TEXTS IN COMPUTER SCIENCE

Computer Vision

Algorithms and Applications

Richard Szeliski

Springer

Richard Szeliski

Computer Vision

Algorithms and Applications

Springer

Dr. Richard Szeliski
Microsoft Research
One Microsoft Way
98052-6399 Redmond
Washington
USA
szeliski@microsoft.com

ISSN 1868-0941 e-ISSN 1868-095X
ISBN 978-1-84882-934-3 e-ISBN 978-1-84882-935-0
DOI 10.1007/978-1-84882-935-0
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2010936817

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The seeds for this book were first planted in 2001 when Steve Seitz at the University of Washington invited me to co-teach a course called "Computer Vision for Computer Graphics". At that time, computer vision techniques were increasingly being used in computer graphics to create image-based models of real-world objects, to create visual effects, and to merge real-world imagery using computational photography techniques. Our decision to focus on the applications of computer vision to fun problems such as image stitching and photo-based 3D modeling from personal photos seemed to resonate well with our students.

Since that time, a similar syllabus and project-oriented course structure has been used to teach general computer vision courses both at the University of Washington and at Stanford. (The latter was a course I co-taught with David Fleet in 2003.) Similar curricula have been adopted at a number of other universities and also incorporated into more specialized courses on computational photography. (For ideas on how to use this book in your own course, please see Table 1.1 in Section 1.4.)

This book also reflects my 20 years' experience doing computer vision research in corporate research labs, mostly at Digital Equipment Corporation's Cambridge Research Lab and at Microsoft Research. In pursuing my work, I have mostly focused on problems and solution techniques (algorithms) that have practical real-world applications and that work well in practice. Thus, this book has more emphasis on basic techniques that work under real-world conditions and less on more esoteric mathematics that has intrinsic elegance but less practical applicability.

This book is suitable for teaching a senior-level undergraduate course in computer vision to students in both computer science and electrical engineering. I prefer students to have either an image processing or a computer graphics course as a prerequisite so that they can spend less time learning general background mathematics and more time studying computer vision techniques. The book is also suitable for teaching graduate-level courses in computer vision (by delving into the more demanding application and algorithmic areas) and as a general reference to fundamental techniques and the recent research literature. To this end, I have attempted wherever possible to at least cite the newest research in each sub-field, even if the technical details are too complex to cover in the book itself.

In teaching our courses, we have found it useful for the students to attempt a number of small implementation projects, which often build on one another, in order to get them used to working with real-world images and the challenges that these present. The students are then asked to choose an individual topic for each of their small-group, final projects. (Sometimes these projects even turn into conference papers!) The exercises at the end of each chapter contain numerous suggestions for smaller mid-term projects, as well as more open-ended

problems whose solutions are still active research topics. Wherever possible, I encourage students to try their algorithms on their own personal photographs, since this better motivates them, often leads to creative variants on the problems, and better acquaints them with the variety and complexity of real-world imagery.

In formulating and solving computer vision problems, I have often found it useful to draw inspiration from three high-level approaches:

- **Scientific:** build detailed models of the image formation process and develop mathematical techniques to invert these in order to recover the quantities of interest (where necessary, making simplifying assumption to make the mathematics more tractable).
- **Statistical:** use probabilistic models to quantify the prior likelihood of your unknowns and the noisy measurement processes that produce the input images, then infer the best possible estimates of your desired quantities and analyze their resulting uncertainties. The inference algorithms used are often closely related to the optimization techniques used to invert the (scientific) image formation processes.
- **Engineering:** develop techniques that are simple to describe and implement but that are also known to work well in practice. Test these techniques to understand their limitation and failure modes, as well as their expected computational costs (run-time performance).

These three approaches build on each other and are used throughout the book.

My personal research and development philosophy (and hence the exercises in the book) have a strong emphasis on *testing* algorithms. It's too easy in computer vision to develop an algorithm that does something *plausible* on a few images rather than something *correct*. The best way to validate your algorithms is to use a three-part strategy.

