TEXTS IN COMPUTER SCIENCE

Computer Vision

Algorithms and Applications

Richard Szeliski

Springer

Texts in Computer Science

Richard Szeliski

Computer Vision

Algorithms and Applications

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)

Chapter 2

Image formation

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_2, © Springer-Verlag London Limited 2011

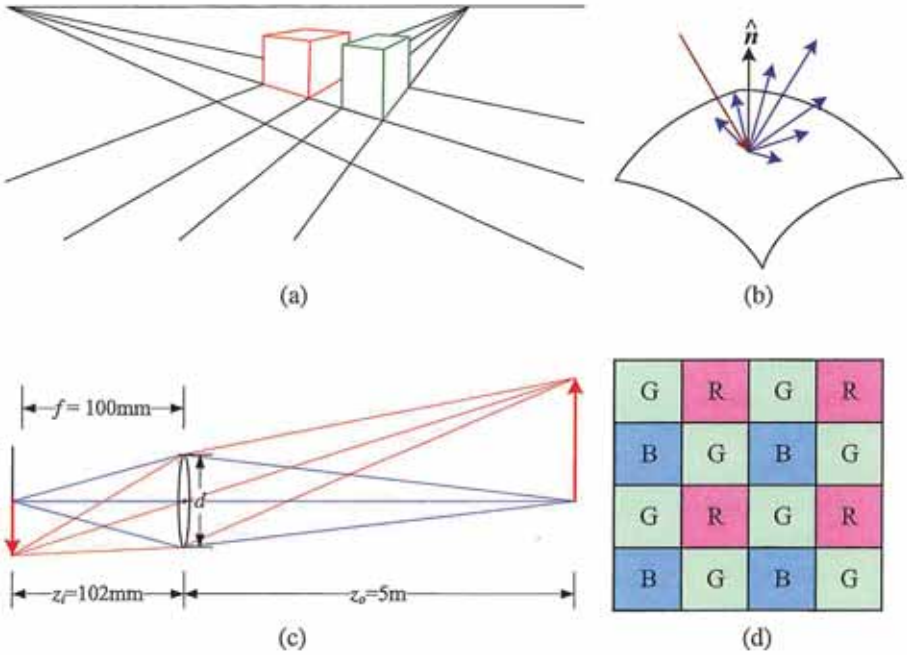


Figure 2.1 A few components of the image formation process: (a) perspective projection; (b) light scattering when hitting a surface; (c) lens optics; (d) Bayer color filter array.

Before we can intelligently analyze and manipulate images, we need to establish a vocabulary for describing the geometry of a scene. We also need to understand the image formation process that produced a particular image given a set of lighting conditions, scene geometry, surface properties, and camera optics. In this chapter, we present a simplified model of such an image formation process.

Section 2.1 introduces the basic geometric primitives used throughout the book (points, lines, and planes) and the *geometric* transformations that project these 3D quantities into 2D image features (Figure 2.1a). Section 2.2 describes how lighting, surface properties (Figure 2.1b), and camera *optics* (Figure 2.1c) interact in order to produce the color values that fall onto the image sensor. Section 2.3 describes how continuous color images are turned into discrete digital *samples* inside the image sensor (Figure 2.1d) and how to avoid (or at least characterize) sampling deficiencies, such as aliasing.

The material covered in this chapter is but a brief summary of a very rich and deep set of topics, traditionally covered in a number of separate fields. A more thorough introduction to the geometry of points, lines, planes, and projections can be found in textbooks on multi-view geometry (Hartley and Zisserman 2004; Faugeras and Luong 2001) and computer graphics (Foley, van Dam, Feiner *et al.* 1995). The image formation (synthesis) process is traditionally taught as part of a computer graphics curriculum (Foley, van Dam, Feiner *et al.* 1995; Glassner 1995; Watt 1995; Shirley 2005) but it is also studied in physics-based computer vision (Wolff, Shafer, and Healey 1992a). The behavior of camera lens systems is studied in optics (Möller 1988; Hecht 2001; Ray 2002). Two good books on color theory are (Wyszecki and Stiles 2000; Healey and Shafer 1992), with (Livingstone 2008) providing a more fun and informal introduction to the topic of color perception. Topics relating to sampling and aliasing are covered in textbooks on signal and image processing (Crane 1997; Jähne 1997; Oppenheim and Schafer 1996; Oppenheim, Schafer, and Buck 1999; Pratt 2007; Russ 2007; Burger and Burge 2008; Gonzales and Woods 2008).

A note to students: If you have already studied computer graphics, you may want to skim the material in Section 2.1, although the sections on projective depth and object-centered projection near the end of Section 2.1.5 may be new to you. Similarly, physics students (as well as computer graphics students) will mostly be familiar with Section 2.2. Finally, students with a good background in image processing will already be familiar with sampling issues (Section 2.3) as well as some of the material in Chapter 3.

2.1 Geometric primitives and transformations

In this section, we introduce the basic 2D and 3D primitives used in this textbook, namely points, lines, and planes. We also describe how 3D features are projected into 2D features. More detailed descriptions of these topics (along with a gentler and more intuitive introduction) can be found in textbooks on multiple-view geometry (Hartley and Zisserman 2004; Faugeras and Luong 2001).

2.1.1 Geometric primitives

Geometric primitives form the basic building blocks used to describe three-dimensional shapes. In this section, we introduce points, lines, and planes. Later sections of the book discuss

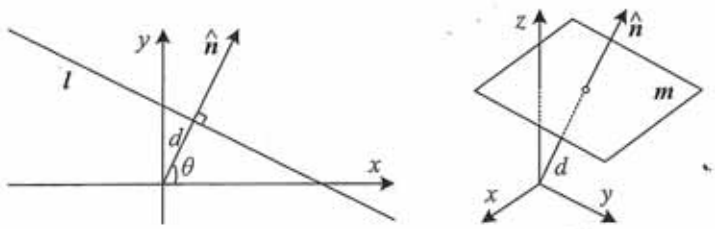


Figure 2.2 (a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal $\hat{\boldsymbol{n}}$ and distance to the origin d.

curves (Sections 5.1 and 11.2), surfaces (Section 12.3), and volumes (Section 12.5).

2D points. 2D points (pixel coordinates in an image) can be denoted using a pair of values, $\boldsymbol{x} = (x, y) \in \mathcal{R}^2$, or alternatively,

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.1}$$

(As stated in the introduction, we use the $(x_1, x_2, \ldots)$ notation to denote column vectors.)

2D points can also be represented using *homogeneous coordinates*, $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$, where vectors that differ only by scale are considered to be equivalent. $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$ is called the 2D *projective space*.

A homogeneous vector $\tilde{\boldsymbol{x}}$ can be converted back into an *inhomogeneous* vector $\boldsymbol{x}$ by dividing through by the last element $\tilde{w}$, i.e.,

$$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\boldsymbol{x}}, \tag{2.2}$$

where $\bar{\boldsymbol{x}} = (x, y, 1)$ is the *augmented vector*. Homogeneous points whose last element is $\tilde{w} = 0$ are called *ideal points* or *points at infinity* and do not have an equivalent inhomogeneous representation.

2D lines. 2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$. The corresponding *line equation* is

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0. \tag{2.3}$$

We can normalize the line equation vector so that $\boldsymbol{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$. In this case, $\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the line and d is its distance to the origin (Figure 2.2). (The one exception to this normalization is the *line at infinity* $\tilde{\boldsymbol{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.)

We can also express $\hat{\boldsymbol{n}}$ as a function of rotation angle θ, $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y) = (\cos\theta, \sin\theta)$ (Figure 2.2a). This representation is commonly used in the *Hough transform* line-finding algorithm, which is discussed in Section 4.3.2. The combination (θ, d) is also known as *polar coordinates*.

When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2, \tag{2.4}$$

where $\times$ is the cross product operator. Similarly, the line joining two points can be written as

$$\bar{l} = \bar{x}_1 \times \bar{x}_2. \tag{2.5}$$

When trying to fit an intersection point to multiple lines or, conversely, a line to multiple points, least squares techniques (Section 6.1.1 and Appendix A.2) can be used, as discussed in Exercise 2.1.

2D conics. There are other algebraic curves that can be expressed with simple polynomial homogeneous equations. For example, the *conic sections* (so called because they arise as the intersection of a plane and a 3D cone) can be written using a *quadric* equation

$$\bar{x}^T Q \bar{x} = 0. \tag{2.6}$$

Quadric equations play useful roles in the study of multi-view geometry and camera calibration (Hartley and Zisserman 2004; Faugeras and Luong 2001) but are not used extensively in this book.

3D points. Point coordinates in three dimensions can be written using inhomogeneous coordinates $\boldsymbol{x} = (x, y, z) \in \mathcal{R}^3$ or homogeneous coordinates $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y}, \bar{z}, \bar{w}) \in \mathcal{P}^3$. As before, it is sometimes useful to denote a 3D point using the augmented vector $\bar{\boldsymbol{x}} = (x, y, z, 1)$ with $\bar{\boldsymbol{x}} = \bar{w}\bar{\boldsymbol{x}}$.

3D planes. 3D planes can also be represented as homogeneous coordinates $\bar{\boldsymbol{m}} = (a, b, c, d)$ with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \bar{\boldsymbol{m}} = ax + by + cz + d = 0. \tag{2.7}$$

We can also normalize the plane equation as $\boldsymbol{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$. In this case, $\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the plane and d is its distance to the origin (Figure 2.2b). As with the case of 2D lines, the *plane at infinity* $\bar{\boldsymbol{m}} = (0, 0, 0, 1)$, which contains all the points at infinity, cannot be normalized (i.e., it does not have a unique normal or a finite distance).

We can express $\hat{\boldsymbol{n}}$ as a function of two angles (θ, ϕ),

$$\hat{\boldsymbol{n}} = (\cos\theta\cos\phi, \sin\theta\cos\phi, \sin\phi), \tag{2.8}$$

i.e., using *spherical coordinates*, but these are less commonly used than polar coordinates since they do not uniformly sample the space of possible normal vectors.

3D lines. Lines in 3D are less elegant than either lines in 2D or planes in 3D. One possible representation is to use two points on the line, $(\boldsymbol{p}, \boldsymbol{q})$. Any other point on the line can be expressed as a linear combination of these two points

$$\boldsymbol{r} = (1 - \lambda)\boldsymbol{p} + \lambda\boldsymbol{q}, \tag{2.9}$$

as shown in Figure 2.3. If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

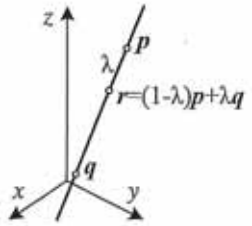


Figure 2.3 3D line equation, $\boldsymbol{r} = (1-\lambda)\boldsymbol{p} + \lambda\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

$$\bar{\boldsymbol{r}} = \mu\bar{\boldsymbol{p}} + \lambda\bar{\boldsymbol{q}}. \tag{2.10}$$

A special case of this is when the second point is at infinity, i.e., $\bar{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$. Here, we see that $\hat{\boldsymbol{d}}$ is the *direction* of the line. We can then re-write the inhomogeneous 3D line equation as

$$\boldsymbol{r} = \boldsymbol{p} + \lambda\hat{\boldsymbol{d}}. \tag{2.11}$$

A disadvantage of the endpoint representation for 3D lines is that it has too many degrees of freedom, i.e., six (three for each endpoint) instead of the four degrees that a 3D line truly has. However, if we fix the two points on the line to lie in specific planes, we obtain a representation with four degrees of freedom. For example, if we are representing nearly vertical lines, then $z = 0$ and $z = 1$ form two suitable planes, i.e., the (x, y) coordinates in both planes provide the four coordinates describing the line. This kind of two-plane parameterization is used in the *light field* and *Lumigraph* image-based rendering systems described in Chapter 13 to represent the collection of rays seen by a camera as it moves in front of an object. The two-endpoint representation is also useful for representing line segments, even when their exact endpoints cannot be seen (only guessed at).

If we wish to represent all possible lines without bias towards any particular orientation, we can use *Plücker coordinates* (Hartley and Zisserman 2004, Chapter 2; Faugeras and Luong 2001, Chapter 3). These coordinates are the six independent non-zero entries in the 4×4 skew symmetric matrix

$$\boldsymbol{L} = \bar{\boldsymbol{p}}\bar{\boldsymbol{q}}^T - \bar{\boldsymbol{q}}\bar{\boldsymbol{p}}^T, \tag{2.12}$$

where $\bar{\boldsymbol{p}}$ and $\bar{\boldsymbol{q}}$ are *any* two (non-identical) points on the line. This representation has only four degrees of freedom, since $\boldsymbol{L}$ is homogeneous and also satisfies $det(\boldsymbol{L}) = 0$, which results in a quadratic constraint on the Plücker coordinates.

In practice, the minimal representation is not essential for most applications. An adequate model of 3D lines can be obtained by estimating their direction (which may be known ahead of time, e.g., for architecture) and some point within the visible portion of the line (see Section 7.5.1) or by using the two endpoints, since lines are most often visible as finite line segments. However, if you are interested in more details about the topic of minimal line parameterizations, Förstner (2005) discusses various ways to infer and model 3D lines in projective geometry, as well as how to estimate the uncertainty in such fitted models.

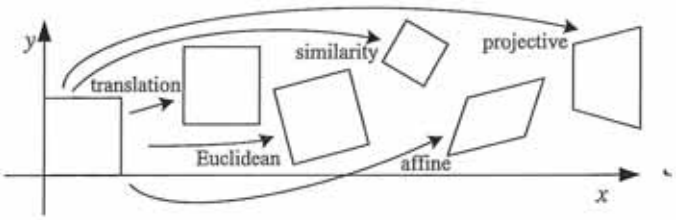


Figure 2.4 Basic set of 2D planar transformations.

3D quadrics. The 3D analog of a conic section is a quadric surface

$$\bar{x}^T Q \bar{x} = 0 \tag{2.13}$$

(Hartley and Zisserman 2004, Chapter 2). Again, while quadric surfaces are useful in the study of multi-view geometry and can also serve as useful modeling primitives (spheres, ellipsoids, cylinders), we do not study them in great detail in this book.

2.1.2 2D transformations

Having defined our basic primitives, we can now turn our attention to how they can be transformed. The simplest transformations occur in the 2D plane and are illustrated in Figure 2.4.

Translation. 2D translations can be written as $x' = x + t$ or

$$x' = \begin{bmatrix} I & t \end{bmatrix} \bar{x} \tag{2.14}$$

where I is the (2×2) identity matrix or

$$\bar{x}' = \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix} \bar{x} \tag{2.15}$$

where $\mathbf{0}$ is the zero vector. Using a 2×3 matrix results in a more compact notation, whereas using a full-rank 3×3 matrix (which can be obtained from the 2×3 matrix by appending a $[0^T\ 1]$ row) makes it possible to chain transformations using matrix multiplication. Note that in any equation where an augmented vector such as $\bar{x}$ appears on both sides, it can always be replaced with a full homogeneous vector $\tilde{x}$.

Rotation + translation. This transformation is also known as *2D rigid body motion* or the *2D Euclidean transformation* (since Euclidean distances are preserved). It can be written as $x' = Rx + t$ or

$$x' = \begin{bmatrix} R & t \end{bmatrix} \bar{x} \tag{2.16}$$

where

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.17}$$

is an orthonormal rotation matrix with $RR^T = I$ and $|R| = 1$.

Scaled rotation. Also known as the *similarity transform*, this transformation can be expressed as $x' = sRx + t$ where s is an arbitrary scale factor. It can also be written as

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x}, \tag{2.18}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

Affine. The affine transformation is written as $x' = A\bar{x}$, where A is an arbitrary 2×3 matrix, i.e.,

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{x}. \tag{2.19}$$

Parallel lines remain parallel under affine transformations.

Projective. This transformation, also known as a *perspective transform* or *homography*, operates on homogeneous coordinates,

$$\tilde{x}' = \tilde{H}\tilde{x}, \tag{2.20}$$

where $\tilde{H}$ is an arbitrary 3×3 matrix. Note that $\tilde{H}$ is homogeneous, i.e., it is only defined up to a scale, and that two $\tilde{H}$ matrices that differ only by scale are equivalent. The resulting homogeneous coordinate $\bar{x}'$ must be normalized in order to obtain an inhomogeneous result x, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \text{ and } y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \tag{2.21}$$

Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

Hierarchy of 2D transformations. The preceding set of transformations are illustrated in Figure 2.4 and summarized in Table 2.1. The easiest way to think of them is as a set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors. Hartley and Zisserman (2004) contains a more detailed description of the hierarchy of 2D planar transformations.

The above transformations form a nested set of *groups*, i.e., they are closed under composition and have an inverse that is a member of the same group. (This will be important later when applying these transformations to images in Section 3.6.) Each (simpler) group is a subset of the more complex group below it.

Co-vectors. While the above transformations can be used to transform points in a 2D plane, can they also be used directly to transform a line equation? Consider the homogeneous equation $\tilde{l} \cdot \tilde{x} = 0$. If we transform $x' = \tilde{H}x$, we obtain

$$\tilde{l}' \cdot \tilde{x}' = \tilde{l}'^T \tilde{H}\tilde{x} = (\tilde{H}^T \tilde{l}')^T \tilde{x} = \tilde{l} \cdot \tilde{x} = 0, \tag{2.22}$$

i.e., $\tilde{l}' = \tilde{H}^{-T}\tilde{l}$. Thus, the action of a projective transformation on a *co-vector* such as a 2D line or 3D normal can be represented by the transposed inverse of the matrix, which is equivalent to the *adjoint* of $\tilde{H}$, since projective transformation matrices are homogeneous. Jim

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[\mathbf{0}^T\ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Blinn (1998) describes (in Chapters 9 and 10) the ins and outs of notating and manipulating co-vectors.

While the above transformations are the ones we use most extensively, a number of additional transformations are sometimes used.

Stretch/squash. This transformation changes the aspect ratio of an image,

$$\begin{aligned} x' &= s_x x + t_x \\ y' &= s_y y + t_y, \end{aligned}$$

and is a restricted form of an affine transformation. Unfortunately, it does not nest cleanly with the groups listed in Table 2.1.

Planar surface flow. This eight-parameter transformation (Horn 1986; Bergen, Anandan, Hanna *et al.* 1992; Girod, Greiner, and Niemann 2000),

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_6 x^2 + a_7 xy \\ y' &= a_3 + a_4 x + a_5 y + a_7 x^2 + a_6 xy, \end{aligned}$$

arises when a planar surface undergoes a small 3D motion. It can thus be thought of as a small motion approximation to a full homography. Its main attraction is that it is *linear* in the motion parameters, a_k, which are often the quantities being estimated.

Bilinear interpolant. This eight-parameter transform (Wolberg 1990),

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_6 xy \\ y' &= a_3 + a_4 x + a_5 y + a_7 xy, \end{aligned}$$

can be used to interpolate the deformation due to the motion of the four corner points of a square. (In fact, it can interpolate the motion of any four non-collinear points.) While

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times4}$	15	straight lines	

Table 2.2 Hierarchy of 3D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 3×4 matrices are extended with a fourth $[\mathbf{0}^T\ 1]$ row to form a full 4×4 matrix for homogeneous coordinate transformations. The mnemonic icons are drawn in 2D but are meant to suggest transformations occurring in a full 3D cube.

the deformation is linear in the motion parameters, it does not generally preserve straight lines (only lines parallel to the square axes). However, it is often quite useful, e.g., in the interpolation of sparse grids using splines (Section 8.3).

2.1.3 3D transformations

The set of three-dimensional coordinate transformations is very similar to that available for 2D transformations and is summarized in Table 2.2. As in 2D, these transformations form a nested set of groups. Hartley and Zisserman (2004, Section 2.4) give a more detailed description of this hierarchy.

Translation. 3D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.23}$$

where $\boldsymbol{I}$ is the (3×3) identity matrix and $\mathbf{0}$ is the zero vector.

Rotation + translation. Also known as 3D *rigid body motion* or the 3D *Euclidean transformation*, it can be written as $\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.24}$$

where $\boldsymbol{R}$ is a 3×3 orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$. Note that sometimes it is more convenient to describe a rigid motion using

$$\boldsymbol{x}' = \boldsymbol{R}(\boldsymbol{x} - \boldsymbol{c}) = \boldsymbol{R}\boldsymbol{x} - \boldsymbol{R}\boldsymbol{c}, \tag{2.25}$$

where $\boldsymbol{c}$ is the center of rotation (often the camera center).

Compactly parameterizing a 3D rotation is a non-trivial task, which we describe in more detail below.

Scaled rotation. The 3D *similarity transform* can be expressed as $x' = sRx + t$ where s is an arbitrary scale factor. It can also be written as

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \bar{x}. \tag{2.26}$$

This transformation preserves angles between lines and planes.

Affine. The affine transform is written as $x' = A\bar{x}$, where A is an arbitrary 3×4 matrix, i.e.,

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \bar{x}. \tag{2.27}$$

Parallel lines and planes remain parallel under affine transformations.

Projective. This transformation, variously known as a *3D perspective transform*, *homography*, or *collineation*, operates on homogeneous coordinates,

$$\tilde{x}' = \tilde{H}\tilde{x}, \tag{2.28}$$

where $\tilde{H}$ is an arbitrary 4×4 homogeneous matrix. As in 2D, the resulting homogeneous coordinate $\tilde{x}'$ must be normalized in order to obtain an inhomogeneous result x. Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

2.1.4 3D rotations

The biggest difference between 2D and 3D coordinate transformations is that the parameterization of the 3D rotation matrix R is not as straightforward but several possibilities exist.

Euler angles

A rotation matrix can be formed as the product of three rotations around three cardinal axes, e.g., x, y, and z, or x, y, and x. This is generally a bad idea, as the result depends on the order in which the transforms are applied. What is worse, it is not always possible to move smoothly in the parameter space, i.e., sometimes one or more of the Euler angles change dramatically in response to a small change in rotation.[1] For these reasons, we do not even give the formula for Euler angles in this book—interested readers can look in other textbooks or technical reports (Faugeras 1993; Diebel 2006). Note that, in some applications, if the rotations are known to be a set of uni-axial transforms, they can always be represented using an explicit set of rigid transformations.

Axis/angle (exponential twist)

A rotation can be represented by a rotation axis $\hat{n}$ and an angle θ, or equivalently by a 3D vector $\omega = \theta\hat{n}$. Figure 2.5 shows how we can compute the equivalent rotation. First, we project the vector v onto the axis $\hat{n}$ to obtain

$$v_{\parallel} = \hat{n}(\hat{n} \cdot v) = (\hat{n}\hat{n}^T)v, \tag{2.29}$$

[1] In robotics, this is sometimes referred to as *gimbal lock*.

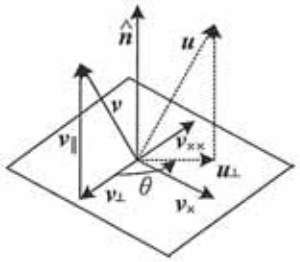


Figure 2.5 Rotation around an axis $\hat{\boldsymbol{n}}$ by an angle θ.

which is the component of $\boldsymbol{v}$ that is not affected by the rotation. Next, we compute the perpendicular residual of $\boldsymbol{v}$ from $\hat{\boldsymbol{n}}$,

$$\boldsymbol{v}_\perp = \boldsymbol{v} - \boldsymbol{v}_\parallel = (\boldsymbol{I} - \hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\boldsymbol{v}. \tag{2.30}$$

We can rotate this vector by 90° using the cross product,

$$\boldsymbol{v}_\times = \hat{\boldsymbol{n}} \times \boldsymbol{v} = [\hat{\boldsymbol{n}}]_\times \boldsymbol{v}, \tag{2.31}$$

where $[\hat{\boldsymbol{n}}]_\times$ is the matrix form of the cross product operator with the vector $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$,

$$[\hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}. \tag{2.32}$$

Note that rotating this vector by another 90° is equivalent to taking the cross product again,

$$\boldsymbol{v}_{\times\times} = \hat{\boldsymbol{n}} \times \boldsymbol{v}_\times = [\hat{\boldsymbol{n}}]_\times^2 \boldsymbol{v} = -\boldsymbol{v}_\perp,$$

and hence

$$\boldsymbol{v}_\parallel = \boldsymbol{v} - \boldsymbol{v}_\perp = \boldsymbol{v} + \boldsymbol{v}_{\times\times} = (\boldsymbol{I} + [\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

We can now compute the in-plane component of the rotated vector $\boldsymbol{u}$ as

$$\boldsymbol{u}_\perp = \cos\theta \boldsymbol{v}_\perp + \sin\theta \boldsymbol{v}_\times = (\sin\theta[\hat{\boldsymbol{n}}]_\times - \cos\theta[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\boldsymbol{u} = \boldsymbol{u}_\perp + \boldsymbol{v}_\parallel = (\boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2)\boldsymbol{v}. \tag{2.33}$$

We can therefore write the rotation matrix corresponding to a rotation by θ around an axis $\hat{\boldsymbol{n}}$ as

$$\boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2, \tag{2.34}$$

which is known as *Rodriguez's formula* (Ayache 1989).

