh model (Furukawa and Ponce 2010) © 2010 IEEE; (b) depth maps and surface normals from two stages of 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.

[8] For outdoor scenes that go to infinity, an inverted gridding of space may be preferable (Slabaugh, Culbertson *et al.* 2004; Zhang, Riegler *et al.* 2020).

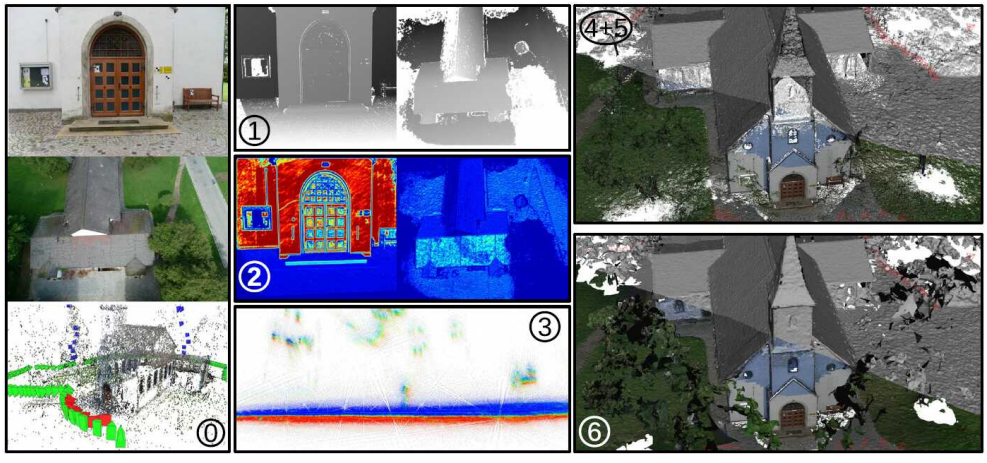


Figure 12.27 *3D reconstruction pipeline from Kuhn, Hirschmüller* et al. *(2017) © 2017 Springer: (0) structure from motion; (1) stereo matching using semi-global matching; (2) depth quality estimation; (3) probabilistic space occupancy; (4+5) probabilistic point optimization and outlier filtering; (6) 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 textureless 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,

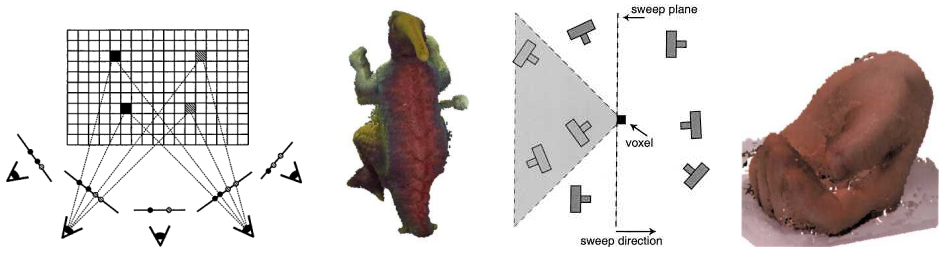


Figure 12.28 *Voxel coloring (Seitz and Dyer 1999) © 1999 Springer and space carving (Kutulakos and Seitz 2000) © 2000 Springer. (a–b): voxel coloring sweeps a plane through the scene in a front-to-back manner with respect to the cameras. (c–d): space carving uses multiple sweep directions to deal with 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

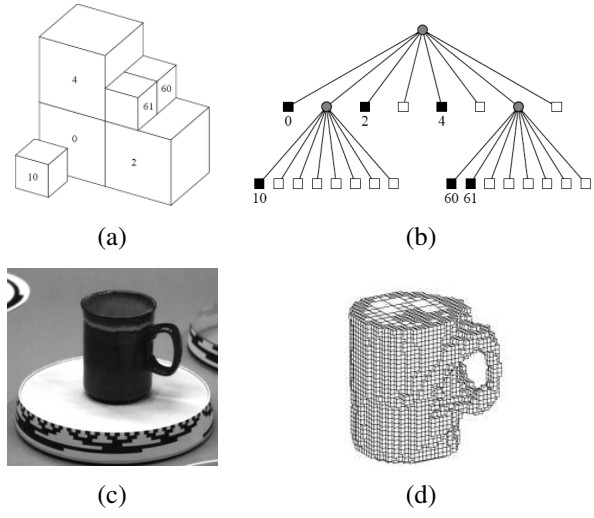


Figure 12.29 *Volumetric octree reconstruction from binary silhouettes (Szeliski 1993) © 1993 Elsevier: (a) octree representation and its corresponding (b) tree structure; (c) input image of an object on a turntable; (d) computed 3D volumetric octree model.*

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

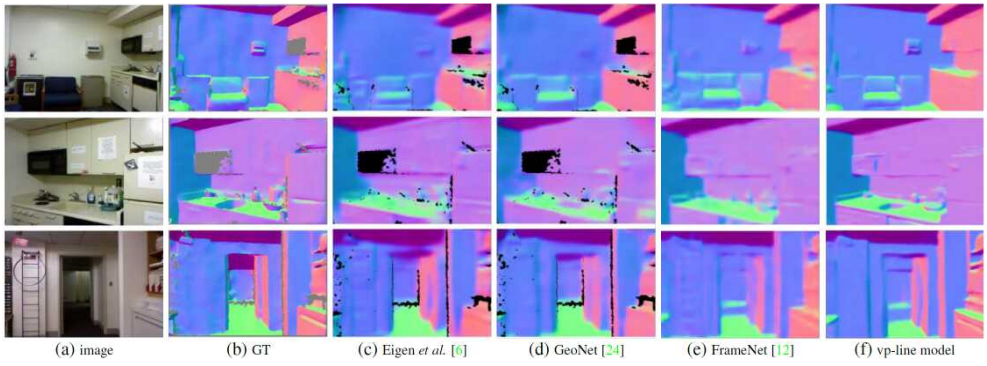


Figure 12.30 *Monocular depth inference (shown as color-coded normal maps) from images in the NYU Depth Dataset V2 (Wang, Geraghty* et al. *2020) © 2020 IEEE.*

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

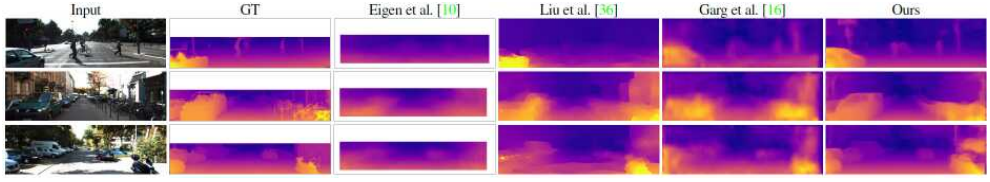


Figure 12.31 *Monocular depth map estimates from images in the KITTI dataset (Godard, Mac Aodha, and Brostow 2017) © 2017 IEEE.*

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 co-planarity 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.