First, test your algorithm on clean synthetic data, for which the exact results are known. Second, add noise to the data and evaluate how the performance degrades as a function of noise level. Finally, test the algorithm on real-world data, preferably drawn from a wide variety of sources, such as photos found on the Web. Only then can you truly know if your algorithm can deal with real-world complexity, i.e., images that do not fit some simplified model or assumptions.

In order to help students in this process, this books comes with a large amount of supplementary material, which can be found on the book's Web site http://szeliski.org/Book. This material, which is described in Appendix C, includes:

- pointers to commonly used data sets for the problems, which can be found on the Web
- pointers to software libraries, which can help students get started with basic tasks such as reading/writing images or creating and manipulating images
- slide sets corresponding to the material covered in this book
- a BibTeX bibliography of the papers cited in this book.

The latter two resources may be of more interest to instructors and researchers publishing new papers in this field, but they will probably come in handy even with regular students. Some of the software libraries contain implementations of a wide variety of computer vision algorithms, which can enable you to tackle more ambitious projects (with your instructor's consent).

Chapter 11

Stereo correspondence

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_11, © Springer-Verlag London Limited 2011

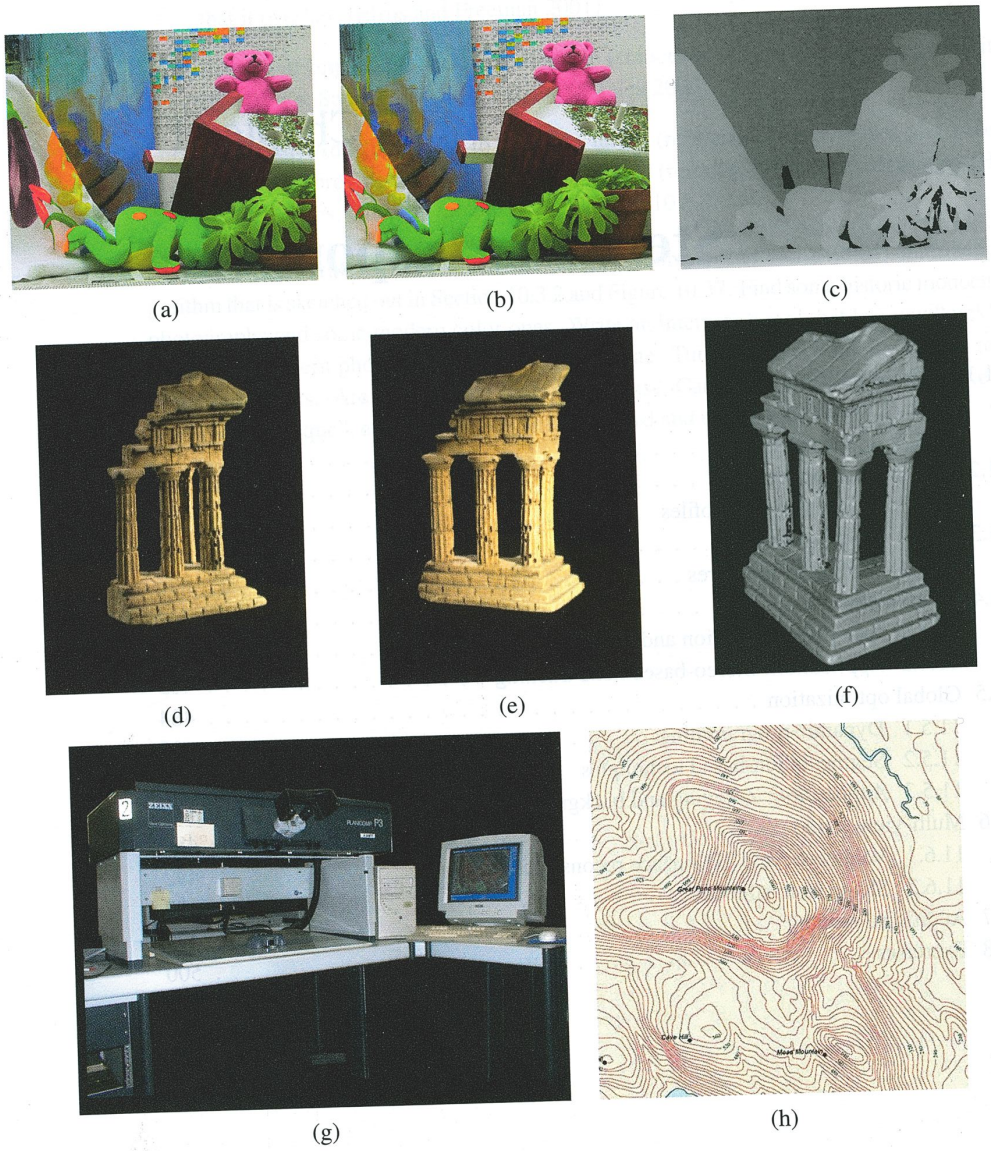


Figure 11.1 Stereo reconstruction techniques can convert (a–b) a pair of images into (c) a depth map (http://vision.middlebury.edu/stereo/data/scenes2003/) or (d–e) a sequence of images into (f) a 3D model (http://vision.middlebury.edu/mview/data/). (g) An analytical stereo plotter, courtesy of Kenney Aerial Mapping, Inc., can generate (h) contour plots.

Stereo matching is the process of taking two or more images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into 3D depths. In Chapters 6–7, 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.

Why are people interested in 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 11.1a–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 11.1), 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 (Figure 11.1g). 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, Diebel *et al.* 2006), and continues to be one of the most active research areas. 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, Dahlkamp *et al.* 2006), as well as view interpolation and image-based rendering (Figure 11.2a–d), 3D model building (Figure 11.2e–f and h–j), and mixing live action with computer-generated imagery (Figure 11.2g).

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 11.1 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.