The product of the axis $\hat{\boldsymbol{n}}$ and angle θ, $\boldsymbol{\omega} = \theta\hat{\boldsymbol{n}} = (\omega_x, \omega_y, \omega_z)$, is a minimal representation for a 3D rotation. Rotations through common angles such as multiples of 90° can be represented exactly (and converted to exact matrices) if θ is stored in degrees. Unfortunately,

this representation is not unique, since we can always add a multiple of 360° (2π radians) to θ and get the same rotation matrix. As well, $(\hat{\boldsymbol{n}}, \theta)$ and $(-\hat{\boldsymbol{n}}, -\theta)$ represent the same rotation.

However, for small rotations (e.g., corrections to rotations), this is an excellent choice. In particular, for small (infinitesimal or instantaneous) rotations and θ expressed in radians, Rodriguez's formula simplifies to

$$\boldsymbol{R}(\boldsymbol{\omega}) \approx \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times \approx \boldsymbol{I} + [\theta\hat{\boldsymbol{n}}]_\times = \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix}, \tag{2.35}$$

which gives a nice linearized relationship between the rotation parameters $\boldsymbol{\omega}$ and $\boldsymbol{R}$. We can also write $\boldsymbol{R}(\boldsymbol{\omega})\boldsymbol{v} \approx \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{v}$, which is handy when we want to compute the derivative of $\boldsymbol{Rv}$ with respect to $\boldsymbol{\omega}$,

$$\frac{\partial \boldsymbol{Rv}}{\partial \boldsymbol{\omega}^T} = -[\boldsymbol{v}]_\times = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}. \tag{2.36}$$

Another way to derive a rotation through a finite angle is called the *exponential twist* (Murray, Li, and Sastry 1994). A rotation by an angle θ is equivalent to k rotations through θ/k. In the limit as $k \to \infty$, we obtain

$$\boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \lim_{k\to\infty} (\boldsymbol{I} + \frac{1}{k}[\theta\hat{\boldsymbol{n}}]_\times)^k = \exp[\boldsymbol{\omega}]_\times. \tag{2.37}$$

If we expand the matrix exponential as a Taylor series (using the identity $[\hat{\boldsymbol{n}}]_\times^{k+2} = -[\hat{\boldsymbol{n}}]_\times^k$, $k > 0$, and again assuming θ is in radians),

$$\begin{aligned} \exp[\boldsymbol{\omega}]_\times &= \boldsymbol{I} + \theta[\hat{\boldsymbol{n}}]_\times + \frac{\theta^2}{2}[\hat{\boldsymbol{n}}]_\times^2 + \frac{\theta^3}{3!}[\hat{\boldsymbol{n}}]_\times^3 + \cdots \\ &= \boldsymbol{I} + (\theta - \frac{\theta^3}{3!} + \cdots)[\hat{\boldsymbol{n}}]_\times + (\frac{\theta^2}{2} - \frac{\theta^3}{4!} + \cdots)[\hat{\boldsymbol{n}}]_\times^2 \\ &= \boldsymbol{I} + \sin\theta[\hat{\boldsymbol{n}}]_\times + (1 - \cos\theta)[\hat{\boldsymbol{n}}]_\times^2, \end{aligned} \tag{2.38}$$

which yields the familiar Rodriguez's formula.

Unit quaternions

The unit quaternion representation is closely related to the angle/axis representation. A unit quaternion is a unit length 4-vector whose components can be written as $\boldsymbol{q} = (q_x, q_y, q_z, q_w)$ or $\boldsymbol{q} = (x, y, z, w)$ for short. Unit quaternions live on the unit sphere $\|\boldsymbol{q}\| = 1$ and *antipodal* (opposite sign) quaternions, $\boldsymbol{q}$ and $-\boldsymbol{q}$, represent the same rotation (Figure 2.6). Other than this ambiguity (dual covering), the unit quaternion representation of a rotation is unique. Furthermore, the representation is *continuous*, i.e., as rotation matrices vary continuously, one can find a continuous quaternion representation, although the path on the quaternion sphere may wrap all the way around before returning to the "origin" $\boldsymbol{q}_o = (0, 0, 0, 1)$. For these and other reasons given below, quaternions are a very popular representation for pose and for pose interpolation in computer graphics (Shoemake 1985).

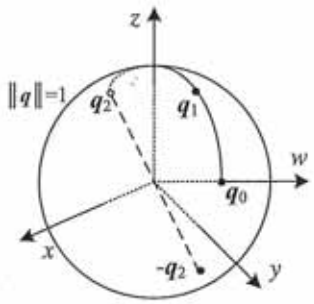


Figure 2.6 Unit quaternions live on the unit sphere $\|q\| = 1$. This figure shows a smooth trajectory through the three quaternions q_0, q_1, and q_2. The *antipodal* point to q_2, namely $-q_2$, represents the same rotation as q_2.

Quaternions can be derived from the axis/angle representation through the formula

$$q = (v, w) = (\sin\frac{\theta}{2}\hat{n}, \cos\frac{\theta}{2}), \tag{2.39}$$

where $\hat{n}$ and θ are the rotation axis and angle. Using the trigonometric identities $\sin\theta = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}$ and $(1-\cos\theta) = 2\sin^2\frac{\theta}{2}$, Rodriguez's formula can be converted to

$$\begin{aligned} R(\hat{n}, \theta) &= I + \sin\theta[\hat{n}]_\times + (1-\cos\theta)[\hat{n}]_\times^2 \\ &= I + 2w[v]_\times + 2[v]_\times^2. \end{aligned} \tag{2.40}$$

This suggests a quick way to rotate a vector v by a quaternion using a series of cross products, scalings, and additions. To obtain a formula for $R(q)$ as a function of (x, y, z, w), recall that

$$[v]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \text{ and } [v]_\times^2 = \begin{bmatrix} -y^2-z^2 & xy & xz \\ xy & -x^2-z^2 & yz \\ xz & yz & -x^2-y^2 \end{bmatrix}.$$

We thus obtain

$$R(q) = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-zw) & 2(xz+yw) \\ 2(xy+zw) & 1-2(x^2+z^2) & 2(yz-xw) \\ 2(xz-yw) & 2(yz+xw) & 1-2(x^2+y^2) \end{bmatrix}. \tag{2.41}$$

The diagonal terms can be made more symmetrical by replacing $1-2(y^2+z^2)$ with $(x^2+w^2-y^2-z^2)$, etc.

The nicest aspect of unit quaternions is that there is a simple algebra for composing rotations expressed as unit quaternions. Given two quaternions $q_0 = (v_0, w_0)$ and $q_1 = (v_1, w_1)$, the *quaternion multiply* operator is defined as

$$q_2 = q_0 q_1 = (v_0 \times v_1 + w_0 v_1 + w_1 v_0,\ w_0 w_1 - v_0 \cdot v_1), \tag{2.42}$$

with the property that $R(q_2) = R(q_0)R(q_1)$. Note that quaternion multiplication is *not* commutative, just as 3D rotations and matrix multiplications are not.

procedure *slerp*(q_0, q_1, α):

1. $q_r = q_1/q_0 = (\boldsymbol{v}_r, w_r)$
2. if $w_r < 0$ then $q_r \leftarrow -q_r$
3. $\theta_r = 2\tan^{-1}(\|\boldsymbol{v}_r\|/w_r)$
4. $\hat{\boldsymbol{n}}_r = \mathcal{N}(\boldsymbol{v}_r) = \boldsymbol{v}_r/\|\boldsymbol{v}_r\|$
5. $\theta_\alpha = \alpha\theta_r$
6. $q_\alpha = (\sin\frac{\theta_\alpha}{2}\hat{\boldsymbol{n}}_r, \cos\frac{\theta_\alpha}{2})$
7. **return** $q_2 = q_\alpha q_0$

Algorithm 2.1 Spherical linear interpolation (slerp). The axis and total angle are first computed from the quaternion ratio. (This computation can be lifted outside an inner loop that generates a set of interpolated position for animation.) An incremental quaternion is then computed and multiplied by the starting rotation quaternion.

Taking the inverse of a quaternion is easy: Just flip the sign of $\boldsymbol{v}$ or w (but not both!). (You can verify this has the desired effect of transposing the $\boldsymbol{R}$ matrix in (2.41).) Thus, we can also define *quaternion division* as

$$q_2 = q_0/q_1 = q_0 q_1^{-1} = (\boldsymbol{v}_0 \times \boldsymbol{v}_1 + w_0\boldsymbol{v}_1 - w_1\boldsymbol{v}_0,\ -w_0w_1 - \boldsymbol{v}_0 \cdot \boldsymbol{v}_1). \tag{2.43}$$

This is useful when the *incremental rotation* between two rotations is desired.

In particular, if we want to determine a rotation that is partway between two given rotations, we can compute the incremental rotation, take a fraction of the angle, and compute the new rotation. This procedure is called *spherical linear interpolation* or *slerp* for short (Shoemake 1985) and is given in Algorithm 2.1. Note that Shoemake presents two formulas other than the one given here. The first exponentiates q_r by alpha before multiplying the original quaternion,

$$q_2 = q_r^\alpha q_0, \tag{2.44}$$

while the second treats the quaternions as 4-vectors on a sphere and uses

$$q_2 = \frac{\sin(1-\alpha)\theta}{\sin\theta} q_0 + \frac{\sin\alpha\theta}{\sin\theta} q_1, \tag{2.45}$$

where $\theta = \cos^{-1}(q_0 \cdot q_1)$ and the dot product is directly between the quaternion 4-vectors. All of these formulas give comparable results, although care should be taken when q_0 and q_1 are close together, which is why I prefer to use an arctangent to establish the rotation angle.

Which rotation representation is better?

The choice of representation for 3D rotations depends partly on the application.

The axis/angle representation is minimal, and hence does not require any additional constraints on the parameters (no need to re-normalize after each update). If the angle is expressed in degrees, it is easier to understand the pose (say, 90° twist around x-axis), and also

easier to express exact rotations. When the angle is in radians, the derivatives of $\boldsymbol{R}$ with respect to ω can easily be computed (2.36).

Quaternions, on the other hand, are better if you want to keep track of a smoothly moving camera, since there are no discontinuities in the representation. It is also easier to interpolate between rotations and to chain rigid transformations (Murray, Li, and Sastry 1994; Bregler and Malik 1998).

My usual preference is to use quaternions, but to update their estimates using an incremental rotation, as described in Section 6.2.2.

2.1.5 3D to 2D projections

Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives are projected onto the image plane. We can do this using a linear 3D to 2D projection matrix. The simplest model is orthography, which requires no division to get the final (inhomogeneous) result. The more commonly used model is perspective, since this more accurately models the behavior of real cameras.

Orthography and para-perspective

An orthographic projection simply drops the z component of the three-dimensional coordinate $\boldsymbol{p}$ to obtain the 2D point $\boldsymbol{x}$. (In this section, we use $\boldsymbol{p}$ to denote 3D points and $\boldsymbol{x}$ to denote 2D points.) This can be written as

$$\boldsymbol{x} = [\boldsymbol{I}_{2\times 2}|\boldsymbol{0}]\,\boldsymbol{p}. \tag{2.46}$$

If we are using homogeneous (projective) coordinates, we can write

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{p}}, \tag{2.47}$$

i.e., we drop the z component but keep the w component. Orthography is an approximate model for long focal length (telephoto) lenses and objects whose depth is *shallow* relative to their distance to the camera (Sawhney and Hanson 1991). It is exact only for *telecentric* lenses (Baker and Nayar 1999, 2001).

In practice, world coordinates (which may measure dimensions in meters) need to be scaled to fit onto an image sensor (physically measured in millimeters, but ultimately measured in pixels). For this reason, *scaled orthography* is actually more commonly used,

$$\boldsymbol{x} = [s\boldsymbol{I}_{2\times 2}|\boldsymbol{0}]\,\boldsymbol{p}. \tag{2.48}$$

This model is equivalent to first projecting the world points onto a local fronto-parallel image plane and then scaling this image using regular perspective projection. The scaling can be the same for all parts of the scene (Figure 2.7b) or it can be different for objects that are being modeled independently (Figure 2.7c). More importantly, the scaling can vary from frame to frame when estimating *structure from motion*, which can better model the scale change that occurs as an object approaches the camera.

Scaled orthography is a popular model for reconstructing the 3D shape of objects far away from the camera, since it greatly simplifies certain computations. For example, *pose* (camera

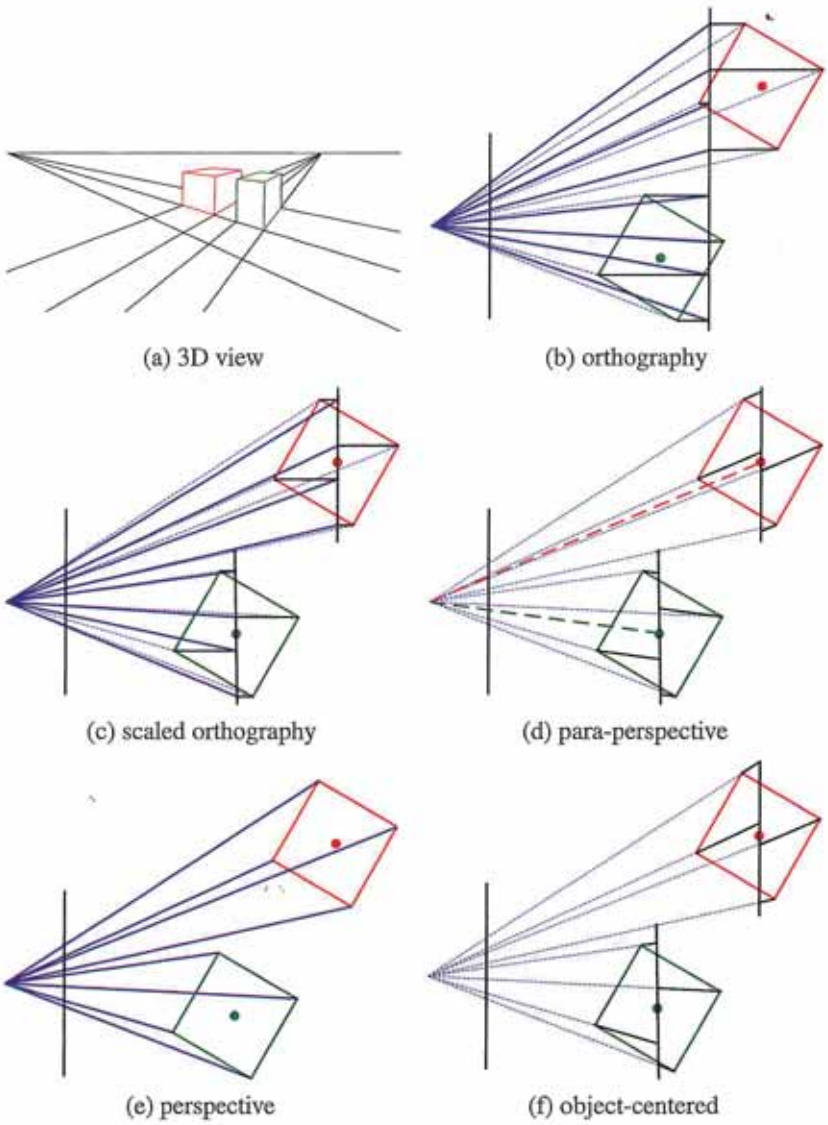


Figure 2.7 Commonly used projection models: (a) 3D view of world, (b) orthography, (c) scaled orthography, (d) para-perspective, (e) perspective, (f) object-centered. Each diagram shows a top-down view of the projection. Note how parallel lines on the ground plane and box sides remain parallel in the non-perspective projections.

orientation) can be estimated using simple least squares (Section 6.2.1). Under orthography, structure and motion can simultaneously be estimated using *factorization* (singular value decomposition), as discussed in Section 7.3 (Tomasi and Kanade 1992).

A closely related projection model is *para-perspective* (Aloimonos 1990; Poelman and Kanade 1997). In this model, object points are again first projected onto a local reference parallel to the image plane. However, rather than being projected orthogonally to this plane, they are projected *parallel* to the line of sight to the object center (Figure 2.7d). This is followed by the usual projection onto the final image plane, which again amounts to a scaling. The combination of these two projections is therefore *affine* and can be written as

$$\bar{\boldsymbol{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{p}}. \tag{2.49}$$

Note how parallel lines in 3D remain parallel after projection in Figure 2.7b–d. Para-perspective provides a more accurate projection model than scaled orthography, without incurring the added complexity of per-pixel perspective division, which invalidates traditional factorization methods (Poelman and Kanade 1997).

Perspective

The most commonly used projection in computer graphics and computer vision is true 3D *perspective* (Figure 2.7e). Here, points are projected onto the image plane by dividing them by their z component. Using inhomogeneous coordinates, this can be written as

$$\bar{\boldsymbol{x}} = \mathcal{P}_z(\boldsymbol{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}. \tag{2.50}$$

In homogeneous coordinates, the projection has a simple linear form,

$$\bar{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\boldsymbol{p}}, \tag{2.51}$$

i.e., we drop the w component of $\boldsymbol{p}$. Thus, after projection, it is not possible to recover the *distance* of the 3D point from the image, which makes sense for a 2D imaging sensor.

A form often seen in computer graphics systems is a two-step projection that first projects 3D coordinates into *normalized device coordinates* in the range $(x, y, z) \in [-1, -1] \times [-1, 1] \times [0, 1]$, and then rescales these coordinates to integer pixel coordinates using a *viewport* transformation (Watt 1995; OpenGL-ARB 1997). The (initial) perspective projection is then represented using a 4×4 matrix

$$\bar{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{\text{far}}/z_{\text{range}} & z_{\text{near}} z_{\text{far}}/z_{\text{range}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\boldsymbol{p}}, \tag{2.52}$$

where z_{near} and z_{far} are the near and far z *clipping planes* and $z_{\text{range}} = z_{\text{far}} - z_{\text{near}}$. Note that the first two rows are actually scaled by the focal length and the aspect ratio so that

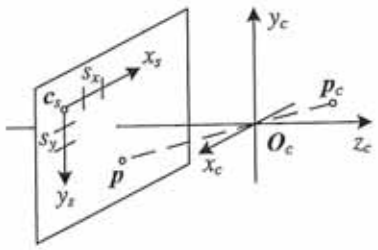


Figure 2.8 Projection of a 3D camera-centered point p_c onto the sensor planes at location p. O_c is the camera center (nodal point), c_s is the 3D origin of the sensor plane coordinate system, and s_x and s_y are the pixel spacings.

visible rays are mapped to $(x, y, z) \in [-1, -1]^2$. The reason for keeping the third row, rather than dropping it, is that visibility operations, such as *z-buffering*, require a depth for every graphical element that is being rendered.

If we set $z_{\text{near}} = 1$, $z_{\text{far}} \to \infty$, and switch the sign of the third row, the third element of the normalized screen vector becomes the inverse depth, i.e., the *disparity* (Okutomi and Kanade 1993). This can be quite convenient in many cases since, for cameras moving around outdoors, the inverse depth to the camera is often a more well-conditioned parameterization than direct 3D distance.

While a regular 2D image sensor has no way of measuring distance to a surface point, *range sensors* (Section 12.2) and stereo matching algorithms (Chapter 11) can compute such values. It is then convenient to be able to map from a sensor-based depth or disparity value d directly back to a 3D location using the inverse of a 4×4 matrix (Section 2.1.5). We can do this if we represent perspective projection using a full-rank 4×4 matrix, as in (2.64).

Camera intrinsics

Once we have projected a 3D point through an ideal pinhole using a projection matrix, we must still transform the resulting coordinates according to the pixel sensor spacing and the relative position of the sensor plane to the origin. Figure 2.8 shows an illustration of the geometry involved. In this section, we first present a mapping from 2D pixel coordinates to 3D rays using a sensor homography $\boldsymbol{M}_s$, since this is easier to explain in terms of physically measurable quantities. We then relate these quantities to the more commonly used camera intrinsic matrix $\boldsymbol{K}$, which is used to map 3D camera-centered points $\boldsymbol{p}_c$ to 2D pixel coordinates $\tilde{\boldsymbol{x}}_s$.

Image sensors return pixel values indexed by integer *pixel coordinates* (x_s, y_s), often with the coordinates starting at the upper-left corner of the image and moving down and to the right. (This convention is not obeyed by all imaging libraries, but the adjustment for other coordinate systems is straightforward.) To map pixel centers to 3D coordinates, we first scale the (x_s, y_s) values by the pixel spacings (s_x, s_y) (sometimes expressed in microns for solid-state sensors) and then describe the orientation of the sensor array relative to the camera projection center $\boldsymbol{O}_c$ with an origin $\boldsymbol{c}_s$ and a 3D rotation $\boldsymbol{R}_s$ (Figure 2.8).

The combined 2D to 3D projection can then be written as

$$p = \left[\begin{array}{c|c} R_s & c_s \end{array} \right] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = M_s \bar{x}_s. \tag{2.53}$$

The first two columns of the 3×3 matrix M_s are the 3D vectors corresponding to unit steps in the image pixel array along the x_s and y_s directions, while the third column is the 3D image array origin c_s.

The matrix M_s is parameterized by eight unknowns: the three parameters describing the rotation R_s, the three parameters describing the translation c_s, and the two scale factors (s_x, s_y). Note that we ignore here the possibility of *skew* between the two axes on the image plane, since solid-state manufacturing techniques render this negligible. In practice, unless we have accurate external knowledge of the sensor spacing or sensor orientation, there are only seven degrees of freedom, since the distance of the sensor from the origin cannot be teased apart from the sensor spacing, based on external image measurement alone.

However, estimating a camera model M_s with the required seven degrees of freedom (i.e., where the first two columns are orthogonal after an appropriate re-scaling) is impractical, so most practitioners assume a general 3×3 homogeneous matrix form.

The relationship between the 3D pixel center p and the 3D camera-centered point p_c is given by an unknown scaling s, $p = sp_c$. We can therefore write the complete projection between p_c and a homogeneous version of the pixel address $\bar{x}_s$ as

$$\bar{x}_s = \alpha M_s^{-1} p_c = K p_c. \tag{2.54}$$

The 3×3 matrix K is called the *calibration matrix* and describes the camera *intrinsics* (as opposed to the camera's orientation in space, which are called the *extrinsics*).

From the above discussion, we see that K has seven degrees of freedom in theory and eight degrees of freedom (the full dimensionality of a 3×3 homogeneous matrix) in practice. Why, then, do most textbooks on 3D computer vision and multi-view geometry (Faugeras 1993; Hartley and Zisserman 2004; Faugeras and Luong 2001) treat K as an upper-triangular matrix with five degrees of freedom?

While this is usually not made explicit in these books, it is because we cannot recover the full K matrix based on external measurement alone. When calibrating a camera (Chapter 6) based on external 3D points or other measurements (Tsai 1987), we end up estimating the intrinsic (K) and extrinsic (R, t) camera parameters simultaneously using a series of measurements,

$$\bar{x}_s = K \left[\begin{array}{c|c} R & t \end{array} \right] p_w = P p_w, \tag{2.55}$$

where p_w are known 3D world coordinates and

$$P = K[R|t] \tag{2.56}$$

is known as the *camera matrix*. Inspecting this equation, we see that we can post-multiply K by R_1 and pre-multiply $[R|t]$ by R_1^T, and still end up with a valid calibration. Thus, it is impossible based on image measurements alone to know the true orientation of the sensor and the true camera intrinsics.

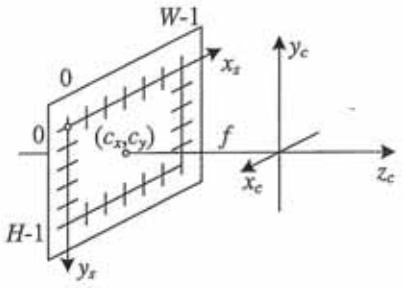


Figure 2.9 Simplified camera intrinsics showing the focal length f and the optical center (c_x, c_y). The image width and height are W and H.

The choice of an upper-triangular form for $\boldsymbol{K}$ seems to be conventional. Given a full 3×4 camera matrix $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]$, we can compute an upper-triangular $\boldsymbol{K}$ matrix using QR factorization (Golub and Van Loan 1996). (Note the unfortunate clash of terminologies: In matrix algebra textbooks, $\boldsymbol{R}$ represents an upper-triangular (right of the diagonal) matrix; in computer vision, $\boldsymbol{R}$ is an orthogonal rotation.)