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

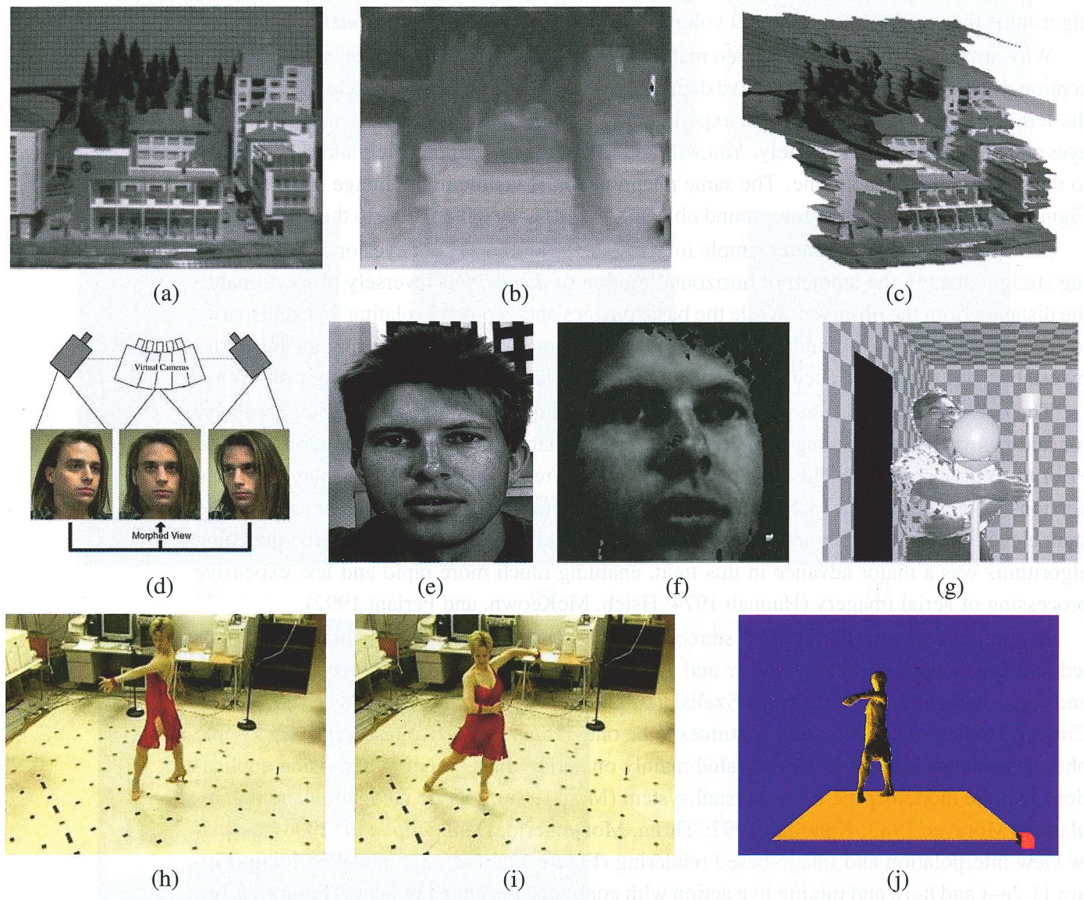


Figure 11.2 Applications of stereo vision: (a) input image, (b) computed depth map, and (c) new view generation from multi-view stereo (Matthies, Kanade, and Szeliski 1989) © 1989 Springer; (d) view morphing between two images (Seitz and Dyer 1996) © 1996 ACM; (e–f) 3D face modeling (images courtesy of Frédéric Devernay); (g) *z-keying* live and computer-generated imagery (Kanade, Yoshida, Oda *et al.* 1996) © 1996 IEEE; (h–j) building 3D surface models from multiple video streams in Virtualized Reality (Kanade, Rander, and Narayanan 1997).

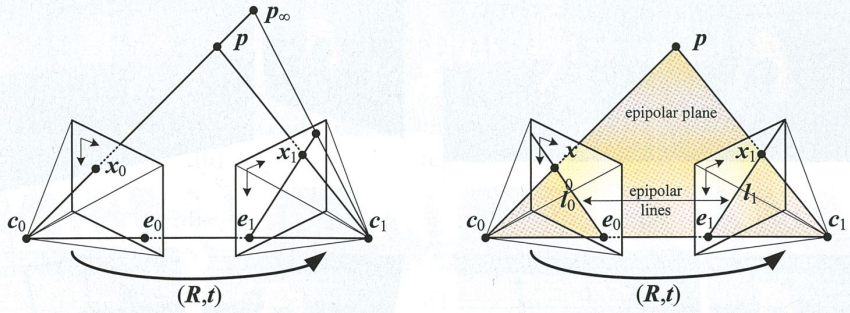


Figure 11.3 Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.

Next, we briefly survey techniques for the *sparse* stereo matching of interest points and edge-like features (Section 11.2). 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 11.1c). This involves first selecting a pixel matching criterion (Section 11.3) and then using either local area-based aggregation (Section 11.4) or global optimization (Section 11.5) to help disambiguate potential matches. In Section 11.6, we discuss *multi-view stereo* methods that aim to reconstruct a complete 3D model instead of just a single disparity image (Figure 11.1d–f).

11.1 Epipolar geometry

Given a pixel in one image, how can we compute its correspondence in the other image? In Chapter 8, we saw that a variety of search techniques can be used to match pixels based on 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 7.2).

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 11.3a shows how a pixel in one image $\boldsymbol{x}_0$ 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 $\boldsymbol{p}_\infty$ and at the other end by the projection of the original camera center $\boldsymbol{c}_0$ into the second camera, which is known as the *epipole* $\boldsymbol{e}_1$. 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 $\boldsymbol{e}_0$. Extending both line segments to infinity, we get a pair of corresponding *epipolar lines* (Figure 11.3b), which are the intersection of the two image planes with the *epipolar plane* that passes through both camera centers $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ as well as the point of interest $\boldsymbol{p}$ (Faugeras and Luong 2001; Hartley and Zisserman 2004).

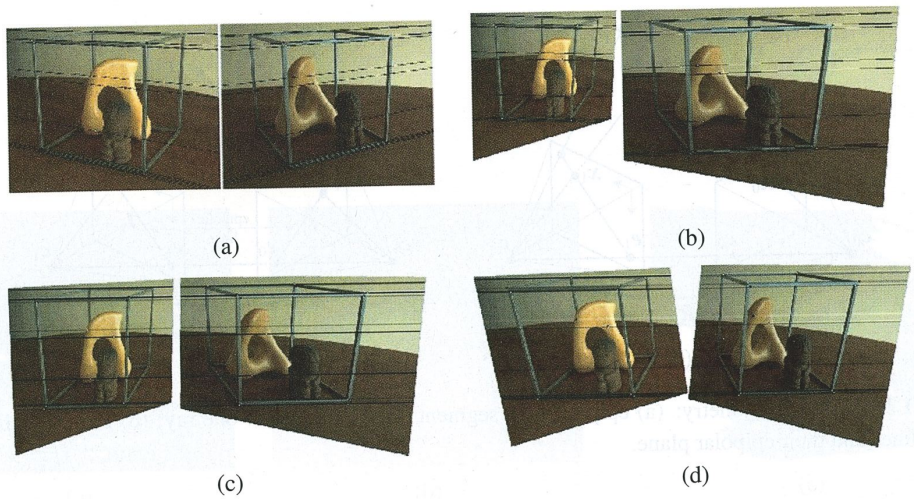


Figure 11.4 The multi-stage stereo rectification algorithm of Loop and Zhang (1999) © 1999 IEEE. (a) Original image pair overlaid with several epipolar lines; (b) images transformed so that epipolar lines are parallel; (c) images rectified so that epipolar lines are horizontal and in vertial correspondence; (d) final rectification that minimizes horizontal distortions.

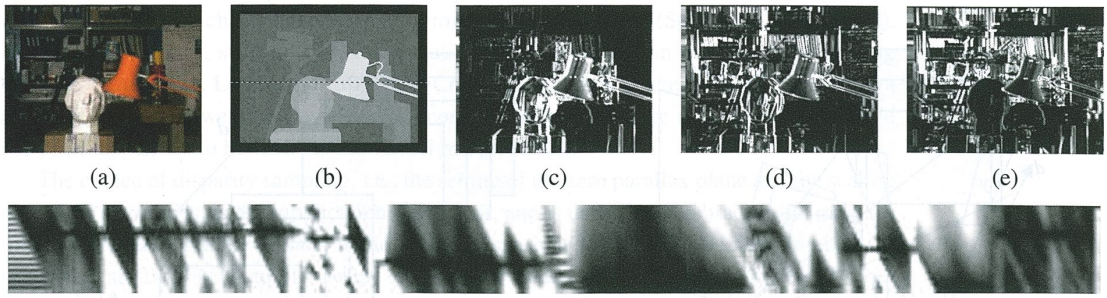
11.1.1 Rectification

As we saw in Section 7.2, 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 8.4), 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 11.4).

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

[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, Curless *et al.* 2007) may be preferable.



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

(f)

Figure 11.5 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).

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 11.1.) 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,

$$d = f\frac{B}{Z}, \tag{11.1}$$

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

$$x' = x + d(x, y), \quad y' = y \tag{11.2}$$

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 11.5). 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).

[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 studied phenomenon, but vertical disparity is possible if the eyes are verged.