There are several ways to write the upper-triangular form of $\boldsymbol{K}$. One possibility is

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.57}$$

which uses independent *focal lengths* f_x and f_y for the sensor x and y dimensions. The entry s encodes any possible *skew* between the sensor axes due to the sensor not being mounted perpendicular to the optical axis and (c_x, c_y) denotes the *optical center* expressed in pixel coordinates. Another possibility is

$$\boldsymbol{K} = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.58}$$

where the *aspect ratio* a has been made explicit and a common focal length f is used.

In practice, for many applications an even simpler form can be obtained by setting $a = 1$ and $s = 0$,

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.59}$$

Often, setting the origin at roughly the center of the image, e.g., $(c_x, c_y) = (W/2, H/2)$, where W and H are the image height and width, can result in a perfectly usable camera model with a single unknown, i.e., the focal length f.

Figure 2.9 shows how these quantities can be visualized as part of a simplified imaging model. Note that now we have placed the image plane *in front* of the nodal point (projection center of the lens). The sense of the y axis has also been flipped to get a coordinate system compatible with the way that most imaging libraries treat the vertical (row) coordinate. Certain graphics libraries, such as Direct3D, use a left-handed coordinate system, which can lead to some confusion.

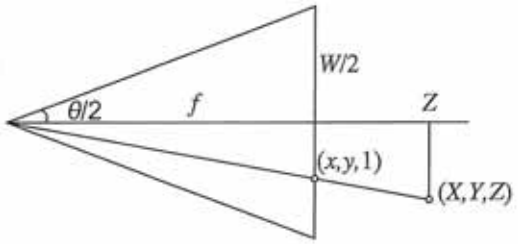


Figure 2.10 Central projection, showing the relationship between the 3D and 2D coordinates, $\boldsymbol{p}$ and $\boldsymbol{x}$, as well as the relationship between the focal length f, image width W, and the field of view θ.

A note on focal lengths

The issue of how to express focal lengths is one that often causes confusion in implementing computer vision algorithms and discussing their results. This is because the focal length depends on the units used to measure pixels.

If we number pixel coordinates using integer values, say $[0, W) \times [0, H)$, the focal length f and camera center (c_x, c_y) in (2.59) can be expressed as pixel values. How do these quantities relate to the more familiar focal lengths used by photographers?

Figure 2.10 illustrates the relationship between the focal length f, the sensor width W, and the field of view θ, which obey the formula

$$\tan\frac{\theta}{2} = \frac{W}{2f} \quad \text{or} \quad f = \frac{W}{2}\left[\tan\frac{\theta}{2}\right]^{-1}. \tag{2.60}$$

For conventional film cameras, $W = 35\text{mm}$, and hence f is also expressed in millimeters. Since we work with digital images, it is more convenient to express W in pixels so that the focal length f can be used directly in the calibration matrix $\boldsymbol{K}$ as in (2.59).

Another possibility is to scale the pixel coordinates so that they go from $[-1, 1)$ along the longer image dimension and $[-a^{-1}, a^{-1})$ along the shorter axis, where $a \geq 1$ is the *image aspect ratio* (as opposed to the *sensor cell aspect ratio* introduced earlier). This can be accomplished using *modified normalized device coordinates*,

$$x'_s = (2x_s - W)/S \text{ and } y'_s = (2y_s - H)/S, \quad \text{where} \quad S = \max(W, H). \tag{2.61}$$

This has the advantage that the focal length f and optical center (c_x, c_y) become independent of the image resolution, which can be useful when using multi-resolution, image-processing algorithms, such as image pyramids (Section 3.5).[2] The use of S instead of W also makes the focal length the same for landscape (horizontal) and portrait (vertical) pictures, as is the case in 35mm photography. (In some computer graphics textbooks and systems, normalized device coordinates go from $[-1, 1] \times [-1, 1]$, which requires the use of two different focal lengths to describe the camera intrinsics (Watt 1995; OpenGL-ARB 1997).) Setting $S = W = 2$ in (2.60), we obtain the simpler (unitless) relationship

$$f^{-1} = \tan\frac{\theta}{2}. \tag{2.62}$$

[2] To make the conversion truly accurate after a downsampling step in a pyramid, floating point values of W and H would have to be maintained since they can become non-integral if they are ever odd at a larger resolution in the pyramid.

The conversion between the various focal length representations is straightforward, e.g., to go from a unitless f to one expressed in pixels, multiply by $W/2$, while to convert from an f expressed in pixels to the equivalent 35mm focal length, multiply by $35/W$.

Camera matrix

Now that we have shown how to parameterize the calibration matrix $\boldsymbol{K}$, we can put the camera intrinsics and extrinsics together to obtain a single 3×4 *camera matrix*

$$\boldsymbol{P} = \boldsymbol{K}\left[\begin{array}{c|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]. \tag{2.63}$$

It is sometimes preferable to use an invertible 4×4 matrix, which can be obtained by not dropping the last row in the $\boldsymbol{P}$ matrix,

$$\tilde{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \tilde{\boldsymbol{K}}\boldsymbol{E}, \tag{2.64}$$

where $\boldsymbol{E}$ is a 3D rigid-body (Euclidean) transformation and $\tilde{\boldsymbol{K}}$ is the full-rank calibration matrix. The 4×4 camera matrix $\tilde{\boldsymbol{P}}$ can be used to map directly from 3D world coordinates $\bar{\boldsymbol{p}}_w = (x_w, y_w, z_w, 1)$ to screen coordinates (plus disparity), $\boldsymbol{x}_s = (x_s, y_s, 1, d)$,

$$\boldsymbol{x}_s \sim \tilde{\boldsymbol{P}}\bar{\boldsymbol{p}}_w, \tag{2.65}$$

where $\sim$ indicates equality up to scale. Note that after multiplication by $\tilde{\boldsymbol{P}}$, the vector is divided by the *third* element of the vector to obtain the normalized form $\boldsymbol{x}_s = (x_s, y_s, 1, d)$.

Plane plus parallax (projective depth)

In general, when using the 4×4 matrix $\tilde{\boldsymbol{P}}$, we have the freedom to remap the last row to whatever suits our purpose (rather than just being the "standard" interpretation of disparity as inverse depth). Let us re-write the last row of $\tilde{\boldsymbol{P}}$ as $\boldsymbol{p}_3 = s_3[\hat{\boldsymbol{n}}_0|c_0]$, where $\|\hat{\boldsymbol{n}}_0\| = 1$. We then have the equation

$$d = \frac{s_3}{z}(\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p}_w + c_0), \tag{2.66}$$

where $z = \boldsymbol{p}_2 \cdot \bar{\boldsymbol{p}}_w = \boldsymbol{r}_z \cdot (\boldsymbol{p}_w - \boldsymbol{c})$ is the distance of $\boldsymbol{p}_w$ from the camera center C (2.25) along the optical axis Z (Figure 2.11). Thus, we can interpret d as the *projective disparity* or *projective depth* of a 3D scene point $\boldsymbol{p}_w$ from the *reference plane* $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p}_w + c_0 = 0$ (Szeliski and Coughlan 1997; Szeliski and Golland 1999; Shade, Gortler, He *et al.* 1998; Baker, Szeliski, and Anandan 1998). (The projective depth is also sometimes called *parallax* in reconstruction algorithms that use the term *plane plus parallax* (Kumar, Anandan, and Hanna 1994; Sawhney 1994).) Setting $\hat{\boldsymbol{n}}_0 = \boldsymbol{0}$ and $c_0 = 1$, i.e., putting the reference plane at infinity, results in the more standard $d = 1/z$ version of disparity (Okutomi and Kanade 1993).

Another way to see this is to invert the $\tilde{\boldsymbol{P}}$ matrix so that we can map pixels plus disparity directly back to 3D points,

$$\tilde{\boldsymbol{p}}_w = \tilde{\boldsymbol{P}}^{-1}\boldsymbol{x}_s. \tag{2.67}$$

In general, we can choose $\tilde{\boldsymbol{P}}$ to have whatever form is convenient, i.e., to sample space using an arbitrary projection. This can come in particularly handy when setting up multi-view

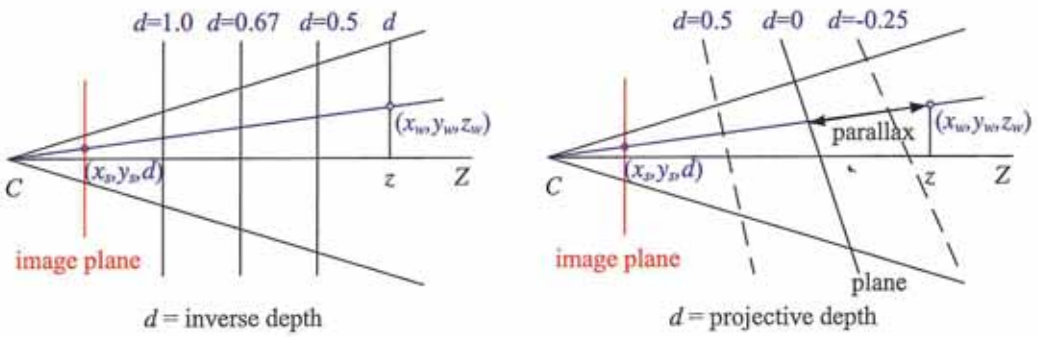


Figure 2.11 Regular disparity (inverse depth) and projective depth (parallax from a reference plane).

stereo reconstruction algorithms, since it allows us to sweep a series of planes (Section 11.1.2) through space with a variable (projective) sampling that best matches the sensed image motions (Collins 1996; Szeliski and Golland 1999; Saito and Kanade 1999).

Mapping from one camera to another

What happens when we take two images of a 3D scene from different camera positions or orientations (Figure 2.12a)? Using the full rank 4×4 camera matrix $\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{K}}\boldsymbol{E}$ from (2.64), we can write the projection from world to screen coordinates as

$$\tilde{\boldsymbol{x}}_0 \sim \tilde{\boldsymbol{K}}_0\boldsymbol{E}_0\boldsymbol{p} = \tilde{\boldsymbol{P}}_0\boldsymbol{p}. \tag{2.68}$$

Assuming that we know the z-buffer or disparity value d_0 for a pixel in one image, we can compute the 3D point location $\boldsymbol{p}$ using

$$\boldsymbol{p} \sim \boldsymbol{E}_0^{-1}\tilde{\boldsymbol{K}}_0^{-1}\tilde{\boldsymbol{x}}_0 \tag{2.69}$$

and then project it into another image yielding

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{K}}_1\boldsymbol{E}_1\boldsymbol{p} = \tilde{\boldsymbol{K}}_1\boldsymbol{E}_1\boldsymbol{E}_0^{-1}\tilde{\boldsymbol{K}}_0^{-1}\tilde{\boldsymbol{x}}_0 = \tilde{\boldsymbol{P}}_1\tilde{\boldsymbol{P}}_0^{-1}\tilde{\boldsymbol{x}}_0 = \boldsymbol{M}_{10}\tilde{\boldsymbol{x}}_0. \tag{2.70}$$

Unfortunately, we do not usually have access to the depth coordinates of pixels in a regular photographic image. However, for a *planar scene*, as discussed above in (2.66), we can replace the last row of $\boldsymbol{P}_0$ in (2.64) with a general *plane equation*, $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + c_0$ that maps points on the plane to $d_0 = 0$ values (Figure 2.12b). Thus, if we set $d_0 = 0$, we can ignore the last column of $\boldsymbol{M}_{10}$ in (2.70) and also its last row, since we do not care about the final z-buffer depth. The mapping equation (2.70) thus reduces to

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{H}}_{10}\tilde{\boldsymbol{x}}_0, \tag{2.71}$$

where $\tilde{\boldsymbol{H}}_{10}$ is a general 3×3 homography matrix and $\tilde{\boldsymbol{x}}_1$ and $\tilde{\boldsymbol{x}}_0$ are now 2D homogeneous coordinates (i.e., 3-vectors) (Szeliski 1996).This justifies the use of the 8-parameter homography as a general alignment model for mosaics of planar scenes (Mann and Picard 1994; Szeliski 1996).

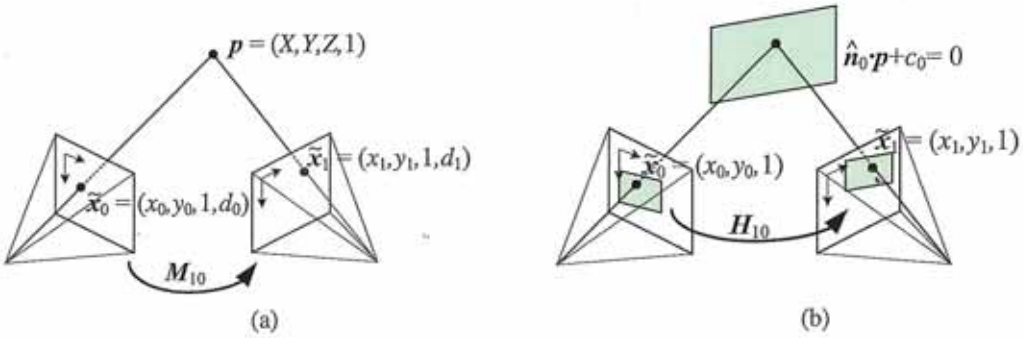


Figure 2.12 A point is projected into two images: (a) relationship between the 3D point coordinate $(X, Y, Z, 1)$ and the 2D projected point $(x, y, 1, d)$; (b) planar homography induced by points all lying on a common plane $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + c_0 = 0$.

The other special case where we do not need to know depth to perform inter-camera mapping is when the camera is undergoing pure rotation (Section 9.1.3), i.e., when $\boldsymbol{t}_0 = \boldsymbol{t}_1$. In this case, we can write

$$\tilde{\boldsymbol{x}}_1 \sim \boldsymbol{K}_1 \boldsymbol{R}_1 \boldsymbol{R}_0^{-1} \boldsymbol{K}_0^{-1} \tilde{\boldsymbol{x}}_0 = \boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1} \tilde{\boldsymbol{x}}_0, \tag{2.72}$$

which again can be represented with a 3×3 homography. If we assume that the calibration matrices have known aspect ratios and centers of projection (2.59), this homography can be parameterized by the rotation amount and the two unknown focal lengths. This particular formulation is commonly used in image-stitching applications (Section 9.1.3).

Object-centered projection

When working with long focal length lenses, it often becomes difficult to reliably estimate the focal length from image measurements alone. This is because the focal length and the distance to the object are highly correlated and it becomes difficult to tease these two effects apart. For example, the change in scale of an object viewed through a zoom telephoto lens can either be due to a zoom change or a motion towards the user. (This effect was put to dramatic use in some of Alfred Hitchcock's film *Vertigo*, where the simultaneous change of zoom and camera motion produces a disquieting effect.)

This ambiguity becomes clearer if we write out the projection equation corresponding to the simple calibration matrix $\boldsymbol{K}$ (2.59),

$$x_s = f\frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z} + c_x \tag{2.73}$$

$$y_s = f\frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z} + c_y, \tag{2.74}$$

where $\boldsymbol{r}_x$, $\boldsymbol{r}_y$, and $\boldsymbol{r}_z$ are the three rows of $\boldsymbol{R}$. If the distance to the object center $t_z \gg \|\boldsymbol{p}\|$ (the size of the object), the denominator is approximately t_z and the overall scale of the projected object depends on the ratio of f to t_z. It therefore becomes difficult to disentangle these two quantities.

To see this more clearly, let $\eta_z = t_z^{-1}$ and $s = \eta_z f$. We can then re-write the above equations as

$$x_s = s\frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{1 + \eta_z \boldsymbol{r}_z \cdot \boldsymbol{p}} + c_x \tag{2.75}$$

$$y_s = s\frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{1 + \eta_z \boldsymbol{r}_z \cdot \boldsymbol{p}} + c_y \tag{2.76}$$

(Szeliski and Kang 1994; Pighin, Hecker, Lischinski *et al.* 1998). The scale of the projection s can be reliably estimated if we are looking at a known object (i.e., the 3D coordinates $\boldsymbol{p}$ are known). The inverse distance η_z is now mostly decoupled from the estimates of s and can be estimated from the amount of *foreshortening* as the object rotates. Furthermore, as the lens becomes longer, i.e., the projection model becomes orthographic, there is no need to replace a perspective imaging model with an orthographic one, since the same equation can be used, with $\eta_z \to 0$ (as opposed to f and t_z both going to infinity). This allows us to form a natural link between orthographic reconstruction techniques such as factorization and their projective/perspective counterparts (Section 7.3).

2.1.6 Lens distortions

The above imaging models all assume that cameras obey a *linear* projection model where straight lines in the world result in straight lines in the image. (This follows as a natural consequence of linear matrix operations being applied to homogeneous coordinates.) Unfortunately, many wide-angle lenses have noticeable *radial distortion*, which manifests itself as a visible curvature in the projection of straight lines. (See Section 2.2.3 for a more detailed discussion of lens optics, including chromatic aberration.) Unless this distortion is taken into account, it becomes impossible to create highly accurate photorealistic reconstructions. For example, image mosaics constructed without taking radial distortion into account will often exhibit blurring due to the mis-registration of corresponding features before pixel blending (Chapter 9).

Fortunately, compensating for radial distortion is not that difficult in practice. For most lenses, a simple quartic model of distortion can produce good results. Let (x_c, y_c) be the pixel coordinates obtained *after* perspective division but *before* scaling by focal length f and shifting by the optical center (c_x, c_y), i.e.,

$$\begin{aligned} x_c &= \frac{\boldsymbol{r}_x \cdot \boldsymbol{p} + t_x}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z} \\ y_c &= \frac{\boldsymbol{r}_y \cdot \boldsymbol{p} + t_y}{\boldsymbol{r}_z \cdot \boldsymbol{p} + t_z}. \end{aligned} \tag{2.77}$$

The radial distortion model says that coordinates in the observed images are displaced away (*barrel* distortion) or towards (*pincushion* distortion) the image center by an amount proportional to their radial distance (Figure 2.13a–b).[3] The simplest radial distortion models use low-order polynomials, e.g.,

$$\begin{aligned} \hat{x}_c &= x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \\ \hat{y}_c &= y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4), \end{aligned} \tag{2.78}$$

[3] Anamorphic lenses, which are widely used in feature film production, do not follow this radial distortion model. Instead, they can be thought of, to a first approximation, as inducing different vertical and horizontal scalings, i.e., non-square pixels.

(a) (b) (c)

Figure 2.13 Radial lens distortions: (a) barrel, (b) pincushion, and (c) fisheye. The fisheye image spans almost 180° from side-to-side.

where $r_c^2 = x_c^2 + y_c^2$ and κ_1 and κ_2 are called the *radial distortion parameters*.[4] After the radial distortion step, the final pixel coordinates can be computed using

$$\begin{aligned} x_s &= f x_c' + c_x \\ y_s &= f y_c' + c_y. \end{aligned} \tag{2.79}$$

A variety of techniques can be used to estimate the radial distortion parameters for a given lens, as discussed in Section 6.3.5.

Sometimes the above simplified model does not model the true distortions produced by complex lenses accurately enough (especially at very wide angles). A more complete analytic model also includes *tangential distortions* and *decentering distortions* (Slama 1980), but these distortions are not covered in this book.

Fisheye lenses (Figure 2.13c) require a model that differs from traditional polynomial models of radial distortion. Fisheye lenses behave, to a first approximation, as *equi-distance* projectors of angles away from the optical axis (Xiong and Turkowski 1997), which is the same as the *polar projection* described by Equations (9.22–9.24). Xiong and Turkowski (1997) describe how this model can be extended with the addition of an extra quadratic correction in ϕ and how the unknown parameters (center of projection, scaling factor s, etc.) can be estimated from a set of overlapping fisheye images using a direct (intensity-based) non-linear minimization algorithm.

For even larger, less regular distortions, a parametric distortion model using splines may be necessary (Goshtasby 1989). If the lens does not have a single center of projection, it may become necessary to model the 3D *line* (as opposed to *direction*) corresponding to each pixel separately (Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée, Sautot *et al.* 1992; Grossberg and Nayar 2001; Sturm and Ramalingam 2004; Tardif, Sturm, Trudeau *et al.* 2009). Some of these techniques are described in more detail in Section 6.3.5, which discusses how to calibrate lens distortions.

[4] Sometimes the relationship between x_c and $\hat{x}_c$ is expressed the other way around, i.e., $x_c = \hat{x}_c(1 + \kappa_1 \hat{r}_c^2 + \kappa_2 \hat{r}_c^4)$. This is convenient if we map image pixels into (warped) rays by dividing through by f. We can then undistort the rays and have true 3D rays in space.

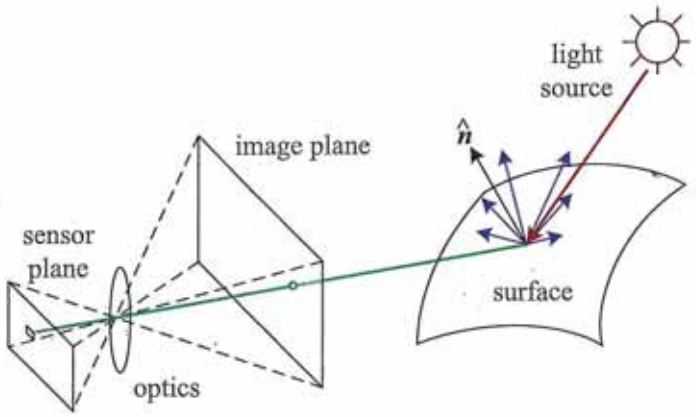


Figure 2.14 A simplified model of photometric image formation. Light is emitted by one or more light sources and is then reflected from an object's surface. A portion of this light is directed towards the camera. This simplified model ignores multiple reflections, which often occur in real-world scenes.

There is one subtle issue associated with the simple radial distortion model that is often glossed over. We have introduced a non-linearity between the perspective projection and final sensor array projection steps. Therefore, we cannot, in general, post-multiply an arbitrary 3×3 matrix $\boldsymbol{K}$ with a rotation to put it into upper-triangular form and absorb this into the global rotation. However, this situation is not as bad as it may at first appear. For many applications, keeping the simplified diagonal form of (2.59) is still an adequate model. Furthermore, if we correct radial and other distortions to an accuracy where straight lines are preserved, we have essentially converted the sensor back into a linear imager and the previous decomposition still applies.

2.2 Photometric image formation

In modeling the image formation process, we have described how 3D geometric features in the world are projected into 2D features in an image. However, images are not composed of 2D features. Instead, they are made up of discrete color or intensity values. Where do these values come from? How do they relate to the lighting in the environment, surface properties and geometry, camera optics, and sensor properties (Figure 2.14)? In this section, we develop a set of models to describe these interactions and formulate a generative process of image formation. A more detailed treatment of these topics can be found in other textbooks on computer graphics and image synthesis (Glassner 1995; Weyrich, Lawrence, Lensch *et al.* 2008; Foley, van Dam, Feiner *et al.* 1995; Watt 1995; Cohen and Wallace 1993; Sillion and Puech 1994).

2.2.1 Lighting

Images cannot exist without light. To produce an image, the scene must be illuminated with one or more light sources. (Certain modalities such as fluorescent microscopy and X-ray

tomography do not fit this model, but we do not deal with them in this book.) Light sources can generally be divided into point and area light sources.

A point light source originates at a single location in space (e.g., a small light bulb), potentially at infinity (e.g., the sun). (Note that for some applications such as modeling soft shadows (*penumbras*), the sun may have to be treated as an area light source.) In addition to its location, a point light source has an intensity and a color spectrum, i.e., a distribution over wavelengths $L(\lambda)$. The intensity of a light source falls off with the square of the distance between the source and the object being lit, because the same light is being spread over a larger (spherical) area. A light source may also have a directional falloff (dependence), but we ignore this in our simplified model.

Area light sources are more complicated. A simple area light source such as a fluorescent ceiling light fixture with a diffuser can be modeled as a finite rectangular area emitting light equally in all directions (Cohen and Wallace 1993; Sillion and Puech 1994; Glassner 1995). When the distribution is strongly directional, a four-dimensional lightfield can be used instead (Ashdown 1993).

A more complex light distribution that approximates, say, the incident illumination on an object sitting in an outdoor courtyard, can often be represented using an *environment map* (Greene 1986) (originally called a *reflection map* (Blinn and Newell 1976)). This representation maps incident light directions $\hat{v}$ to color values (or wavelengths, λ),

$$L(\hat{v}; \lambda), \tag{2.80}$$

and is equivalent to assuming that all light sources are at infinity. Environment maps can be represented as a collection of cubical faces (Greene 1986), as a single longitude–latitude map (Blinn and Newell 1976), or as the image of a reflecting sphere (Watt 1995). A convenient way to get a rough model of a real-world environment map is to take an image of a reflective mirrored sphere and to unwrap this image onto the desired environment map (Debevec 1998). Watt (1995) gives a nice discussion of environment mapping, including the formulas needed to map directions to pixels for the three most commonly used representations.