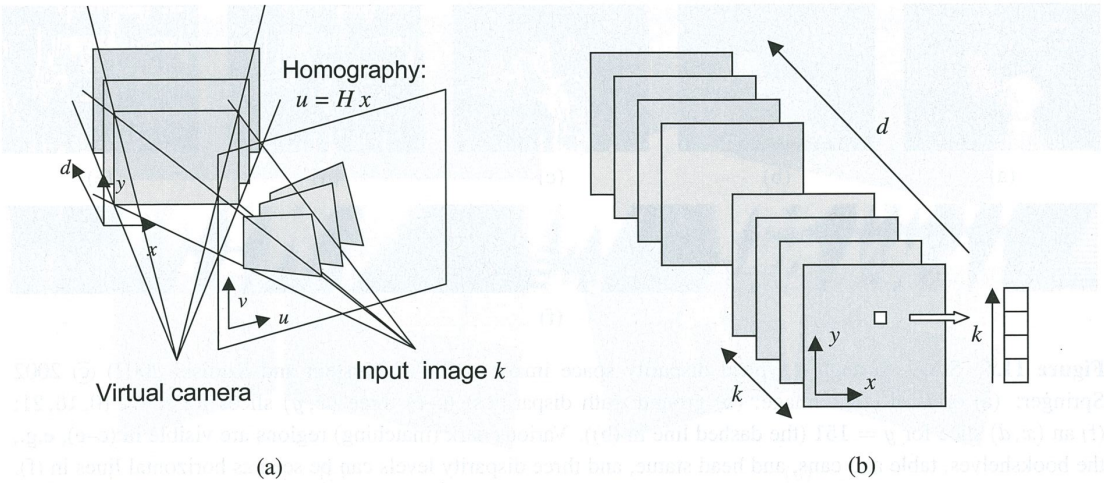


Figure 11.6 Sweeping a set of planes through a scene (Szeliski and Golland 1999) © 1999 Springer: (a) The set of planes seen from a virtual camera induces a set of homographies in any other source (input) camera image. (b) The warped images from all the other cameras can be stacked into a generalized disparity space volume $\tilde{I}(x, y, d, k)$ indexed by pixel location (x, y), disparity d, and camera k.

11.1.2 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 11.6). This process is commonly known as the *plane sweep* algorithm (Collins 1996; Szeliski and Golland 1999; Saito and Kanade 1999).

As we saw in Section 2.1.5, 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 $\tilde{\boldsymbol{P}}$ can be set to an arbitrary plane equation $\boldsymbol{p}_3 = s_3[\hat{\boldsymbol{n}}_0|c_0]$. The resulting four-dimensional projective transform (*collineation*) (2.68) maps 3D world points $\boldsymbol{p} = (X, Y, Z, 1)$ into screen coordinates $\boldsymbol{x}_s = (x_s, y_s, 1, d)$, 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 11.6a, corresponds to mapping each input image into the *virtual camera* $\tilde{\boldsymbol{P}}$ defining the disparity space through a series of homographies (2.68–2.71),

$$\tilde{\boldsymbol{x}}_k \sim \tilde{\boldsymbol{P}}_k \tilde{\boldsymbol{P}}^{-1} \boldsymbol{x}_s = \tilde{\boldsymbol{H}}_k \tilde{\boldsymbol{x}} + \boldsymbol{t}_k d = (\tilde{\boldsymbol{H}}_k + \boldsymbol{t}_k [0\ 0\ d]) \tilde{\boldsymbol{x}}, \tag{11.3}$$

as shown in Figure 2.12b, where $\tilde{\boldsymbol{x}}_k$ and $\tilde{\boldsymbol{x}}$ are the homogeneous pixel coordinates in the source and virtual (reference) images (Szeliski and Golland 1999). The members of the family of homographies $\tilde{\boldsymbol{H}}_k(d) = \tilde{\boldsymbol{H}}_k + \boldsymbol{t}_k[0\ 0\ d]$, which are parametererized 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 *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 13.1 and 13.2). In other applications, such as view interpolation for gaze correction in video-conferencing (Section 11.4.2) (Ott, Lewis, and Cox 1993; Criminisi, Shotton, Blake *et al.* 2003), a camera centrally located between the two input cameras is preferable, since 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 11.6b) (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

$$C(x, y, d) = \sum_k \rho(\tilde{I}(x, y, d, k) - I_r(x, y)). \tag{11.4}$$

However, it is also possible to compute alternative statistics such as robust variance, focus, or entropy (Section 11.3.1) (Vaish, Szeliski, Zitnick *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 11.6.

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 (Chapter 9), are often used (Ishiguro, Yamamoto, and Tsuji 1992; Kang and Szeliski 1997; Shum and Szeliski 1999; Li, Shum, Tang *et al.* 2004; Zheng, Kang, Cohen *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 11.5 shows examples of slices through a typical DSI. More figures of this kind can be found in the paper by Bobick and Intille (1999).

11.2 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

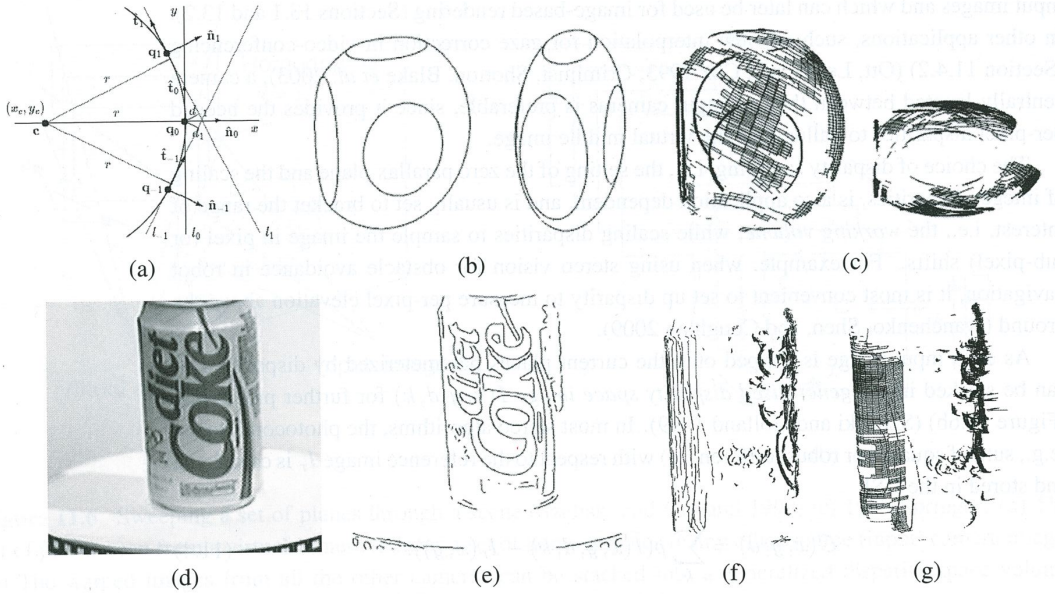


Figure 11.7 Surface reconstruction from occluding contours (Szeliski and Weiss 1998) © 2002 Springer: (a) circular arc fitting in the epipolar plane; (b) synthetic example of an ellipsoid with a truncated side and elliptic surface markings; (c) partially reconstructed surface mesh seen from an oblique 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.)

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 3.7.1 and 12.3.1.

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). 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, Curless *et al.* 2007) or free-space reasoning (Taylor 2003), as described in more detail in Section 11.6.