2.2.2 Reflectance and shading

When light hits an object's surface, it is scattered and reflected (Figure 2.15a). Many different models have been developed to describe this interaction. In this section, we first describe the most general form, the bidirectional reflectance distribution function, and then look at some more specialized models, including the diffuse, specular, and Phong shading models. We also discuss how these models can be used to compute the *global illumination* corresponding to a scene.

The Bidirectional Reflectance Distribution Function (BRDF)

The most general model of light scattering is the *bidirectional reflectance distribution function* (BRDF).[5] Relative to some local coordinate frame on the surface, the BRDF is a four-dimensional function that describes how much of each wavelength arriving at an *incident*

[5] Actually, even more general models of light transport exist, including some that model spatial variation along the surface, sub-surface scattering, and atmospheric effects—see Section 12.7.1—(Dorsey, Rushmeier, and Sillion 2007; Weyrich, Lawrence, Lensch *et al.* 2008).

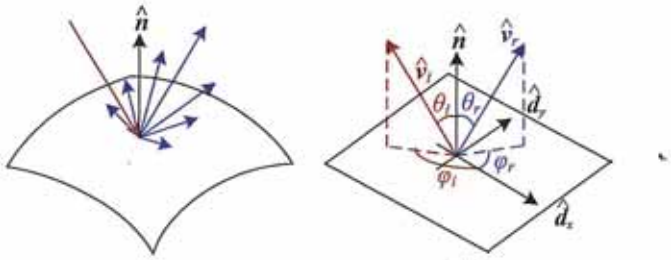


Figure 2.15 (a) Light scatters when it hits a surface. (b) The bidirectional reflectance distribution function (BRDF) $f(\theta_i, \phi_i, \theta_r, \phi_r)$ is parameterized by the angles that the incident, $\hat{\boldsymbol{v}}_i$, and reflected, $\hat{\boldsymbol{v}}_r$, light ray directions make with the local surface coordinate frame $(\hat{\boldsymbol{d}}_x, \hat{\boldsymbol{d}}_y, \hat{\boldsymbol{n}})$.

direction $\hat{\boldsymbol{v}}_i$ is emitted in a *reflected* direction $\hat{\boldsymbol{v}}_r$ (Figure 2.15b). The function can be written in terms of the angles of the incident and reflected directions relative to the surface frame as

$$f_r(\theta_i, \phi_i, \theta_r, \phi_r; \lambda). \tag{2.81}$$

The BRDF is *reciprocal*, i.e., because of the physics of light transport, you can interchange the roles of $\hat{\boldsymbol{v}}_i$ and $\hat{\boldsymbol{v}}_r$ and still get the same answer (this is sometimes called *Helmholtz reciprocity*).

Most surfaces are *isotropic*, i.e., there are no preferred directions on the surface as far as light transport is concerned. (The exceptions are *anisotropic* surfaces such as brushed (scratched) aluminum, where the reflectance depends on the light orientation relative to the direction of the scratches.) For an isotropic material, we can simplify the BRDF to

$$f_r(\theta_i, \theta_r, |\phi_r - \phi_i|; \lambda) \text{ or } f_r(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{v}}_r, \hat{\boldsymbol{n}}; \lambda), \tag{2.82}$$

since the quantities θ_i, θ_r and $\phi_r - \phi_i$ can be computed from the directions $\hat{\boldsymbol{v}}_i$, $\hat{\boldsymbol{v}}_r$, and $\hat{\boldsymbol{n}}$.

To calculate the amount of light exiting a surface point $\boldsymbol{p}$ in a direction $\hat{\boldsymbol{v}}_r$ under a given lighting condition, we integrate the product of the incoming light $L_i(\hat{\boldsymbol{v}}_i; \lambda)$ with the BRDF (some authors call this step a *convolution*). Taking into account the *foreshortening* factor $\cos^+ \theta_i$, we obtain

$$L_r(\hat{\boldsymbol{v}}_r; \lambda) = \int L_i(\hat{\boldsymbol{v}}_i; \lambda) f_r(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{v}}_r, \hat{\boldsymbol{n}}; \lambda) \cos^+ \theta_i \, d\hat{\boldsymbol{v}}_i, \tag{2.83}$$

where

$$\cos^+ \theta_i = \max(0, \cos \theta_i). \tag{2.84}$$

If the light sources are discrete (a finite number of point light sources), we can replace the integral with a summation,

$$L_r(\hat{\boldsymbol{v}}_r; \lambda) = \sum_i L_i(\lambda) f_r(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{v}}_r, \hat{\boldsymbol{n}}; \lambda) \cos^+ \theta_i. \tag{2.85}$$

BRDFs for a given surface can be obtained through physical modeling (Torrance and Sparrow 1967; Cook and Torrance 1982; Glassner 1995), heuristic modeling (Phong 1975), or

Figure 2.16 This close-up of a statue shows both diffuse (smooth shading) and specular (shiny highlight) reflection, as well as darkening in the grooves and creases due to reduced light visibility and interreflections. (Photo courtesy of the Caltech Vision Lab, http://www.vision.caltech.edu/archive.html.)

through empirical observation (Ward 1992; Westin, Arvo, and Torrance 1992; Dana, van Ginneken, Nayar *et al.* 1999; Dorsey, Rushmeier, and Sillion 2007; Weyrich, Lawrence, Lensch *et al.* 2008).[6] Typical BRDFs can often be split into their *diffuse* and *specular* components, as described below.

Diffuse reflection

The diffuse component (also known as *Lambertian* or *matte* reflection) scatters light uniformly in all directions and is the phenomenon we most normally associate with *shading*, e.g., the smooth (non-shiny) variation of intensity with surface normal that is seen when observing a statue (Figure 2.16). Diffuse reflection also often imparts a strong *body color* to the light since it is caused by selective absorption and re-emission of light inside the object's material (Shafer 1985; Glassner 1995).

While light is scattered uniformly in all directions, i.e., the BRDF is constant,

$$f_d(\hat{\boldsymbol{v}}_i, \hat{\boldsymbol{v}}_r, \hat{\boldsymbol{n}}; \lambda) = f_d(\lambda), \tag{2.86}$$

the amount of light depends on the angle between the incident light direction and the surface normal θ_i. This is because the surface area exposed to a given amount of light becomes larger at oblique angles, becoming completely self-shadowed as the outgoing surface normal points away from the light (Figure 2.17a). (Think about how you orient yourself towards the sun or fireplace to get maximum warmth and how a flashlight projected obliquely against a wall is less bright than one pointing directly at it.) The *shading equation* for diffuse reflection can thus be written as

$$L_d(\hat{\boldsymbol{v}}_r; \lambda) = \sum_i L_i(\lambda) f_d(\lambda) \cos^+ \theta_i = \sum_i L_i(\lambda) f_d(\lambda) [\hat{\boldsymbol{v}}_i \cdot \hat{\boldsymbol{n}}]^+, \tag{2.87}$$

[6] See http://www1.cs.columbia.edu/CAVE/software/curet/ for a database of some empirically sampled BRDFs.

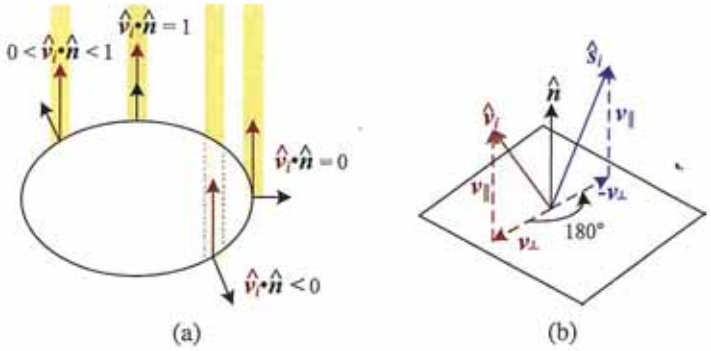


Figure 2.17 (a) The diminution of returned light caused by *foreshortening* depends on $\hat{\boldsymbol{v}}_i \cdot \hat{\boldsymbol{n}}$, the cosine of the angle between the incident light direction $\hat{\boldsymbol{v}}_i$ and the surface normal $\hat{\boldsymbol{n}}$. (b) Mirror (specular) reflection: The incident light ray direction $\hat{\boldsymbol{v}}_i$ is reflected onto the specular direction $\hat{\boldsymbol{s}}_i$ around the surface normal $\hat{\boldsymbol{n}}$.

where

$$[\hat{\boldsymbol{v}}_i \cdot \hat{\boldsymbol{n}}]^+ = \max(0, \hat{\boldsymbol{v}}_i \cdot \hat{\boldsymbol{n}}). \tag{2.88}$$

Specular reflection

The second major component of a typical BRDF is *specular* (gloss or highlight) reflection, which depends strongly on the direction of the outgoing light. Consider light reflecting off a mirrored surface (Figure 2.17b). Incident light rays are reflected in a direction that is rotated by 180° around the surface normal $\hat{\boldsymbol{n}}$. Using the same notation as in Equations (2.29–2.30), we can compute the *specular reflection* direction $\hat{\boldsymbol{s}}_i$ as

$$\hat{\boldsymbol{s}}_i = \boldsymbol{v}_\parallel - \boldsymbol{v}_\perp = (2\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T - \boldsymbol{I})\boldsymbol{v}_i. \tag{2.89}$$

The amount of light reflected in a given direction $\hat{\boldsymbol{v}}_r$ thus depends on the angle $\theta_s = \cos^{-1}(\hat{\boldsymbol{v}}_r \cdot \hat{\boldsymbol{s}}_i)$ between the view direction $\hat{\boldsymbol{v}}_r$ and the specular direction $\hat{\boldsymbol{s}}_i$. For example, the Phong (1975) model uses a power of the cosine of the angle,

$$f_s(\theta_s; \lambda) = k_s(\lambda) \cos^{k_e} \theta_s, \tag{2.90}$$

while the Torrance and Sparrow (1967) micro-facet model uses a Gaussian,

$$f_s(\theta_s; \lambda) = k_s(\lambda) \exp(-c_s^2 \theta_s^2). \tag{2.91}$$

Larger exponents k_e (or inverse Gaussian widths c_s) correspond to more specular surfaces with distinct highlights, while smaller exponents better model materials with softer gloss.

Phong shading

Phong (1975) combined the diffuse and specular components of reflection with another term, which he called the *ambient illumination*. This term accounts for the fact that objects are generally illuminated not only by point light sources but also by a general diffuse illumination corresponding to inter-reflection (e.g., the walls in a room) or distant sources, such as the

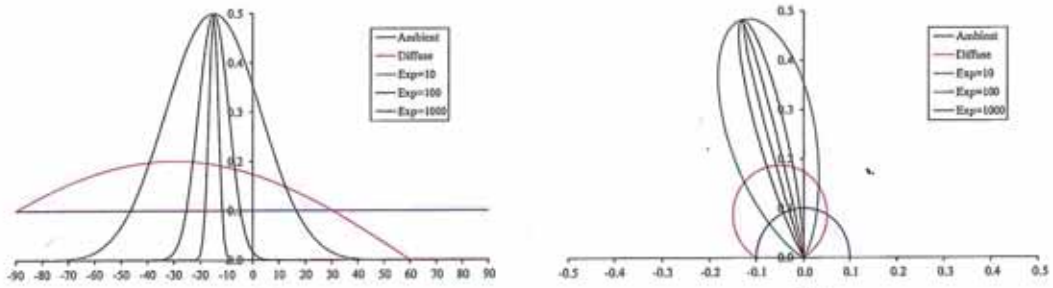


Figure 2.18 Cross-section through a Phong shading model BRDF for a fixed incident illumination direction: (a) component values as a function of angle away from surface normal; (b) polar plot. The value of the Phong exponent k_e is indicated by the "Exp" labels and the light source is at an angle of 30° away from the normal.

blue sky. In the Phong model, the ambient term does not depend on surface orientation, but depends on the color of both the ambient illumination $L_a(\lambda)$ and the object $k_a(\lambda)$,

$$f_a(\lambda) = k_a(\lambda)L_a(\lambda). \tag{2.92}$$

Putting all of these terms together, we arrive at the *Phong shading* model,

$$L_r(\hat{\boldsymbol{v}}_r; \lambda) = k_a(\lambda)L_a(\lambda) + k_d(\lambda)\sum_i L_i(\lambda)[\hat{\boldsymbol{v}}_i \cdot \hat{\boldsymbol{n}}]^+ + k_s(\lambda)\sum_i L_i(\lambda)(\hat{\boldsymbol{v}}_r \cdot \hat{\boldsymbol{s}}_i)^{k_e}. \tag{2.93}$$

Figure 2.18 shows a typical set of Phong shading model components as a function of the angle away from the surface normal (in a plane containing both the lighting direction and the viewer).

Typically, the ambient and diffuse reflection color distributions $k_a(\lambda)$ and $k_d(\lambda)$ are the same, since they are both due to sub-surface scattering (body reflection) inside the surface material (Shafer 1985). The specular reflection distribution $k_s(\lambda)$ is often uniform (white), since it is caused by interface reflections that do not change the light color. (The exception to this are *metallic* materials, such as copper, as opposed to the more common *dielectric* materials, such as plastics.)

The ambient illumination $L_a(\lambda)$ often has a different color cast from the direct light sources $L_i(\lambda)$, e.g., it may be blue for a sunny outdoor scene or yellow for an interior lit with candles or incandescent lights. (The presence of ambient sky illumination in shadowed areas is what often causes shadows to appear bluer than the corresponding lit portions of a scene). Note also that the diffuse component of the Phong model (or of any shading model) depends on the angle of the *incoming* light source $\hat{\boldsymbol{v}}_i$, while the specular component depends on the relative angle between the viewer $\boldsymbol{v}_r$ and the specular reflection direction $\hat{\boldsymbol{s}}_i$ (which itself depends on the incoming light direction $\hat{\boldsymbol{v}}_i$ and the surface normal $\hat{\boldsymbol{n}}$).

The Phong shading model has been superseded in terms of physical accuracy by a number of more recently developed models in computer graphics, including the model developed by Cook and Torrance (1982) based on the original micro-facet model of Torrance and Sparrow (1967). Until recently, most computer graphics hardware implemented the Phong model but the recent advent of programmable pixel shaders makes the use of more complex models feasible.

Chapter 4

Feature detection and matching

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_4, © Springer-Verlag London Limited 2011

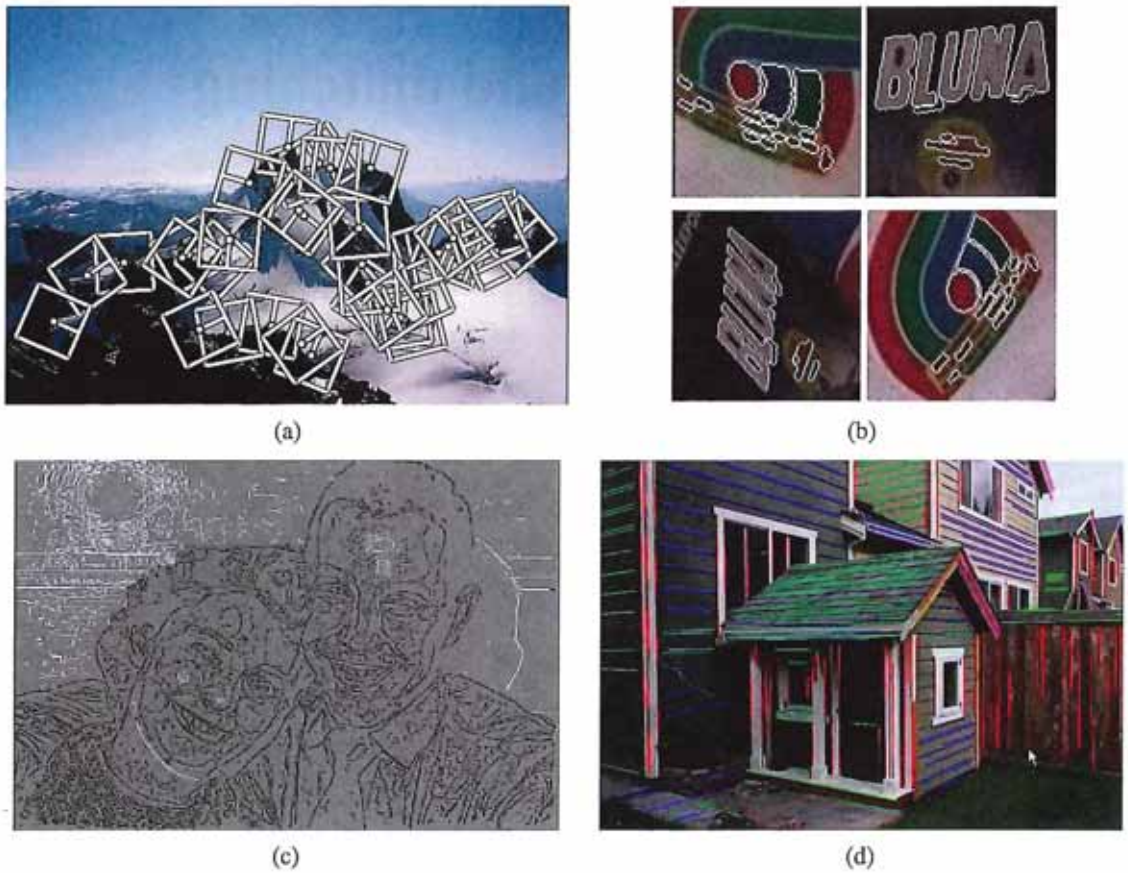


Figure 4.1 A variety of feature detectors and descriptors can be used to analyze, describe and match images: (a) point-like interest operators (Brown, Szeliski, and Winder 2005) © 2005 IEEE; (b) region-like interest operators (Matas, Chum, Urban *et al.* 2004) © 2004 Elsevier; (c) edges (Elder and Goldberg 2001) © 2001 IEEE; (d) straight lines (Sinha, Steedly, Szeliski *et al.* 2008) © 2008 ACM.

Feature detection and matching are an essential component of many computer vision applications. Consider the two pairs of images shown in Figure 4.2. For the first pair, we may wish to *align* the two images so that they can be seamlessly stitched into a composite mosaic (Chapter 9). For the second pair, we may wish to establish a dense set of *correspondences* so that a 3D model can be constructed or an in-between view can be generated (Chapter 11). In either case, what kinds of *features* should you detect and then match in order to establish such an alignment or set of correspondences? Think about this for a few moments before reading on.

The first kind of feature that you may notice are specific locations in the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized feature are often called *keypoint features* or *interest points* (or even *corners*) and are often described by the appearance of patches of pixels surrounding the point location (Section 4.1). Another class of important features are *edges*, e.g., the profile of mountains against the sky, (Section 4.2). These kinds of features can be matched based on their orientation and local appearance (edge profiles) and can also be good indicators of object boundaries and *occlusion* events in image sequences. Edges can be grouped into longer *curves* and *straight line segments*, which can be directly matched or analyzed to find *vanishing points* and hence internal and external camera parameters (Section 4.3).

In this chapter, we describe some practical approaches to detecting such features and also discuss how feature correspondences can be established across different images. Point features are now used in such a wide variety of applications that it is good practice to read and implement some of the algorithms from (Section 4.1). Edges and lines provide information that is complementary to both keypoint and region-based descriptors and are well-suited to describing object boundaries and man-made objects. These alternative descriptors, while extremely useful, can be skipped in a short introductory course.

4.1 Points and patches

Point features can be used to find a sparse set of corresponding locations in different images, often as a pre-cursor to computing camera pose (Chapter 7), which is a prerequisite for computing a denser set of correspondences using stereo matching (Chapter 11). Such correspondences can also be used to align different images, e.g., when stitching image mosaics or performing video stabilization (Chapter 9). They are also used extensively to perform object instance and category recognition (Sections 14.3 and 14.4). A key advantage of keypoints is that they permit matching even in the presence of clutter (occlusion) and large scale and orientation changes.

Feature-based correspondence techniques have been used since the early days of stereo matching (Hannah 1974; Moravec 1983; Hannah 1988) and have more recently gained popularity for image-stitching applications (Zoghlami, Faugeras, and Deriche 1997; Brown and Lowe 2007) as well as fully automated 3D modeling (Beardsley, Torr, and Zisserman 1996; Schaffalitzky and Zisserman 2002; Brown and Lowe 2003; Snavely, Seitz, and Szeliski 2006).

There are two main approaches to finding feature points and their correspondences. The first is to find features in one image that can be accurately *tracked* using a local search technique, such as correlation or least squares (Section 4.1.4). The second is to independently

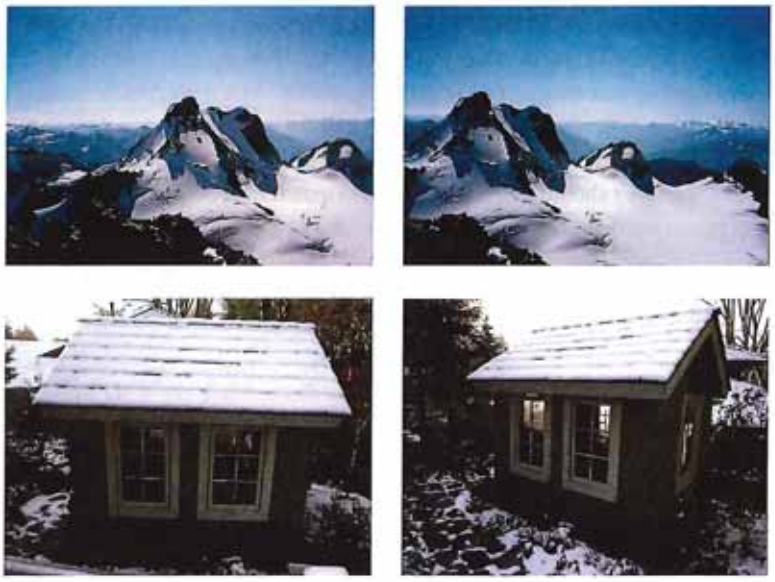


Figure 4.2 Two pairs of images to be matched. What kinds of feature might one use to establish a set of *correspondences* between these images?

detect features in all the images under consideration and then *match* features based on their local appearance (Section 4.1.3). The former approach is more suitable when images are taken from nearby viewpoints or in rapid succession (e.g., video sequences), while the latter is more suitable when a large amount of motion or appearance change is expected, e.g., in stitching together panoramas (Brown and Lowe 2007), establishing correspondences in *wide baseline stereo* (Schaffalitzky and Zisserman 2002), or performing object recognition (Fergus, Perona, and Zisserman 2007).

In this section, we split the keypoint detection and matching pipeline into four separate stages. During the *feature detection* (extraction) stage (Section 4.1.1), each image is searched for locations that are likely to match well in other images. At the *feature description* stage (Section 4.1.2), each region around detected keypoint locations is converted into a more compact and stable (invariant) *descriptor* that can be matched against other descriptors. The *feature matching* stage (Section 4.1.3) efficiently searches for likely matching candidates in other images. The *feature tracking* stage (Section 4.1.4) is an alternative to the third stage that only searches a small neighborhood around each detected feature and is therefore more suitable for video processing.

A wonderful example of all of these stages can be found in David Lowe's (2004) paper, which describes the development and refinement of his *Scale Invariant Feature Transform* (SIFT). Comprehensive descriptions of alternative techniques can be found in a series of survey and evaluation papers covering both feature detection (Schmid, Mohr, and Bauckhage 2000; Mikolajczyk, Tuytelaars, Schmid *et al.* 2005; Tuytelaars and Mikolajczyk 2007) and feature descriptors (Mikolajczyk and Schmid 2005). Shi and Tomasi (1994) and Triggs (2004) also provide nice reviews of feature detection techniques.

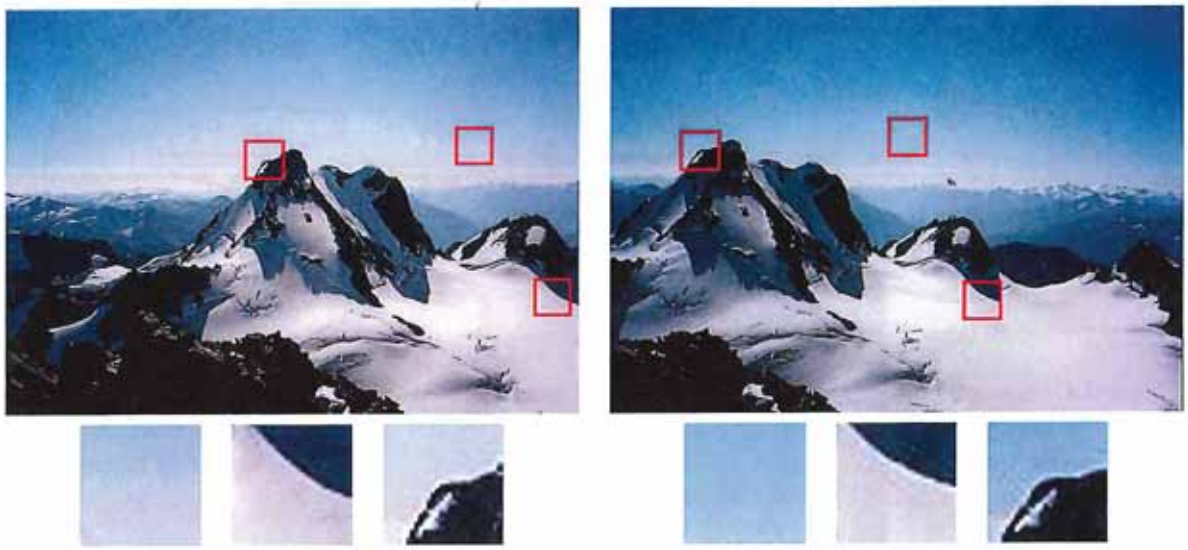


Figure 4.3 Image pairs with extracted patches below. Notice how some patches can be localized or matched with higher accuracy than others.

4.1.1 Feature detectors

How can we find image locations where we can reliably find correspondences with other images, i.e., what are good features to track (Shi and Tomasi 1994; Triggs 2004)? Look again at the image pair shown in Figure 4.3 and at the three sample *patches* to see how well they might be matched or tracked. As you may notice, textureless patches are nearly impossible to localize. Patches with large contrast changes (gradients) are easier to localize, although straight line segments at a single orientation suffer from the *aperture problem* (Horn and Schunck 1981; Lucas and Kanade 1981; Anandan 1989), i.e., it is only possible to align the patches along the direction *normal* to the edge direction (Figure 4.4b). Patches with gradients in at least two (significantly) different orientations are the easiest to localize, as shown schematically in Figure 4.4a.

These intuitions can be formalized by looking at the simplest possible matching criterion for comparing two image patches, i.e., their (weighted) summed square difference,

$$E_{\text{WSSD}}(\boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{4.1}$$

where I_0 and I_1 are the two images being compared, $\boldsymbol{u} = (u, v)$ is the *displacement* vector, $w(\boldsymbol{x})$ is a spatially varying weighting (or window) function, and the summation i is over all the pixels in the patch. Note that this is the same formulation we later use to estimate motion between complete images (Section 8.1).

When performing feature detection, we do not know which other image locations the feature will end up being matched against. Therefore, we can only compute how stable this metric is with respect to small variations in position $\Delta\boldsymbol{u}$ by comparing an image patch against

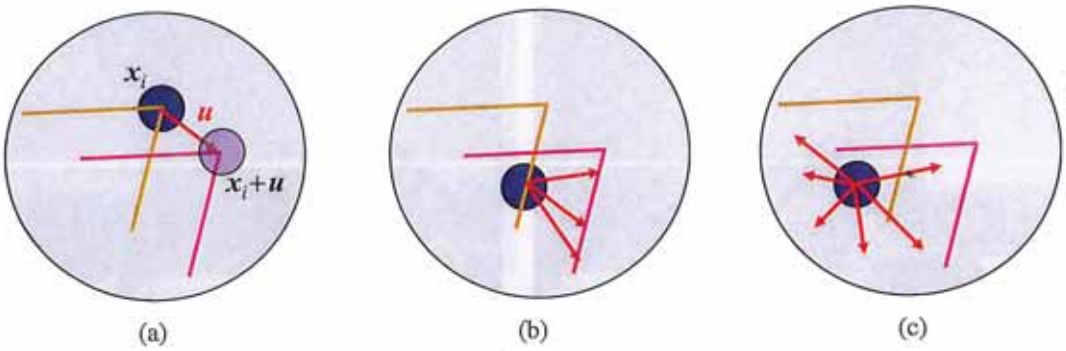


Figure 4.4 Aperture problems for different image patches: (a) stable ("corner-like") flow; (b) classic aperture problem (barber-pole illusion); (c) textureless region. The two images I_0 (yellow) and I_1 (red) are overlaid. The red vector $\boldsymbol{u}$ indicates the displacement between the patch centers and the $w(\boldsymbol{x}_i)$ weighting function (patch window) is shown as a dark circle.

itself, which is known as an *auto-correlation function* or *surface*

$$E_{\mathrm{AC}}(\Delta \boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \tag{4.2}$$

(Figure 4.5).[1] Note how the auto-correlation surface for the textured flower bed (Figure 4.5b and the red cross in the lower right quadrant of Figure 4.5a) exhibits a strong minimum, indicating that it can be well localized. The correlation surface corresponding to the roof edge (Figure 4.5c) has a strong ambiguity along one direction, while the correlation surface corresponding to the cloud region (Figure 4.5d) has no stable minimum.

Using a Taylor Series expansion of the image function $I_0(\boldsymbol{x}_i + \Delta \boldsymbol{u}) \approx I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i) \cdot \Delta \boldsymbol{u}$ (Lucas and Kanade 1981; Shi and Tomasi 1994), we can approximate the auto-correlation surface as

$$\begin{aligned} E_{\mathrm{AC}}(\Delta \boldsymbol{u}) &= \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i + \Delta \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 && (4.3) \\ &\approx \sum_i w(\boldsymbol{x}_i)[I_0(\boldsymbol{x}_i) + \nabla I_0(\boldsymbol{x}_i) \cdot \Delta \boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 && (4.4) \\ &= \sum_i w(\boldsymbol{x}_i)[\nabla I_0(\boldsymbol{x}_i) \cdot \Delta \boldsymbol{u}]^2 && (4.5) \\ &= \Delta \boldsymbol{u}^T \boldsymbol{A} \Delta \boldsymbol{u}, && (4.6) \end{aligned}$$

where

$$\nabla I_0(\boldsymbol{x}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y}\right)(\boldsymbol{x}_i) \tag{4.7}$$

is the *image gradient* at $\boldsymbol{x}_i$. This gradient can be computed using a variety of techniques (Schmid, Mohr, and Bauckhage 2000). The classic "Harris" detector (Harris and Stephens 1988) uses a [-2 -1 0 1 2] filter, but more modern variants (Schmid, Mohr, and Bauckhage 2000; Triggs 2004) convolve the image with horizontal and vertical derivatives of a Gaussian (typically with $\sigma = 1$).

[1] Strictly speaking, a correlation is the *product* of two patches (3.12); I'm using the term here in a more qualitative sense. The weighted sum of squared differences is often called an *SSD surface* (Section 8.1).

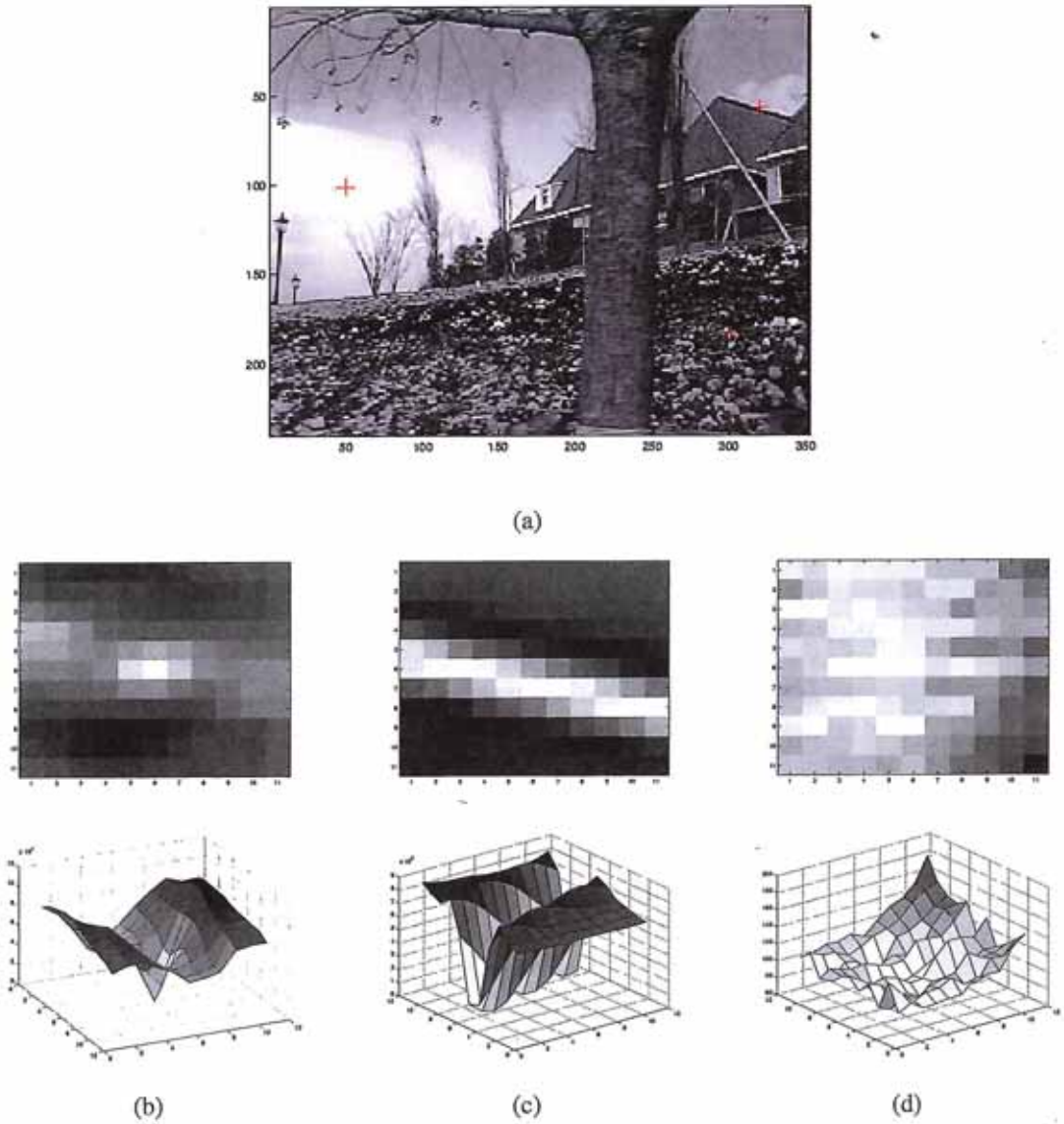


Figure 4.5 Three auto-correlation surfaces $E_{\mathrm{AC}}(\Delta \boldsymbol{u})$ shown as both grayscale images and surface plots: (a) The original image is marked with three red crosses to denote where the auto-correlation surfaces were computed; (b) this patch is from the flower bed (good unique minimum); (c) this patch is from the roof edge (one-dimensional aperture problem); and (d) this patch is from the cloud (no good peak). Each grid point in figures b–d is one value of $\Delta \boldsymbol{u}$.

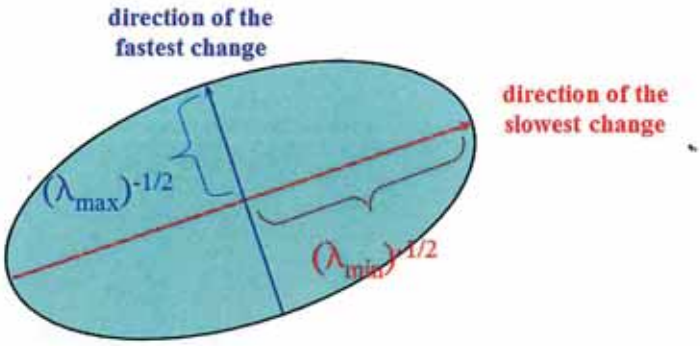


Figure 4.6 Uncertainty ellipse corresponding to an eigenvalue analysis of the auto-correlation matrix $\boldsymbol{A}$.

The auto-correlation matrix $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{4.8}$$

where we have replaced the weighted summations with discrete convolutions with the weighting kernel w. This matrix can be interpreted as a tensor (multiband) image, where the outer products of the gradients ∇I are convolved with a weighting function w to provide a per-pixel estimate of the local (quadratic) shape of the auto-correlation function.

As first shown by Anandan (1984; 1989) and further discussed in Section 8.1.3 and (8.44), the inverse of the matrix $\boldsymbol{A}$ provides a lower bound on the uncertainty in the location of a matching patch. It is therefore a useful indicator of which patches can be reliably matched. The easiest way to visualize and reason about this uncertainty is to perform an eigenvalue analysis of the auto-correlation matrix $\boldsymbol{A}$, which produces two eigenvalues (λ_0, λ_1) and two eigenvector directions (Figure 4.6). Since the larger uncertainty depends on the smaller eigenvalue, i.e., $\lambda_0^{-1/2}$, it makes sense to find maxima in the smaller eigenvalue to locate good features to track (Shi and Tomasi 1994).

Förstner–Harris. While Anandan and Lucas and Kanade (1981) were the first to analyze the uncertainty structure of the auto-correlation matrix, they did so in the context of associating certainties with optic flow measurements. Förstner (1986) and Harris and Stephens (1988) were the first to propose using local maxima in rotationally invariant scalar measures derived from the auto-correlation matrix to locate keypoints for the purpose of sparse feature matching. (Schmid, Mohr, and Bauckhage (2000); Triggs (2004) give more detailed historical reviews of feature detection algorithms.) Both of these techniques also proposed using a Gaussian weighting window instead of the previously used square patches, which makes the detector response insensitive to in-plane image rotations.

The minimum eigenvalue λ_0 (Shi and Tomasi 1994) is not the only quantity that can be used to find keypoints. A simpler quantity, proposed by Harris and Stephens (1988), is

$$\det(\boldsymbol{A}) - \alpha \operatorname{trace}(\boldsymbol{A})^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \tag{4.9}$$

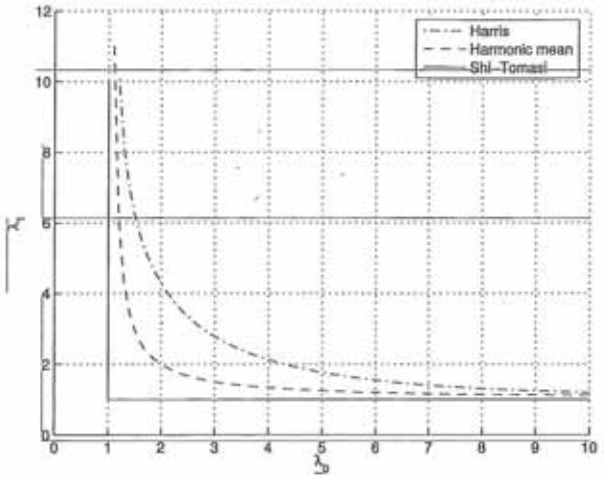


Figure 4.7 Isocontours of popular keypoint detection functions (Brown, Szeliski, and Winder 2004). Each detector looks for points where the eigenvalues λ_0, λ_1 of $\boldsymbol{A} = w * \nabla I \nabla I^T$ are both large.

with $\alpha = 0.06$. Unlike eigenvalue analysis, this quantity does not require the use of square roots and yet is still rotationally invariant and also downweights edge-like features where $\lambda_1 \gg \lambda_0$. Triggs (2004) suggests using the quantity

$$\lambda_0 - \alpha\lambda_1 \tag{4.10}$$

(say, with $\alpha = 0.05$), which also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue. He also shows how the basic 2×2 Hessian can be extended to parametric motions to detect points that are also accurately localizable in scale and rotation. Brown, Szeliski, and Winder (2005), on the other hand, use the harmonic mean,

$$\frac{\det \boldsymbol{A}}{\operatorname{tr} \boldsymbol{A}} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1}, \tag{4.11}$$

which is a smoother function in the region where $\lambda_0 \approx \lambda_1$. Figure 4.7 shows isocontours of the various interest point operators, from which we can see how the two eigenvalues are blended to determine the final interest value.

The steps in the basic auto-correlation-based keypoint detector are summarized in Algorithm 4.1. Figure 4.8 shows the resulting interest operator responses for the classic Harris detector as well as the difference of Gaussian (DoG) detector discussed below.

Adaptive non-maximal suppression (ANMS). While most feature detectors simply look for local maxima in the interest function, this can lead to an uneven distribution of feature points across the image, e.g., points will be denser in regions of higher contrast. To mitigate this problem, Brown, Szeliski, and Winder (2005) only detect features that are both local maxima and whose response value is significantly (10%) greater than that of all of its neighbors within a radius r (Figure 4.9c–d). They devise an efficient way to associate suppression radii with all local maxima by first sorting them by their response strength and then creating a second list sorted by decreasing suppression radius (Brown, Szeliski, and

1. Compute the horizontal and vertical derivatives of the image I_x and I_y by convolving the original image with derivatives of Gaussians (Section 3.2.3).
2. Compute the three images corresponding to the outer products of these gradients. (The matrix $\boldsymbol{A}$ is symmetric, so only three entries are needed.)
3. Convolve each of these images with a larger Gaussian.
4. Compute a scalar interest measure using one of the formulas discussed above.
5. Find local maxima above a certain threshold and report them as detected feature point locations.

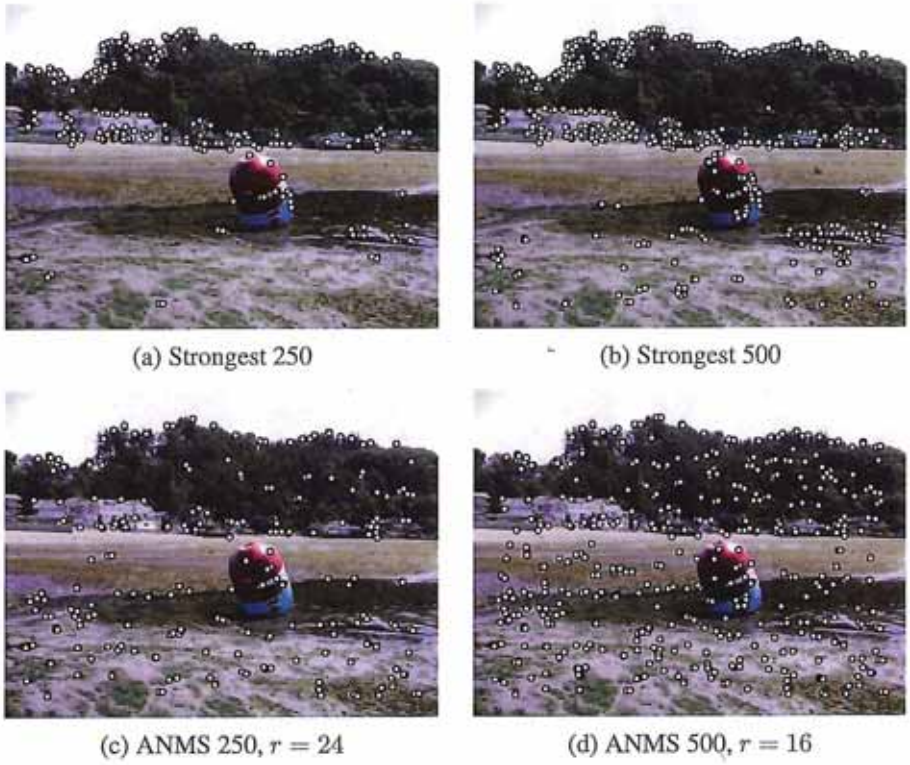
Algorithm 4.1 Outline of a basic feature detection algorithm.

(a) (b) (c)

Figure 4.8 Interest operator responses: (a) Sample image, (b) Harris response, and (c) DoG response. The circle sizes and colors indicate the scale at which each interest point was detected. Notice how the two detectors tend to respond at complementary locations.

Winder 2005). Figure 4.9 shows a qualitative comparison of selecting the top n features and using ANMS.

Measuring repeatability. Given the large number of feature detectors that have been developed in computer vision, how can we decide which ones to use? Schmid, Mohr, and Bauckhage (2000) were the first to propose measuring the *repeatability* of feature detectors, which they define as the frequency with which keypoints detected in one image are found within ϵ (say, $\epsilon = 1.5$) pixels of the corresponding location in a transformed image. In their paper, they transform their planar images by applying rotations, scale changes, illumination changes, viewpoint changes, and adding noise. They also measure the *information content* available at each detected feature point, which they define as the entropy of a set of rotationally invariant local grayscale descriptors. Among the techniques they survey, they find that the improved (Gaussian derivative) version of the Harris operator with $\sigma_d = 1$ (scale of the derivative Gaussian) and $\sigma_i = 2$ (scale of the integration Gaussian) works best.



(a) Strongest 250 (b) Strongest 500

(c) ANMS 250, $r = 24$ (d) ANMS 500, $r = 16$

Figure 4.9 Adaptive non-maximal suppression (ANMS) (Brown, Szeliski, and Winder 2005) © 2005 IEEE: The upper two images show the strongest 250 and 500 interest points, while the lower two images show the interest points selected with adaptive non-maximal suppression, along with the corresponding suppression radius r. Note how the latter features have a much more uniform spatial distribution across the image.

Scale invariance

In many situations, detecting features at the finest stable scale possible may not be appropriate. For example, when matching images with little high frequency detail (e.g., clouds), fine-scale features may not exist.

One solution to the problem is to extract features at a variety of scales, e.g., by performing the same operations at multiple resolutions in a pyramid and then matching features at the same level. This kind of approach is suitable when the images being matched do not undergo large scale changes, e.g., when matching successive aerial images taken from an airplane or stitching panoramas taken with a fixed-focal-length camera. Figure 4.10 shows the output of one such approach, the multi-scale, oriented patch detector of Brown, Szeliski, and Winder (2005), for which responses at five different scales are shown.

However, for most object recognition applications, the scale of the object in the image is unknown. Instead of extracting features at many different scales and then matching all of them, it is more efficient to extract features that are stable in both location *and* scale (Lowe 2004; Mikolajczyk and Schmid 2004).

Early investigations into scale selection were performed by Lindeberg (1993; 1998b), who first proposed using extrema in the Laplacian of Gaussian (LoG) function as interest

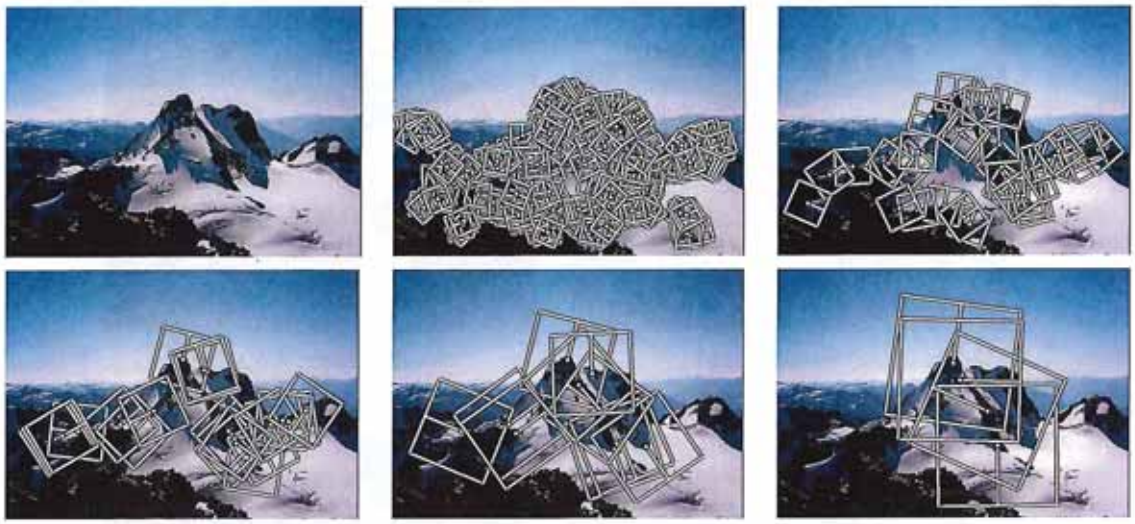


Figure 4.10 Multi-scale oriented patches (MOPS) extracted at five pyramid levels (Brown, Szeliski, and Winder 2005) © 2005 IEEE. The boxes show the feature orientation and the region from which the descriptor vectors are sampled.

point locations. Based on this work, Lowe (2004) proposed computing a set of sub-octave Difference of Gaussian filters (Figure 4.11a), looking for 3D (space+scale) maxima in the resulting structure (Figure 4.11b), and then computing a sub-pixel space+scale location using a quadratic fit (Brown and Lowe 2002). The number of sub-octave levels was determined, after careful empirical investigation, to be three, which corresponds to a quarter-octave pyramid, which is the same as used by Triggs (2004).