11.2.1 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 11.7) 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 11.7a) and therefore obtain semi-dense curved surface meshes directly from the matches (Figures 11.7c 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 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 11.7b 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 11.7a. We then fit an *osculating circle* to the projected lines, parameterizing the circle by its centerpoint $\boldsymbol{c} = (x_c, y_c)$ and radius r,

$$c_i x_c + s_i y_c + r = d_i, \tag{11.5}$$

where $c_i = \hat{\boldsymbol{t}}_i \cdot \hat{\boldsymbol{t}}_0$ and $s_i = -\hat{\boldsymbol{t}}_i \cdot \hat{\boldsymbol{n}}_0$ are the cosine and sine of the angle between viewing ray i and the central viewing ray 0, and $d_i = (\boldsymbol{q}_i - \boldsymbol{q}_0) \cdot \hat{\boldsymbol{n}}_0$ is the perpendicular distance between viewing ray i and the local origin $\boldsymbol{q}_0$, which is a point chosen on the central viewing ray close to the line intersections (Szeliski and Weiss 1998). The resulting set of linear equations can be solved using least squares, and the quality of the solution (residual error) can be used to check for erroneous correspondences.

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 11.7c 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 12.3.1).

11.3 Dense correspondence

While sparse matching algorithms are still occasionally used, most stereo matching algorithms today focus on dense correspondence, since this is required for applications such as

image-based rendering or modeling. This problem is more challenging than sparse correspondence, since 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 11.4), 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 11.3.1).)

Global algorithms, on the other hand, make explicit smoothness assumptions and then solve a a global optimization problem (Section 11.5). 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 distinctions 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, Hanna *et al.* 1992).

11.3.1 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 8.1 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 (8.2), including truncated quadratics and contaminated Gaussians, have been proposed (Black and Anandan 1996; Black and Rangarajan 1996; Scharstein and Szeliski 1998). These measures are useful because they limit the influence of mismatches during aggregation. Vaish, Szeliski, Zitnick *et al.* (2006) compare a number of such robust measures, including a new one based on the entropy of the pixel values at a particular disparity hypothesis (Zitnick, Kang, Uyttendaele *et al.* 2004), which is particularly useful in multi-view stereo.

Other traditional matching costs include normalized cross-correlation (8.11) (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 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, Uyttendaele *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.

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).

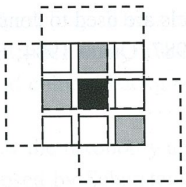


Figure 11.8 Shiftable window (Scharstein and Szeliski 2002) © 2002 Springer. The effect of trying all 3×3 shifted windows around the black pixel is the same as taking the minimum matching score across all *centered* (non-shifted) windows in the same neighborhood. (For clarity, only three of the neighboring shifted windows are shown here.)

In order 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; 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.

11.4 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), or the results of color-based segmentation (Yoon and Kweon 2006; Tombari, Mattoccia, Di Stefano *et al.* 2008). 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 more recent work (which includes visibility and occlusion reasoning) by Zitnick and Kanade (2000).

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

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

Figure 11.9 Aggregation window sizes and weights adapted to image content (Tombari, Mattoccia, Di Stefano *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.

Aggregation with a fixed support region can be performed using 2D or 3D convolution,

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d), \tag{11.6}$$

or, in the case of rectangular windows, using efficient moving average box-filters (Section 3.2.2) (Kanade, Yoshida, Oda *et al.* 1996; Kimura, Shinbo, Yamaguchi *et al.* 1999). Shiftable windows can also be implemented efficiently using a separable sliding min-filter (Figure 11.8) (Scharstein 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, since windows must be large enough to contain sufficient texture and yet small enough so that they do not straddle depth discontinuities (Figure 11.9). 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, Wang *et al.* (2007) and Tombari, Mattoccia, Di Stefano *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 particular, 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 bilinear filtering (Figure 11.9c). (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.

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

[5] More recent and extensive results from Tombari, Mattoccia, Di Stefano *et al.* (2008) can be found at http://www.vision.deis.unibo.it/spe/SPEHome.aspx.