As with the Harris operator, pixels where there is strong asymmetry in the local curvature of the indicator function (in this case, the DoG) are rejected. This is implemented by first computing the local Hessian of the difference image D,

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}, \tag{4.12}$$

and then rejecting keypoints for which

$$\frac{\mathrm{Tr}(\boldsymbol{H})^2}{\mathrm{Det}(\boldsymbol{H})} > 10. \tag{4.13}$$

While Lowe's Scale Invariant Feature Transform (SIFT) performs well in practice, it is not based on the same theoretical foundation of maximum spatial stability as the auto-correlation-based detectors. (In fact, its detection locations are often complementary to those produced by such techniques and can therefore be used in conjunction with these other approaches.) In order to add a scale selection mechanism to the Harris corner detector, Mikolajczyk and Schmid (2004) evaluate the Laplacian of Gaussian function at each detected Harris point (in a multi-scale pyramid) and keep only those points for which the Laplacian is extremal (larger or smaller than both its coarser and finer-level values). An optional iterative refinement for both scale and position is also proposed and evaluated. Additional examples of scale invariant

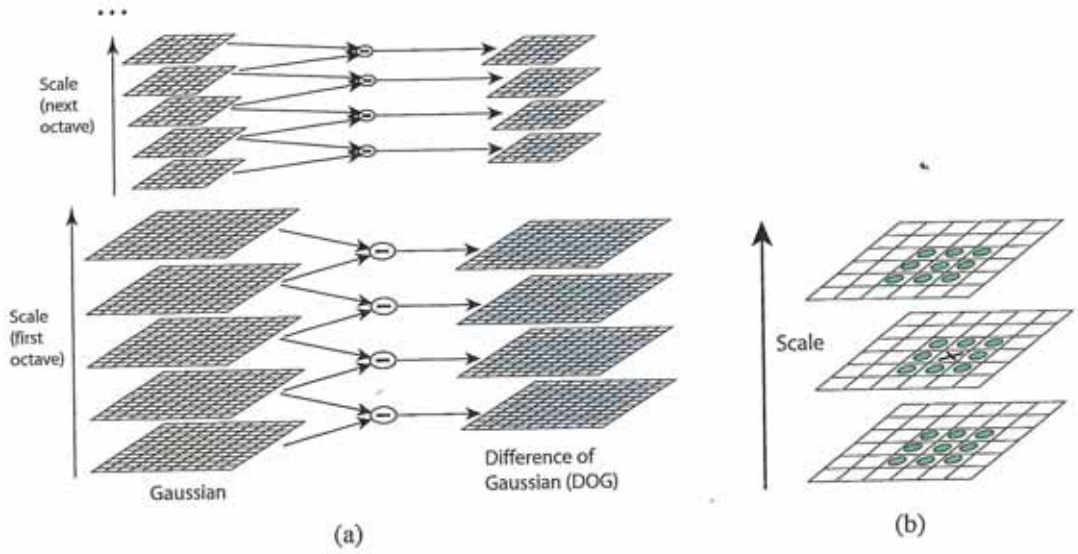


Figure 4.11 Scale-space feature detection using a sub-octave Difference of Gaussian pyramid (Lowe 2004) © 2004 Springer: (a) Adjacent levels of a sub-octave Gaussian pyramid are subtracted to produce Difference of Gaussian images; (b) extrema (maxima and minima) in the resulting 3D volume are detected by comparing a pixel to its 26 neighbors.

region detectors are discussed by Mikolajczyk, Tuytelaars, Schmid *et al.* (2005); Tuytelaars and Mikolajczyk (2007).

Rotational invariance and orientation estimation

In addition to dealing with scale changes, most image matching and object recognition algorithms need to deal with (at least) in-plane image rotation. One way to deal with this problem is to design descriptors that are rotationally invariant (Schmid and Mohr 1997), but such descriptors have poor discriminability, i.e. they map different looking patches to the same descriptor.

A better method is to estimate a *dominant orientation* at each detected keypoint. Once the local orientation and scale of a keypoint have been estimated, a scaled and oriented patch around the detected point can be extracted and used to form a feature descriptor (Figures 4.10 and 4.17).

The simplest possible orientation estimate is the average gradient within a region around the keypoint. If a Gaussian weighting function is used (Brown, Szeliski, and Winder 2005), this average gradient is equivalent to a first-order steerable filter (Section 3.2.3), i.e., it can be computed using an image convolution with the horizontal and vertical derivatives of Gaussian filter (Freeman and Adelson 1991). In order to make this estimate more reliable, it is usually preferable to use a larger aggregation window (Gaussian kernel size) than detection window (Brown, Szeliski, and Winder 2005). The orientations of the square boxes shown in Figure 4.10 were computed using this technique.

Sometimes, however, the averaged (signed) gradient in a region can be small and therefore

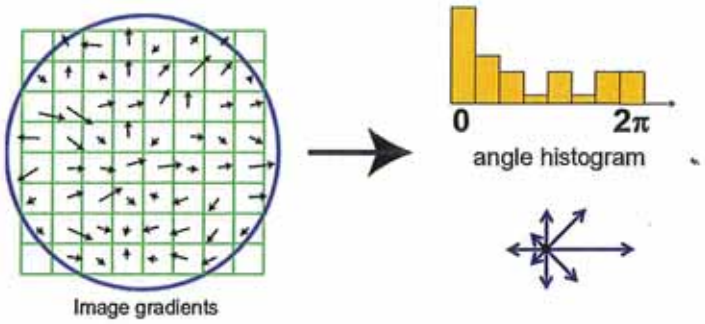


Figure 4.12 A dominant orientation estimate can be computed by creating a histogram of all the gradient orientations (weighted by their magnitudes or after thresholding out small gradients) and then finding the significant peaks in this distribution (Lowe 2004) © 2004 Springer.

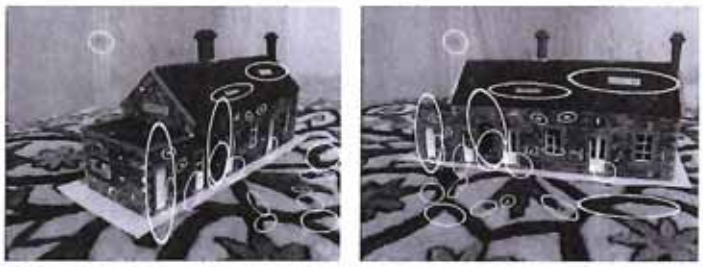


Figure 4.13 Affine region detectors used to match two images taken from dramatically different viewpoints (Mikolajczyk and Schmid 2004) © 2004 Springer.

an unreliable indicator of orientation. A more reliable technique is to look at the *histogram* of orientations computed around the keypoint. Lowe (2004) computes a 36-bin histogram of edge orientations weighted by both gradient magnitude and Gaussian distance to the center, finds all peaks within 80% of the global maximum, and then computes a more accurate orientation estimate using a three-bin parabolic fit (Figure 4.12).

Affine invariance

While scale and rotation invariance are highly desirable, for many applications such as *wide baseline stereo matching* (Pritchett and Zisserman 1998; Schaffalitzky and Zisserman 2002) or location recognition (Chum, Philbin, Sivic *et al.* 2007), full affine invariance is preferred. Affine-invariant detectors not only respond at consistent locations after scale and orientation changes, they also respond consistently across affine deformations such as (local) perspective foreshortening (Figure 4.13). In fact, for a small enough patch, any continuous image warping can be well approximated by an affine deformation.

To introduce affine invariance, several authors have proposed fitting an ellipse to the autocorrelation or Hessian matrix (using eigenvalue analysis) and then using the principal axes and ratios of this fit as the affine coordinate frame (Lindeberg and Garding 1997; Baumberg

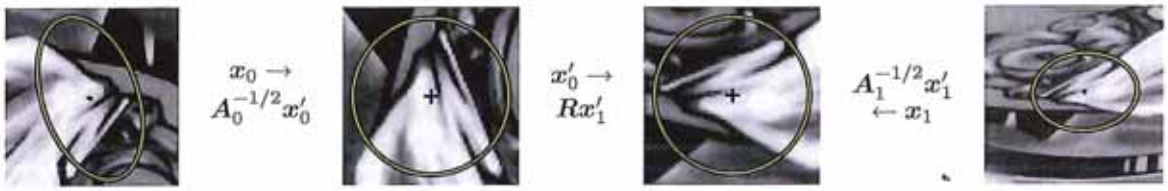


Figure 4.14 Affine normalization using the second moment matrices, as described by Mikolajczyk, Tuytelaars, Schmid *et al.* (2005) © 2005 Springer. After image coordinates are transformed using the matrices $A_0^{-1/2}$ and $A_1^{-1/2}$, they are related by a pure rotation R, which can be estimated using a dominant orientation technique.

Figure 4.15 Maximally stable extremal regions (MSERs) extracted and matched from a number of images (Matas, Chum, Urban *et al.* 2004) © 2004 Elsevier.

2000; Mikolajczyk and Schmid 2004; Mikolajczyk, Tuytelaars, Schmid *et al.* 2005; Tuytelaars and Mikolajczyk 2007). Figure 4.14 shows how the square root of the moment matrix can be used to transform local patches into a frame which is similar up to rotation.

Another important affine invariant region detector is the maximally stable extremal region (MSER) detector developed by Matas, Chum, Urban *et al.* (2004). To detect MSERs, binary regions are computed by thresholding the image at all possible gray levels (the technique therefore only works for grayscale images). This operation can be performed efficiently by first sorting all pixels by gray value and then incrementally adding pixels to each connected component as the threshold is changed (Nistér and Stewénius 2008). As the threshold is changed, the area of each component (region) is monitored; regions whose rate of change of area with respect to the threshold is minimal are defined as *maximally stable*. Such regions are therefore invariant to both affine geometric and photometric (linear bias-gain or smooth monotonic) transformations (Figure 4.15). If desired, an affine coordinate frame can be fit to each detected region using its moment matrix.

The area of feature point detectors continues to be very active, with papers appearing every year at major computer vision conferences (Xiao and Shah 2003; Koethe 2003; Carneiro and Jepson 2005; Kenney, Zuliani, and Manjunath 2005; Bay, Tuytelaars, and Van Gool 2006; Platel, Balmachnova, Florack *et al.* 2006; Rosten and Drummond 2006). Mikolajczyk, Tuytelaars, Schmid *et al.* (2005) survey a number of popular affine region detectors and provide experimental comparisons of their invariance to common image transformations such as scaling, rotations, noise, and blur. These experimental results, code, and pointers to the surveyed papers can be found on their Web site at http://www.robots.ox.ac.uk/~vgg/research/affine/.

Of course, keypoints are not the only features that can be used for registering images. Zoghlami, Faugeras, and Deriche (1997) use line segments as well as point-like features to estimate homographies between pairs of images, whereas Bartoli, Coquerelle, and Sturm (2004) use line segments with local correspondences along the edges to extract 3D structure

Figure 4.16 Feature matching: how can we extract local descriptors that are invariant to inter-image variations and yet still discriminative enough to establish correct correspondences?

and motion. Tuytelaars and Van Gool (2004) use affine invariant regions to detect correspondences for wide baseline stereo matching, whereas Kadir, Zisserman, and Brady (2004) detect salient regions where patch entropy and its rate of change with scale are locally maximal. Corso and Hager (2005) use a related technique to fit 2D oriented Gaussian kernels to homogeneous regions. More details on techniques for finding and matching curves, lines, and regions can be found later in this chapter.

4.1.2 Feature descriptors

After detecting features (keypoints), we must *match* them, i.e., we must determine which features come from corresponding locations in different images. In some situations, e.g., for video sequences (Shi and Tomasi 1994) or for stereo pairs that have been *rectified* (Zhang, Deriche, Faugeras *et al.* 1995; Loop and Zhang 1999; Scharstein and Szeliski 2002), the local motion around each feature point may be mostly translational. In this case, simple error metrics, such as the *sum of squared differences* or *normalized cross-correlation*, described in Section 8.1 can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid (2005), discussed below, uses cross-correlation.) Because feature points may not be exactly located, a more accurate matching score can be computed by performing incremental motion refinement as described in Section 8.1.3 but this can be time consuming and can sometimes even decrease performance (Brown, Szeliski, and Winder 2005).

In most cases, however, the local appearance of features will change in orientation and scale, and sometimes even undergo affine deformations. Extracting a local scale, orientation, or affine frame estimate and then using this to resample the patch before forming the feature descriptor is thus usually preferable (Figure 4.17).

Even after compensating for these changes, the local appearance of image patches will usually still vary from image to image. How can we make image descriptors more invariant to such changes, while still preserving discriminability between different (non-corresponding) patches (Figure 4.16)? Mikolajczyk and Schmid (2005) review some recently developed view-invariant local image descriptors and experimentally compare their performance. Below, we describe a few of these descriptors in more detail.

Bias and gain normalization (MOPS). For tasks that do not exhibit large amounts of foreshortening, such as image stitching, simple normalized intensity patches perform reasonably well and are simple to implement (Brown, Szeliski, and Winder 2005) (Figure 4.17). In

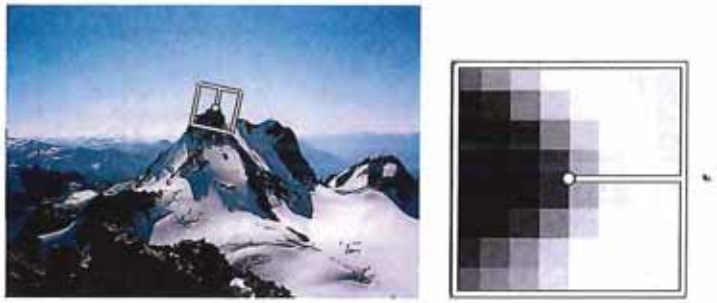


Figure 4.17 MOPS descriptors are formed using an 8×8 sampling of bias and gain normalized intensity values, with a sample spacing of five pixels relative to the detection scale (Brown, Szeliski, and Winder 2005) © 2005 IEEE. This low frequency sampling gives the features some robustness to interest point location error and is achieved by sampling at a higher pyramid level than the detection scale.

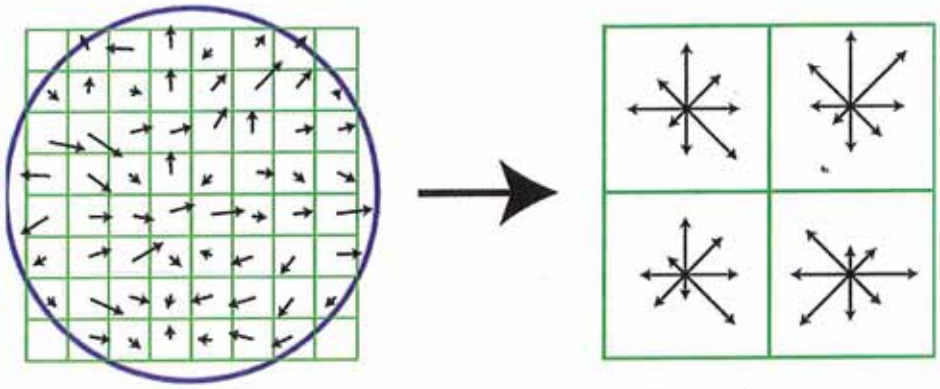
order to compensate for slight inaccuracies in the feature point detector (location, orientation, and scale), these multi-scale oriented patches (MOPS) are sampled at a spacing of five pixels relative to the detection scale, using a coarser level of the image pyramid to avoid aliasing. To compensate for affine photometric variations (linear exposure changes or bias and gain, (3.3)), patch intensities are re-scaled so that their mean is zero and their variance is one.

Scale invariant feature transform (SIFT). SIFT features are formed by computing the gradient at each pixel in a 16×16 window around the detected keypoint, using the appropriate level of the Gaussian pyramid at which the keypoint was detected. The gradient magnitudes are downweighted by a Gaussian fall-off function (shown as a blue circle in (Figure 4.18a) in order to reduce the influence of gradients far from the center, as these are more affected by small misregistrations.

In each 4×4 quadrant, a gradient orientation histogram is formed by (conceptually) adding the weighted gradient value to one of eight orientation histogram bins. To reduce the effects of location and dominant orientation misestimation, each of the original 256 weighted gradient magnitudes is softly added to $2 \times 2 \times 2$ histogram bins using trilinear interpolation. Softly distributing values to adjacent histogram bins is generally a good idea in any application where histograms are being computed, e.g., for Hough transforms (Section 4.3.2) or local histogram equalization (Section 3.1.4).

The resulting 128 non-negative values form a raw version of the SIFT descriptor vector. To reduce the effects of contrast or gain (additive variations are already removed by the gradient), the 128-D vector is normalized to unit length. To further make the descriptor robust to other photometric variations, values are clipped to 0.2 and the resulting vector is once again renormalized to unit length.

PCA-SIFT. Ke and Sukthankar (2004) propose a simpler way to compute descriptors inspired by SIFT; it computes the x and y (gradient) derivatives over a 39×39 patch and then reduces the resulting 3042-dimensional vector to 36 using principal component analysis (PCA) (Section 14.2.1 and Appendix A.1.2). Another popular variant of SIFT is SURF (Bay, Tuytelaars, and Van Gool 2006), which uses box filters to approximate the derivatives and



(a) image gradients (b) keypoint descriptor

Figure 4.18 A schematic representation of Lowe's (2004) scale invariant feature transform (SIFT): (a) Gradient orientations and magnitudes are computed at each pixel and weighted by a Gaussian fall-off function (blue circle). (b) A weighted gradient orientation histogram is then computed in each subregion, using trilinear interpolation. While this figure shows an 8×8 pixel patch and a 2×2 descriptor array, Lowe's actual implementation uses 16×16 patches and a 4×4 array of eight-bin histograms.

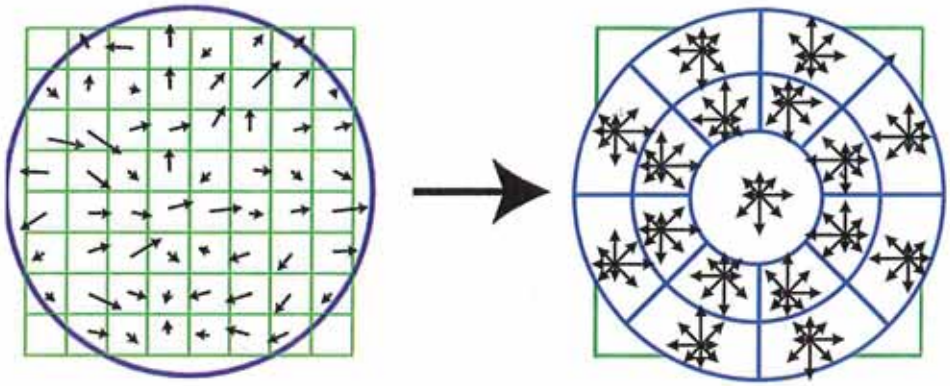
integrals used in SIFT.

Gradient location-orientation histogram (GLOH). This descriptor, developed by Mikolajczyk and Schmid (2005), is a variant on SIFT that uses a log-polar binning structure instead of the four quadrants used by Lowe (2004) (Figure 4.19). The spatial bins are of radius 6, 11, and 15, with eight angular bins (except for the central region), for a total of 17 spatial bins and 16 orientation bins. The 272-dimensional histogram is then projected onto a 128-dimensional descriptor using PCA trained on a large database. In their evaluation, Mikolajczyk and Schmid (2005) found that GLOH, which has the best performance overall, outperforms SIFT by a small margin.

Steerable filters. Steerable filters (Section 3.2.3) are combinations of derivative of Gaussian filters that permit the rapid computation of even and odd (symmetric and anti-symmetric) edge-like and corner-like features at all possible orientations (Freeman and Adelson 1991). Because they use reasonably broad Gaussians, they too are somewhat insensitive to localization and orientation errors.

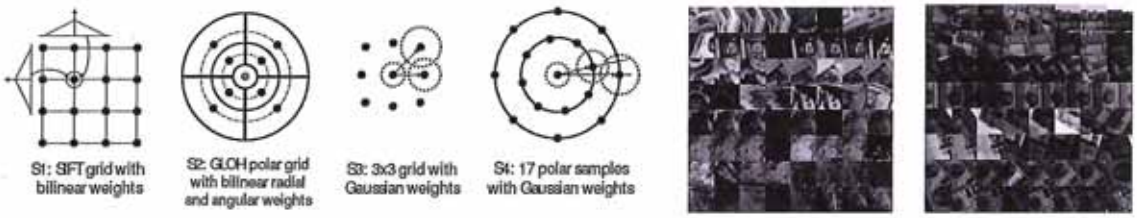
Performance of local descriptors. Among the local descriptors that Mikolajczyk and Schmid (2005) compared, they found that GLOH performed best, followed closely by SIFT (see Figure 4.25). They also present results for many other descriptors not covered in this book.

The field of feature descriptors continues to evolve rapidly, with some of the newer techniques looking at local color information (van de Weijer and Schmid 2006; Abdel-Hakim and Farag 2006). Winder and Brown (2007) develop a multi-stage framework for feature descriptor computation that subsumes both SIFT and GLOH (Figure 4.20a) and also allows



(a) image gradients (b) keypoint descriptor

Figure 4.19 The gradient location-orientation histogram (GLOH) descriptor uses log-polar bins instead of square bins to compute orientation histograms (Mikolajczyk and Schmid 2005).



(a) (b)

Figure 4.20 Spatial summation blocks for SIFT, GLOH, and some newly developed feature descriptors (Winder and Brown 2007) © 2007 IEEE: (a) The parameters for the new features, e.g., their Gaussian weights, are learned from a training database of (b) matched real-world image patches obtained from robust structure from motion applied to Internet photo collections (Hua, Brown, and Winder 2007).

them to learn optimal parameters for newer descriptors that outperform previous hand-tuned descriptors. Hua, Brown, and Winder (2007) extend this work by learning lower-dimensional projections of higher-dimensional descriptors that have the best discriminative power. Both of these papers use a database of real-world image patches (Figure 4.20b) obtained by sampling images at locations that were reliably matched using a robust structure-from-motion algorithm applied to Internet photo collections (Snavely, Seitz, and Szeliski 2006; Goesele, Snavely, Curless *et al.* 2007). In concurrent work, Tola, Lepetit, and Fua (2010) developed a similar DAISY descriptor for dense stereo matching and optimized its parameters based on ground truth stereo data.

While these techniques construct feature detectors that optimize for repeatability across *all* object classes, it is also possible to develop class- or instance-specific feature detectors that maximize *discriminability* from other classes (Ferencz, Learned-Miller, and Malik 2008).

Figure 4.21 Recognizing objects in a cluttered scene (Lowe 2004) © 2004 Springer. Two of the training images in the database are shown on the left. These are matched to the cluttered scene in the middle using SIFT features, shown as small squares in the right image. The affine warp of each recognized database image onto the scene is shown as a larger parallelogram in the right image.

4.1.3 Feature matching

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images. In this section, we divide this problem into two separate components. The first is to select a *matching strategy*, which determines which correspondences are passed on to the next stage for further processing. The second is to devise efficient *data structures* and *algorithms* to perform this matching as quickly as possible. (See the discussion of related techniques in Section 14.3.2.)

Matching strategy and error rates

Determining which feature matches are reasonable to process further depends on the context in which the matching is being performed. Say we are given two images that overlap to a fair amount (e.g., for image stitching, as in Figure 4.16, or for tracking objects in a video). We know that most features in one image are likely to match the other image, although some may not match because they are occluded or their appearance has changed too much.

On the other hand, if we are trying to recognize how many known objects appear in a cluttered scene (Figure 4.21), most of the features may not match. Furthermore, a large number of potentially matching objects must be searched, which requires more efficient strategies, as described below.

To begin with, we assume that the feature descriptors have been designed so that Euclidean (vector magnitude) distances in feature space can be directly used for ranking potential matches. If it turns out that certain parameters (axes) in a descriptor are more reliable than others, it is usually preferable to re-scale these axes ahead of time, e.g., by determining how much they vary when compared against other known good matches (Hua, Brown, and Winder 2007). A more general process, which involves transforming feature vectors into a new scaled basis, is called *whitening* and is discussed in more detail in the context of eigenface-based face recognition (Section 14.2.1).

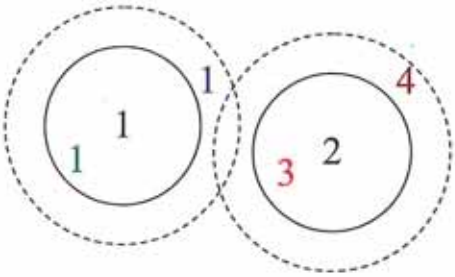


Figure 4.22 False positives and negatives: The black digits 1 and 2 are features being matched against a database of features in other images. At the current threshold setting (the solid circles), the green 1 is a *true positive* (good match), the blue 1 is a *false negative* (failure to match), and the red 3 is a *false positive* (incorrect match). If we set the threshold higher (the dashed circles), the blue 1 becomes a true positive but the brown 4 becomes an additional false positive.

	True matches	True non-matches		
Predicted matches	TP = 18	FP = 4	P' = 22	PPV = 0.82
Predicted non-matches	FN = 2	TN = 76	N' = 78	
	P = 20	N = 80	Total = 100	
	TPR = 0.90	FPR = 0.05		ACC = 0.94

Table 4.1 The number of matches correctly and incorrectly estimated by a feature matching algorithm, showing the number of true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN). The columns sum up to the actual number of positives (P) and negatives (N), while the rows sum up to the predicted number of positives (P') and negatives (N'). The formulas for the true positive rate (TPR), the false positive rate (FPR), the positive predictive value (PPV), and the accuracy (ACC) are given in the text.

Given a Euclidean distance metric, the simplest matching strategy is to set a threshold (maximum distance) and to return all matches from other images within this threshold. Setting the threshold too high results in too many *false positives*, i.e., incorrect matches being returned. Setting the threshold too low results in too many *false negatives*, i.e., too many correct matches being missed (Figure 4.22).

We can quantify the performance of a matching algorithm at a particular threshold by first counting the number of true and false matches and match failures, using the following definitions (Fawcett 2006):

- TP: true positives, i.e., number of correct matches;
- FN: false negatives, matches that were not correctly detected;
- FP: false positives, proposed matches that are incorrect;
- TN: true negatives, non-matches that were correctly rejected.

Table 4.1 shows a sample *confusion matrix* (contingency table) containing such numbers.

We can convert these numbers into *unit rates* by defining the following quantities (Fawcett 2006):

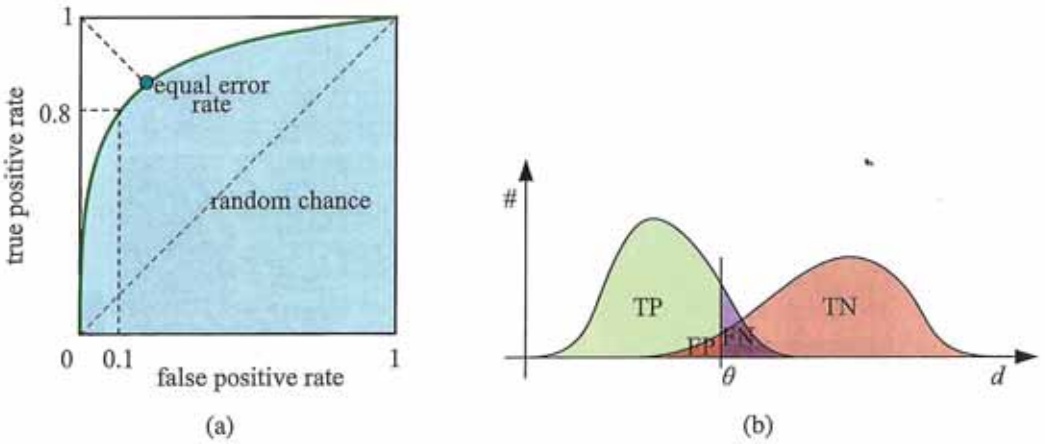


Figure 4.23 ROC curve and its related rates: (a) The ROC curve plots the true positive rate against the false positive rate for a particular combination of feature extraction and matching algorithms. Ideally, the true positive rate should be close to 1, while the false positive rate is close to 0. The area under the ROC curve (AUC) is often used as a single (scalar) measure of algorithm performance. Alternatively, the equal error rate is sometimes used. (b) The distribution of positives (matches) and negatives (non-matches) as a function of inter-feature distance d. As the threshold θ is increased, the number of true positives (TP) and false positives (FP) increases.

- true positive rate (TPR),

$$\text{TPR} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{P}}; \tag{4.14}$$

- false positive rate (FPR),

$$\text{FPR} = \frac{\text{FP}}{\text{FP+TN}} = \frac{\text{FP}}{\text{N}}; \tag{4.15}$$

- positive predictive value (PPV),

$$\text{PPV} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{P'}}; \tag{4.16}$$

- accuracy (ACC),

$$\text{ACC} = \frac{\text{TP+TN}}{\text{P+N}}. \tag{4.17}$$

In the *information retrieval* (or document retrieval) literature (Baeza-Yates and Ribeiro-Neto 1999; Manning, Raghavan, and Schütze 2008), the term *precision* (how many returned documents are relevant) is used instead of PPV and *recall* (what fraction of relevant documents was found) is used instead of TPR.

Any particular matching strategy (at a particular threshold or parameter setting) can be rated by the TPR and FPR numbers; ideally, the true positive rate will be close to 1 and the false positive rate close to 0. As we vary the matching threshold, we obtain a family of such points, which are collectively known as the *receiver operating characteristic (ROC curve)* (Fawcett 2006) (Figure 4.23a). The closer this curve lies to the upper left corner, i.e., the larger the area under the curve (AUC), the better its performance. Figure 4.23b shows how we can plot the number of matches and non-matches as a function of inter-feature distance d.

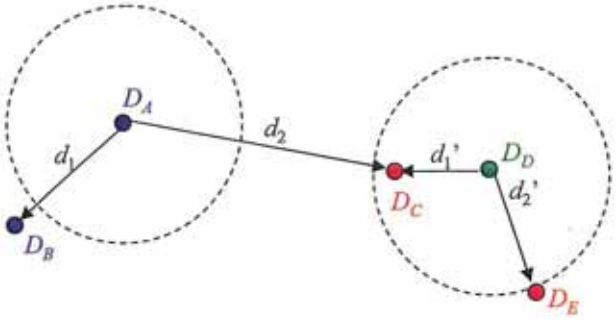


Figure 4.24 Fixed threshold, nearest neighbor, and nearest neighbor distance ratio matching. At a fixed distance threshold (dashed circles), descriptor D_A fails to match D_B and D_D incorrectly matches D_C and D_E. If we pick the nearest neighbor, D_A correctly matches D_B but D_D incorrectly matches D_C. Using nearest neighbor distance ratio (NNDR) matching, the small NNDR d_1/d_2 correctly matches D_A with D_B, and the large NNDR d_1'/d_2' correctly rejects matches for D_D.

These curves can then be used to plot an ROC curve (Exercise 4.3). The ROC curve can also be used to calculate the *mean average precision*, which is the average precision (PPV) as you vary the threshold to select the best results, then the two top results, etc.

The problem with using a fixed threshold is that it is difficult to set; the useful range of thresholds can vary a lot as we move to different parts of the feature space (Lowe 2004; Mikolajczyk and Schmid 2005). A better strategy in such cases is to simply match the *nearest neighbor* in feature space. Since some features may have no matches (e.g., they may be part of background clutter in object recognition or they may be occluded in the other image), a threshold is still used to reduce the number of false positives.

Ideally, this threshold itself will adapt to different regions of the feature space. If sufficient training data is available (Hua, Brown, and Winder 2007), it is sometimes possible to learn different thresholds for different features. Often, however, we are simply given a collection of images to match, e.g., when stitching images or constructing 3D models from unordered photo collections (Brown and Lowe 2007, 2003; Snavely, Seitz, and Szeliski 2006). In this case, a useful heuristic can be to compare the nearest neighbor distance to that of the second nearest neighbor, preferably taken from an image that is known not to match the target (e.g., a different object in the database) (Brown and Lowe 2002; Lowe 2004). We can define this *nearest neighbor distance ratio* (Mikolajczyk and Schmid 2005) as

$$\text{NNDR} = \frac{d_1}{d_2} = \frac{\|D_A - D_B\|}{\|D_A - D_C\|}, \tag{4.18}$$

where d_1 and d_2 are the nearest and second nearest neighbor distances, D_A is the target descriptor, and D_B and D_C are its closest two neighbors (Figure 4.24).

The effects of using these three different matching strategies for the feature descriptors evaluated by Mikolajczyk and Schmid (2005) are shown in Figure 4.25. As you can see, the nearest neighbor and NNDR strategies produce improved ROC curves.

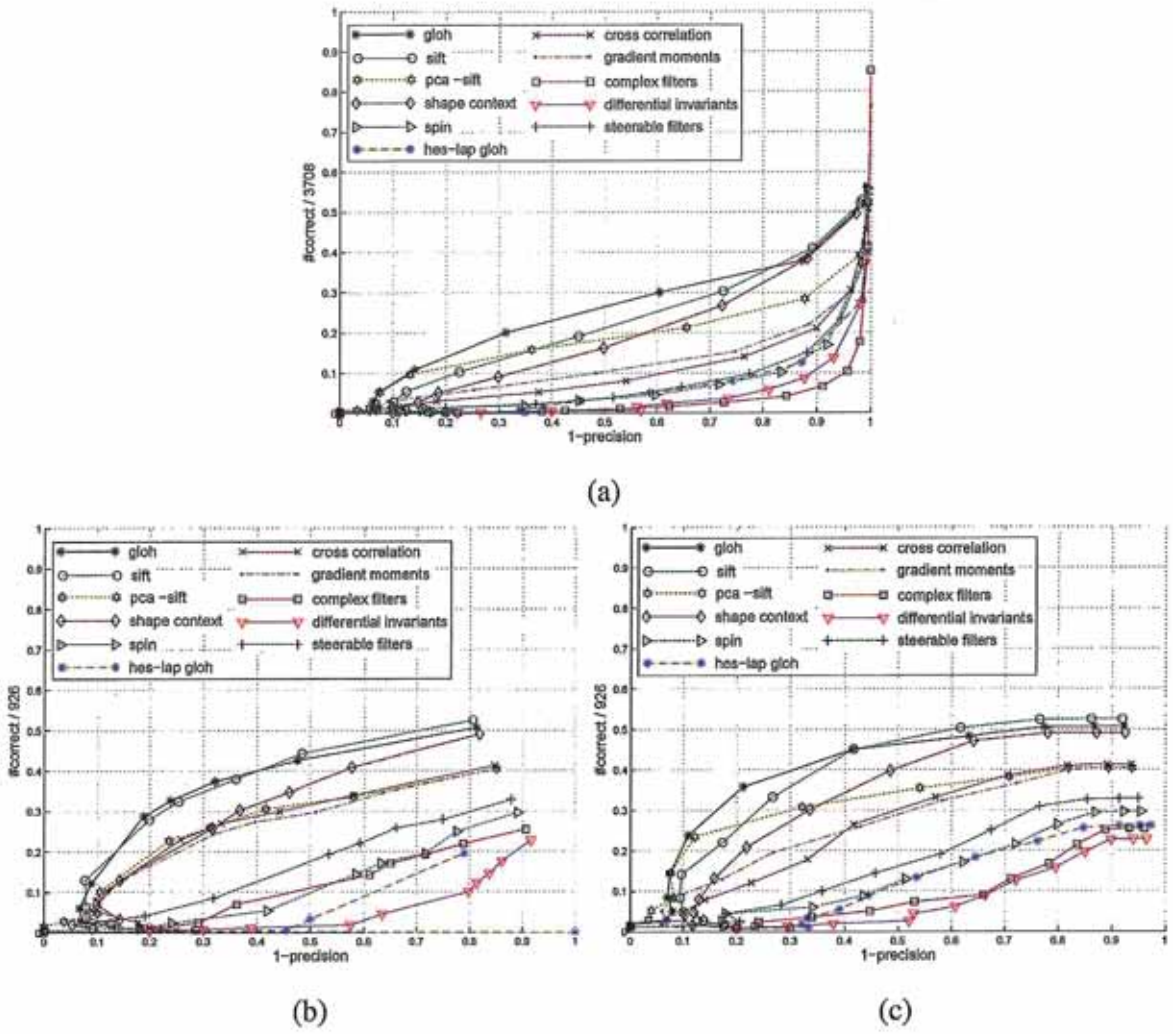


Figure 4.25 Performance of the feature descriptors evaluated by Mikolajczyk and Schmid (2005) © 2005 IEEE, shown for three matching strategies: (a) fixed threshold; (b) nearest neighbor; (c) nearest neighbor distance ratio (NNDR). Note how the ordering of the algorithms does not change that much, but the overall performance varies significantly between the different matching strategies.

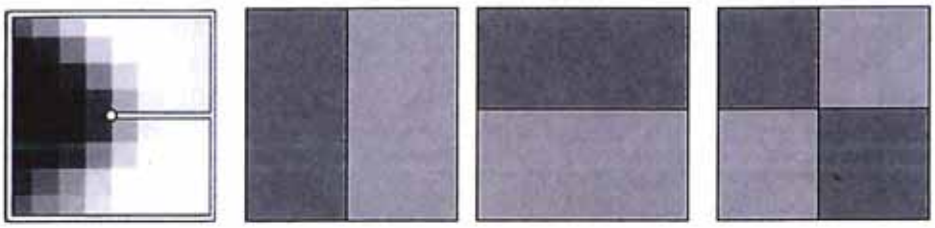


Figure 4.26 The three Haar wavelet coefficients used for hashing the MOPS descriptor devised by Brown, Szeliski, and Winder (2005) are computed by summing each 8×8 normalized patch over the light and dark gray regions and taking their difference.

Efficient matching

Once we have decided on a matching strategy, we still need to search efficiently for potential candidates. The simplest way to find all corresponding feature points is to compare all features against all other features in each pair of potentially matching images. Unfortunately, this is quadratic in the number of extracted features, which makes it impractical for most applications.

A better approach is to devise an *indexing structure*, such as a multi-dimensional search tree or a hash table, to rapidly search for features near a given feature. Such indexing structures can either be built for each image independently (which is useful if we want to only consider certain potential matches, e.g., searching for a particular object) or globally for all the images in a given database, which can potentially be faster, since it removes the need to iterate over each image. For extremely large databases (millions of images or more), even more efficient structures based on ideas from document retrieval (e.g., *vocabulary trees*, (Nistér and Stewénius 2006)) can be used (Section 14.3.2).

One of the simpler techniques to implement is multi-dimensional hashing, which maps descriptors into fixed size buckets based on some function applied to each descriptor vector. At matching time, each new feature is hashed into a bucket, and a search of nearby buckets is used to return potential candidates, which can then be sorted or graded to determine which are valid matches.

A simple example of hashing is the Haar wavelets used by Brown, Szeliski, and Winder (2005) in their MOPS paper. During the matching structure construction, each 8×8 scaled, oriented, and normalized MOPS patch is converted into a three-element index by performing sums over different quadrants of the patch (Figure 4.26). The resulting three values are normalized by their expected standard deviations and then mapped to the two (of $b = 10$) nearest 1D bins. The three-dimensional indices formed by concatenating the three quantized values are used to index the $2^3 = 8$ bins where the feature is stored (added). At query time, only the primary (closest) indices are used, so only a single three-dimensional bin needs to be examined. The coefficients in the bin can then be used to select k approximate nearest neighbors for further processing (such as computing the NNDR).

A more complex, but more widely applicable, version of hashing is called *locality sensitive hashing*, which uses unions of independently computed hashing functions to index the features (Gionis, Indyk, and Motwani 1999; Shakhnarovich, Darrell, and Indyk 2006). Shakhnarovich, Viola, and Darrell (2003) extend this technique to be more sensitive to the

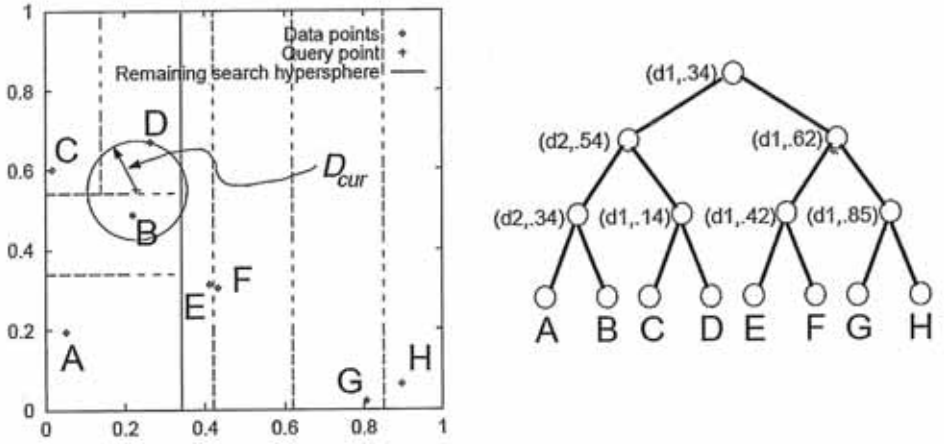


Figure 4.27 K-d tree and best bin first (BBF) search (Beis and Lowe 1999) © 1999 IEEE: (a) The spatial arrangement of the axis-aligned cutting planes is shown using dashed lines. Individual data points are shown as small diamonds. (b) The same subdivision can be represented as a tree, where each interior node represents an axis-aligned cutting plane (e.g., the top node cuts along dimension d1 at value .34) and each leaf node is a data point. During a BBF search, a query point (denoted by "+") first looks in its containing bin (D) and then in its nearest adjacent bin (B), rather than its closest neighbor in the tree (C).

distribution of points in parameter space, which they call *parameter-sensitive hashing*. Even more recent work converts high-dimensional descriptor vectors into binary codes that can be compared using Hamming distances (Torralba, Weiss, and Fergus 2008; Weiss, Torralba, and Fergus 2008) or that can accommodate arbitrary kernel functions (Kulis and Grauman 2009; Raginsky and Lazebnik 2009).

Another widely used class of indexing structures are multi-dimensional search trees. The best known of these are *k-d trees*, also often written as *k*d-trees, which divide the multi-dimensional feature space along alternating axis-aligned hyperplanes, choosing the threshold along each axis so as to maximize some criterion, such as the search tree balance (Samet 1989). Figure 4.27 shows an example of a two-dimensional k-d tree. Here, eight different data points A–H are shown as small diamonds arranged on a two-dimensional plane. The k-d tree recursively splits this plane along axis-aligned (horizontal or vertical) cutting planes. Each split can be denoted using the dimension number and split value (Figure 4.27b). The splits are arranged so as to try to balance the tree, i.e., to keep its maximum depth as small as possible. At query time, a classic k-d tree search first locates the query point (+) in its appropriate bin (D), and then searches nearby leaves in the tree (C, B, ...) until it can guarantee that the nearest neighbor has been found. The best bin first (BBF) search (Beis and Lowe 1999) searches bins in order of their spatial proximity to the query point and is therefore usually more efficient.

Many additional data structures have been developed over the years for solving nearest neighbor problems (Arya, Mount, Netanyahu *et al.* 1998; Liang, Liu, Xu *et al.* 2001; Hjaltason and Samet 2003). For example, Nene and Nayar (1997) developed a technique they call

slicing that uses a series of 1D binary searches on the point list sorted along different dimensions to efficiently cull down a list of candidate points that lie within a hypercube of the query point. Grauman and Darrell (2005) reweight the matches at different levels of an indexing tree, which allows their technique to be less sensitive to discretization errors in the tree construction. Nistér and Stewénius (2006) use a *metric tree*, which compares feature descriptors to a small number of prototypes at each level in a hierarchy. The resulting quantized *visual words* can then be used with classical information retrieval (document relevance) techniques to quickly winnow down a set of potential candidates from a database of millions of images (Section 14.3.2). Muja and Lowe (2009) compare a number of these approaches, introduce a new one of their own (priority search on hierarchical k-means trees), and conclude that multiple randomized k-d trees often provide the best performance. Despite all of this promising work, the rapid computation of image feature correspondences remains a challenging open research problem.

Feature match verification and densification

Once we have some hypothetical (putative) matches, we can often use geometric alignment (Section 6.1) to verify which matches are *inliers* and which ones are *outliers*. For example, if we expect the whole image to be translated or rotated in the matching view, we can fit a global geometric transform and keep only those feature matches that are sufficiently close to this estimated transformation. The process of selecting a small set of seed matches and then verifying a larger set is often called *random sampling* or RANSAC (Section 6.1.4). Once an initial set of correspondences has been established, some systems look for additional matches, e.g., by looking for additional correspondences along epipolar lines (Section 11.1) or in the vicinity of estimated locations based on the global transform. These topics are discussed further in Sections 6.1, 11.2, and 14.3.1.

4.1.4 Feature tracking

An alternative to independently finding features in all candidate images and then matching them is to find a set of likely feature locations in a first image and to then *search* for their corresponding locations in subsequent images. This kind of *detect then track* approach is more widely used for video tracking applications, where the expected amount of motion and appearance deformation between adjacent frames is expected to be small.

The process of selecting good features to track is closely related to selecting good features for more general recognition applications. In practice, regions containing high gradients in both directions, i.e., which have high eigenvalues in the auto-correlation matrix (4.8), provide stable locations at which to find correspondences (Shi and Tomasi 1994).

In subsequent frames, searching for locations where the corresponding patch has low squared difference (4.1) often works well enough. However, if the images are undergoing brightness change, explicitly compensating for such variations (8.9) or using *normalized cross-correlation* (8.11) may be preferable. If the search range is large, it is also often more efficient to use a *hierarchical* search strategy, which uses matches in lower-resolution images to provide better initial guesses and hence speed up the search (Section 8.1.1). Alternatives to this strategy involve learning what the appearance of the patch being tracked should be and then searching for it in the vicinity of its predicted position (Avidan 2001; Jurie and Dhome

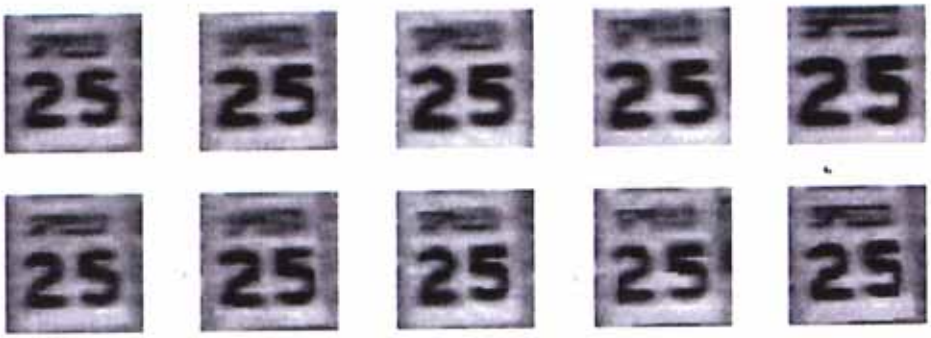


Figure 4.28 Feature tracking using an affine motion model (Shi and Tomasi 1994) © 1994 IEEE, Top row: image patch around the tracked feature location. Bottom row: image patch after warping back toward the first frame using an affine deformation. Even though the speed sign gets larger from frame to frame, the affine transformation maintains a good resemblance between the original and subsequent tracked frames.

2002; Williams, Blake, and Cipolla 2003). These topics are all covered in more detail in Section 8.1.3.

If features are being tracked over longer image sequences, their appearance can undergo larger changes. You then have to decide whether to continue matching against the originally detected patch (feature) or to re-sample each subsequent frame at the matching location. The former strategy is prone to failure as the original patch can undergo appearance changes such as foreshortening. The latter runs the risk of the feature drifting from its original location to some other location in the image (Shi and Tomasi 1994). (Mathematically, small mis-registration errors compound to create a *Markov Random Walk*, which leads to larger drift over time.)

A preferable solution is to compare the original patch to later image locations using an *affine* motion model (Section 8.2). Shi and Tomasi (1994) first compare patches in neighboring frames using a translational model and then use the location estimates produced by this step to initialize an affine registration between the patch in the current frame and the base frame where a feature was first detected (Figure 4.28). In their system, features are only detected infrequently, i.e., only in regions where tracking has failed. In the usual case, an area around the current *predicted* location of the feature is searched with an incremental registration algorithm (Section 8.1.3). The resulting tracker is often called the Kanade–Lucas–Tomasi (KLT) tracker.

Since their original work on feature tracking, Shi and Tomasi's approach has generated a string of interesting follow-on papers and applications. Beardsley, Torr, and Zisserman (1996) use extended feature tracking combined with structure from motion (Chapter 7) to incrementally build up sparse 3D models from video sequences. Kang, Szeliski, and Shum (1997) tie together the corners of adjacent (regularly gridded) patches to provide some additional stability to the tracking, at the cost of poorer handling of occlusions. Tommasini, Fusiello, Trucco *et al.* (1998) provide a better spurious match rejection criterion for the basic Shi and Tomasi algorithm, Collins and Liu (2003) provide improved mechanisms for feature selection and dealing with larger appearance changes over time, and Shafique and Shah (2005) develop algorithms for feature matching (data association) for videos with large numbers of

Figure 4.29 Real-time head tracking using the fast trained classifiers of Lepetit, Pilet, and Fua (2004) © 2004 IEEE.

moving objects or points. Yilmaz, Javed, and Shah (2006) and Lepetit and Fua (2005) survey the larger field of object tracking, which includes not only feature-based techniques but also alternative techniques based on contour and region (Section 5.1).

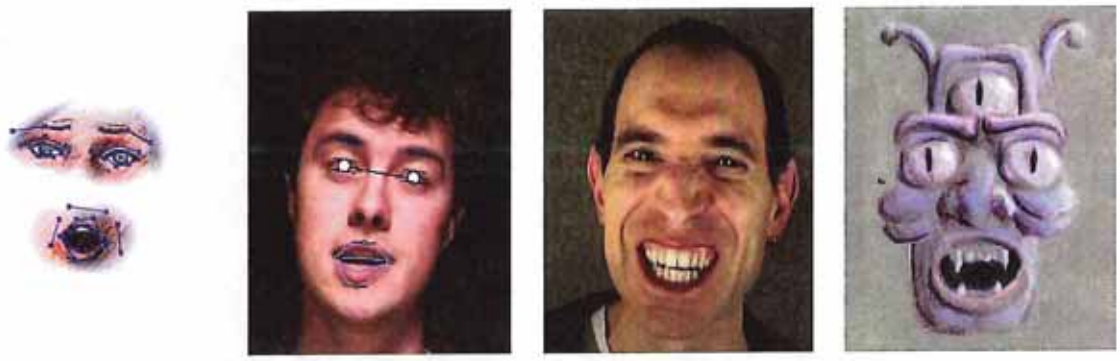
One of the newest developments in feature tracking is the use of learning algorithms to build special-purpose recognizers to rapidly search for matching features anywhere in an image (Lepetit, Pilet, and Fua 2006; Hinterstoisser, Benhimane, Navab *et al.* 2008; Rogez, Rihan, Ramalingam *et al.* 2008; Özuysal, Calonder, Lepetit *et al.* 2010).[2] By taking the time to train classifiers on sample patches and their affine deformations, extremely fast and reliable feature detectors can be constructed, which enables much faster motions to be supported (Figure 4.29). Coupling such features to deformable models (Pilet, Lepetit, and Fua 2008) or structure-from-motion algorithms (Klein and Murray 2008) can result in even higher stability.

4.1.5 *Application*: Performance-driven animation

One of the most compelling applications of fast feature tracking is *performance-driven animation*, i.e., the interactive deformation of a 3D graphics model based on tracking a user's motions (Williams 1990; Litwinowicz and Williams 1994; Lepetit, Pilet, and Fua 2004).

Buck, Finkelstein, Jacobs *et al.* (2000) present a system that tracks a user's facial expressions and head motions and then uses them to morph among a series of hand-drawn sketches. An animator first extracts the eye and mouth regions of each sketch and draws control lines over each image (Figure 4.30a). At run time, a face-tracking system (Toyama 1998) determines the current location of these features (Figure 4.30b). The animation system decides

[2] See also my previous comment on earlier work in learning-based tracking (Avidan 2001; Jurie and Dhome 2002; Williams, Blake, and Cipolla 2003).



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

Figure 4.30 Performance-driven, hand-drawn animation (Buck, Finkelstein, Jacobs *et al.* 2000) © 2000 ACM: (a) eye and mouth portions of hand-drawn sketch with their overlaid control lines; (b) an input video frame with the tracked features overlaid; (c) a different input video frame along with its (d) corresponding hand-drawn animation.

which input images to morph based on nearest neighbor feature appearance matching and triangular barycentric interpolation. It also computes the global location and orientation of the head from the tracked features. The resulting morphed eye and mouth regions are then composited back into the overall head model to yield a frame of hand-drawn animation (Figure 4.30d).

In more recent work, Barnes, Jacobs, Sanders *et al.* (2008) watch users animate paper cutouts on a desk and then turn the resulting motions and drawings into seamless 2D animations.

4.2 Edges

While interest points are useful for finding image locations that can be accurately matched in 2D, edge points are far more plentiful and often carry important semantic associations. For example, the boundaries of objects, which also correspond to occlusion events in 3D, are usually delineated by visible contours. Other kinds of edges correspond to shadow boundaries or crease edges, where surface orientation changes rapidly. Isolated edge points can also be grouped into longer *curves* or *contours*, as well as *straight line segments* (Section 4.3). It is interesting that even young children have no difficulty in recognizing familiar objects or animals from such simple line drawings.

4.2.1 Edge detection

Given an image, how can we find the salient edges? Consider the color images in Figure 4.31. If someone asked you to point out the most "salient" or "strongest" edges or the object boundaries (Martin, Fowlkes, and Malik 2004; Arbeláez, Maire, Fowlkes *et al.* 2010), which ones would you trace? How closely do your perceptions match the edge images shown in Figure 4.31?

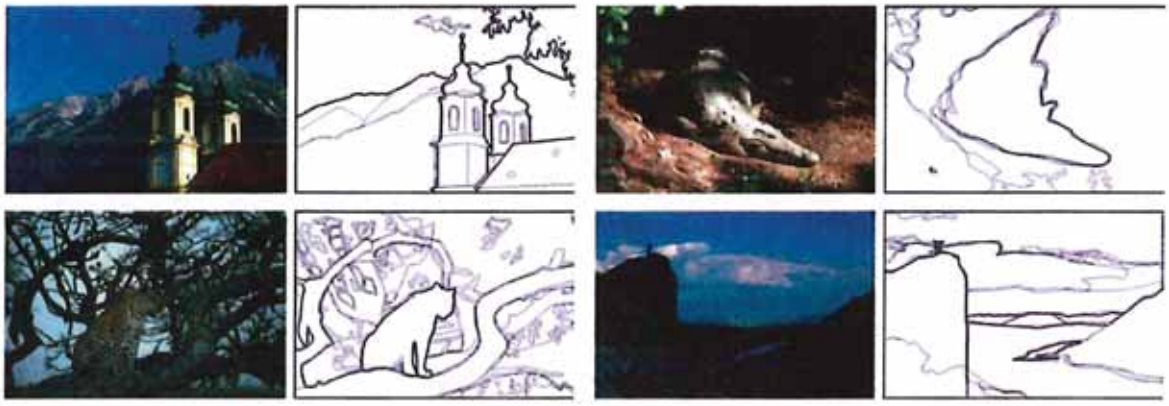


Figure 4.31 Human boundary detection (Martin, Fowlkes, and Malik 2004) © 2004 IEEE. The darkness of the edges corresponds to how many human subjects marked an object boundary at that location.

Qualitatively, edges occur at boundaries between regions of different color, intensity, or texture. Unfortunately, segmenting an image into coherent regions is a difficult task, which we address in Chapter 5. Often, it is preferable to detect edges using only purely local information.

Under such conditions, a reasonable approach is to define an edge as a location of *rapid intensity variation*.[3] Think of an image as a height field. On such a surface, edges occur at locations of *steep slopes*, or equivalently, in regions of closely packed contour lines (on a topographic map).

A mathematical way to define the slope and direction of a surface is through its gradient,

$$\boldsymbol{J}(\boldsymbol{x}) = \nabla I(\boldsymbol{x}) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})(\boldsymbol{x}). \tag{4.19}$$

The local gradient vector $\boldsymbol{J}$ points in the direction of *steepest ascent* in the intensity function. Its magnitude is an indication of the slope or strength of the variation, while its orientation points in a direction *perpendicular* to the local contour.

Unfortunately, taking image derivatives accentuates high frequencies and hence amplifies noise, since the proportion of noise to signal is larger at high frequencies. It is therefore prudent to smooth the image with a low-pass filter prior to computing the gradient. Because we would like the response of our edge detector to be independent of orientation, a circularly symmetric smoothing filter is desirable. As we saw in Section 3.2, the Gaussian is the only separable circularly symmetric filter and so it is used in most edge detection algorithms. Canny (1986) discusses alternative filters and a number of researcher review alternative edge detection algorithms and compare their performance (Davis 1975; Nalwa and Binford 1986; Nalwa 1987; Deriche 1987; Freeman and Adelson 1991; Nalwa 1993; Heath, Sarkar, Sanocki *et al.* 1998; Crane 1997; Ritter and Wilson 2000; Bowyer, Kranenburg, and Dougherty 2001; Arbeláez, Maire, Fowlkes *et al.* 2010).

Because differentiation is a linear operation, it commutes with other linear filtering oper-

[3] We defer the topic of edge detection in color images.

ations. The gradient of the smoothed image can therefore be written as

$$\boldsymbol{J}_\sigma(\boldsymbol{x}) = \nabla[G_\sigma(\boldsymbol{x}) * I(\boldsymbol{x})] = [\nabla G_\sigma](\boldsymbol{x}) * I(\boldsymbol{x}), \tag{4.20}$$

i.e., we can convolve the image with the horizontal and vertical derivatives of the Gaussian kernel function,

$$\nabla G_\sigma(\boldsymbol{x}) = (\frac{\partial G_\sigma}{\partial x}, \frac{\partial G_\sigma}{\partial y})(\boldsymbol{x}) = [-x \ -y]\frac{1}{\sigma^3}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{4.21}$$

(The parameter σ indicates the width of the Gaussian.) This is the same computation that is performed by Freeman and Adelson's (1991) first-order steerable filter, which we already covered in Section 3.2.3.

For many applications, however, we wish to thin such a continuous gradient image to only return isolated edges, i.e., as single pixels at discrete locations along the edge contours. This can be achieved by looking for *maxima* in the edge strength (gradient magnitude) in a direction *perpendicular* to the edge orientation, i.e., along the gradient direction.

Finding this maximum corresponds to taking a directional derivative of the strength field in the direction of the gradient and then looking for zero crossings. The desired directional derivative is equivalent to the dot product between a second gradient operator and the results of the first,

$$S_\sigma(\boldsymbol{x}) = \nabla \cdot \boldsymbol{J}_\sigma(\boldsymbol{x}) = [\nabla^2 G_\sigma](\boldsymbol{x}) * I(\boldsymbol{x})]. \tag{4.22}$$

The gradient operator dot product with the gradient is called the *Laplacian*. The convolution kernel

$$\nabla^2 G_\sigma(\boldsymbol{x}) = \frac{1}{\sigma^3}\left(2 - \frac{x^2+y^2}{2\sigma^2}\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{4.23}$$

is therefore called the *Laplacian of Gaussian* (LoG) kernel (Marr and Hildreth 1980). This kernel can be split into two separable parts,

$$\nabla^2 G_\sigma(\boldsymbol{x}) = \frac{1}{\sigma^3}\left(1 - \frac{x^2}{2\sigma^2}\right)G_\sigma(x)G_\sigma(y) + \frac{1}{\sigma^3}\left(1 - \frac{y^2}{2\sigma^2}\right)G_\sigma(y)G_\sigma(x) \tag{4.24}$$

(Wiejak, Buxton, and Buxton 1985), which allows for a much more efficient implementation using separable filtering (Section 3.2.1).

In practice, it is quite common to replace the Laplacian of Gaussian convolution with a Difference of Gaussian (DoG) computation, since the kernel shapes are qualitatively similar (Figure 3.35). This is especially convenient if a "Laplacian pyramid" (Section 3.5) has already been computed.[4]

In fact, it is not strictly necessary to take differences between adjacent levels when computing the edge field. Think about what a zero crossing in a "generalized" difference of Gaussians image represents. The finer (smaller kernel) Gaussian is a noise-reduced version of the original image. The coarser (larger kernel) Gaussian is an estimate of the average intensity over a larger region. Thus, whenever the DoG image changes sign, this corresponds to the (slightly blurred) image going from relatively darker to relatively lighter, as compared to the average intensity in that neighborhood.

[4] Recall that Burt and Adelson's (1983a) "Laplacian pyramid" actually computed differences of Gaussian-filtered levels.

Once we have computed the sign function $S(\boldsymbol{x})$, we must find its *zero crossings* and convert these into edge elements (*edgels*). An easy way to detect and represent zero crossings is to look for adjacent pixel locations $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ where the sign changes value, i.e., $[S(\boldsymbol{x}_i) > 0] \neq [S(\boldsymbol{x}_j) > 0]$.

The sub-pixel location of this crossing can be obtained by computing the "x-intercept" of the "line" connecting $S(\boldsymbol{x}_i)$ and $S(\boldsymbol{x}_j)$,

$$\boldsymbol{x}_z = \frac{\boldsymbol{x}_i S(\boldsymbol{x}_j) - \boldsymbol{x}_j S(\boldsymbol{x}_i)}{S(\boldsymbol{x}_j) - S(\boldsymbol{x}_i)}. \tag{4.25}$$

The orientation and strength of such edgels can be obtained by linearly interpolating the gradient values computed on the original pixel grid.

An alternative edgel representation can be obtained by linking adjacent edgels on the dual grid to form edgels that live *inside* each square formed by four adjacent pixels in the original pixel grid.[5] The (potential) advantage of this representation is that the edgels now live on a grid offset by half a pixel from the original pixel grid and are thus easier to store and access. As before, the orientations and strengths of the edges can be computed by interpolating the gradient field or estimating these values from the difference of Gaussian image (see Exercise 4.7).

In applications where the accuracy of the edge orientation is more important, higher-order steerable filters can be used (Freeman and Adelson 1991) (see Section 3.2.3). Such filters are more selective for more elongated edges and also have the possibility of better modeling curve intersections because they can represent multiple orientations at the same pixel (Figure 3.16). Their disadvantage is that they are more expensive to compute and the directional derivative of the edge strength does not have a simple closed form solution.[6]

Scale selection and blur estimation

As we mentioned before, the derivative, Laplacian, and Difference of Gaussian filters (4.20–4.23) all require the selection of a spatial scale parameter σ. If we are only interested in detecting sharp edges, the width of the filter can be determined from image noise characteristics (Canny 1986; Elder and Zucker 1998). However, if we want to detect edges that occur at different resolutions (Figures 4.32b–c), a *scale-space* approach that detects and then selects edges at different scales may be necessary (Witkin 1983; Lindeberg 1994, 1998a; Nielsen, Florack, and Deriche 1997).

Elder and Zucker (1998) present a principled approach to solving this problem. Given a known image noise level, their technique computes, for every pixel, the minimum scale at which an edge can be reliably detected (Figure 4.32d). Their approach first computes gradients densely over an image by selecting among gradient estimates computed at different scales, based on their gradient magnitudes. It then performs a similar estimate of minimum scale for directed second derivatives and uses zero crossings of this latter quantity to robustly select edges (Figures 4.32e–f). As an optional final step, the blur width of each edge can be computed from the distance between extrema in the second derivative response minus the width of the Gaussian filter.

[5] This algorithm is a 2D version of the 3D *marching cubes* isosurface extraction algorithm (Lorensen and Cline 1987).

[6] In fact, the edge orientation can have a 180° ambiguity for "bar edges", which makes the computation of zero crossings in the derivative more tricky.

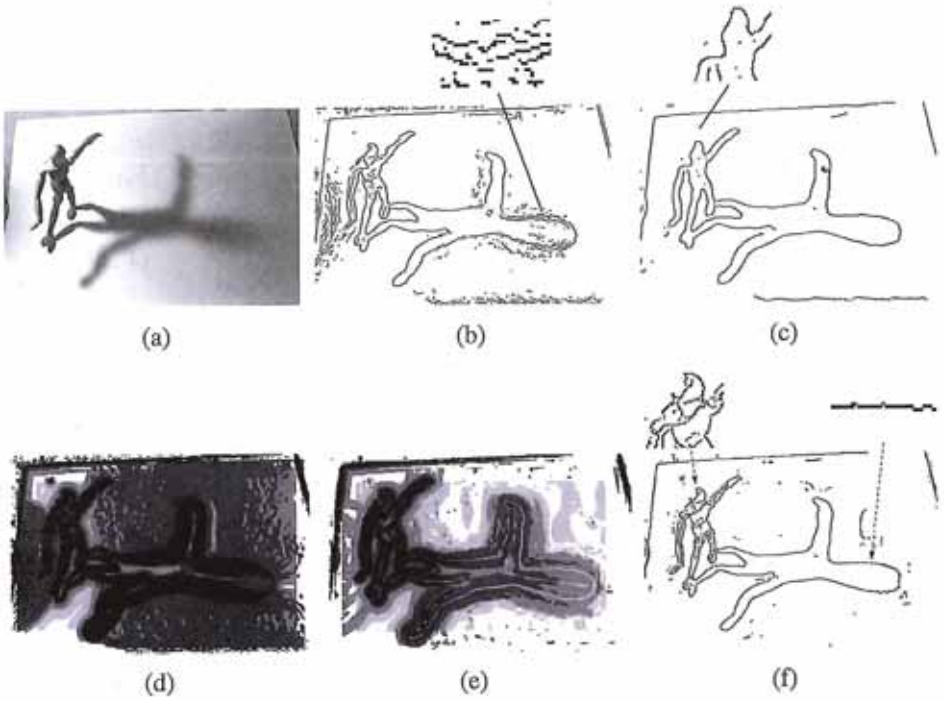


Figure 4.32 Scale selection for edge detection (Elder and Zucker 1998) © 1998 IEEE: (a) original image; (b–c) Canny/Deriche edge detector tuned to the finer (mannequin) and coarser (shadow) scales; (d) minimum reliable scale for gradient estimation; (e) minimum reliable scale for second derivative estimation; (f) final detected edges.

Color edge detection

While most edge detection techniques have been developed for grayscale images, color images can provide additional information. For example, noticeable edges between *iso-luminant* colors (colors that have the same luminance) are useful cues but fail to be detected by grayscale edge operators.

One simple approach is to combine the outputs of grayscale detectors run on each color band separately.[7] However, some care must be taken. For example, if we simply sum up the gradients in each of the color bands, the signed gradients may actually cancel each other! (Consider, for example a pure red-to-green edge.) We could also detect edges independently in each band and then take the union of these, but this might lead to thickened or doubled edges that are hard to link.

A better approach is to compute the *oriented energy* in each band (Morrone and Burr 1988; Perona and Malik 1990a), e.g., using a second-order steerable filter (Section 3.2.3) (Freeman and Adelson 1991), and then sum up the orientation-weighted energies and find their joint best orientation. Unfortunately, the directional derivative of this energy may not have a closed form solution (as in the case of signed first-order steerable filters), so a simple zero crossing-based strategy cannot be used. However, the technique described by Elder and

[7] Instead of using the raw RGB space, a more perceptually uniform color space such as L*a*b* (see Section 2.3.2) can be used instead. When trying to match human performance (Martin, Fowlkes, and Malik 2004), this makes sense. However, in terms of the physics of the underlying image formation and sensing, it may be a questionable strategy.

Zucker (1998) can be used to compute these zero crossings numerically instead.

An alternative approach is to estimate local color statistics in regions around each pixel (Ruzon and Tomasi 2001; Martin, Fowlkes, and Malik 2004). This has the advantage that more sophisticated techniques (e.g., 3D color histograms) can be used to compare regional statistics and that additional measures, such as texture, can also be considered. Figure 4.33 shows the output of such detectors.

Of course, many other approaches have been developed for detecting color edges, dating back to early work by Nevatia (1977). Ruzon and Tomasi (2001) and Gevers, van de Weijer, and Stokman (2006) provide good reviews of these approaches, which include ideas such as fusing outputs from multiple channels, using multidimensional gradients, and vector-based methods.

Combining edge feature cues

If the goal of edge detection is to match human *boundary detection* performance (Bowyer, Kranenburg, and Dougherty 2001; Martin, Fowlkes, and Malik 2004; Arbeláez, Maire, Fowlkes *et al.* 2010), as opposed to simply finding stable features for matching, even better detectors can be constructed by combining multiple low-level cues such as brightness, color, and texture.

Martin, Fowlkes, and Malik (2004) describe a system that combines brightness, color, and texture edges to produce state-of-the-art performance on a database of hand-segmented natural color images (Martin, Fowlkes, Tal *et al.* 2001). First, they construct and train[8] separate oriented half-disc detectors for measuring significant differences in brightness (luminance), color (a* and b* channels, summed responses), and texture (un-normalized filter bank responses from the work of Malik, Belongie, Leung *et al.* (2001)). Some of the responses are then sharpened using a soft non-maximal suppression technique. Finally, the outputs of the three detectors are combined using a variety of machine-learning techniques, from which logistic regression is found to have the best tradeoff between speed, space and accuracy . The resulting system (see Figure 4.33 for some examples) is shown to outperform previously developed techniques. Maire, Arbelaez, Fowlkes *et al.* (2008) improve on these results by combining the detector based on local appearance with a *spectral* (segmentation-based) detector (Belongie and Malik 1998). In more recent work, Arbeláez, Maire, Fowlkes *et al.* (2010) build a hierarchical segmentation on top of this edge detector using a variant of the watershed algorithm.

4.2.2 Edge linking

While isolated edges can be useful for a variety of applications, such as line detection (Section 4.3) and sparse stereo matching (Section 11.2), they become even more useful when linked into continuous contours.

If the edges have been detected using zero crossings of some function, linking them up is straightforward, since adjacent edgels share common endpoints. Linking the edgels into chains involves picking up an unlinked edgel and following its neighbors in both directions. Either a sorted list of edgels (sorted first by x coordinates and then by y coordinates, for example) or a 2D array can be used to accelerate the neighbor finding. If edges were not

[8] The training uses 200 labeled images and testing is performed on a different set of 100 images.

Figure 4.33 Combined brightness, color, texture boundary detector (Martin, Fowlkes, and Malik 2004) © 2004 IEEE. Successive rows show the outputs of the brightness gradient (BG), color gradient (CG), texture gradient (TG), and combined (BG+CG+TG) detectors. The final row shows human-labeled boundaries derived from a database of hand-segmented images (Martin, Fowlkes, Tal *et al.* 2001).

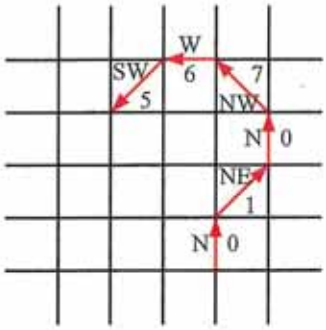


Figure 4.34 Chain code representation of a grid-aligned linked edge chain. The code is represented as a series of direction codes, e.g, 0 1 0 7 6 5, which can further be compressed using predictive and run-length coding.

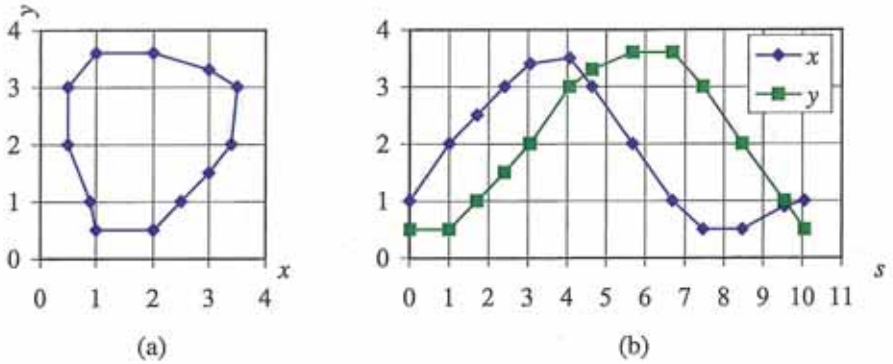


Figure 4.35 Arc-length parameterization of a contour: (a) discrete points along the contour are first transcribed as (b) (x, y) pairs along the arc length s. This curve can then be regularly re-sampled or converted into alternative (e.g., Fourier) representations.

detected using zero crossings, finding the continuation of an edgel can be tricky. In this case, comparing the orientation (and, optionally, phase) of adjacent edgels can be used for disambiguation. Ideas from connected component computation can also sometimes be used to make the edge linking process even faster (see Exercise 4.8).

Once the edgels have been linked into chains, we can apply an optional thresholding with hysteresis to remove low-strength contour segments (Canny 1986). The basic idea of hysteresis is to set two different thresholds and allow a curve being tracked above the higher threshold to dip in strength down to the lower threshold.

Linked edgel lists can be encoded more compactly using a variety of alternative representations. A *chain code* encodes a list of connected points lying on an $\mathcal{N}_8$ grid using a three-bit code corresponding to the eight cardinal directions (N, NE, E, SE, S, SW, W, NW) between a point and its successor (Figure 4.34). While this representation is more compact than the original edgel list (especially if predictive variable-length coding is used), it is not very suitable for further processing.

A more useful representation is the *arc length parameterization* of a contour, $\boldsymbol{x}(s)$, where s denotes the arc length along a curve. Consider the linked set of edgels shown in Fig-

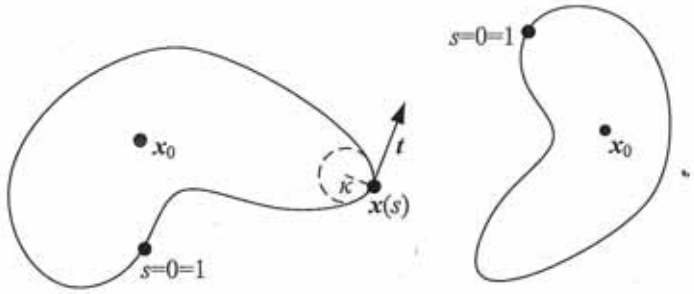


Figure 4.36 Matching two contours using their arc-length parameterization. If both curves are normalized to unit length, $s \in [0, 1]$ and centered around their centroid $\boldsymbol{x}_0$, they will have the same descriptor up to an overall "temporal" shift (due to different starting points for $s = 0$) and a phase (x-y) shift (due to rotation).

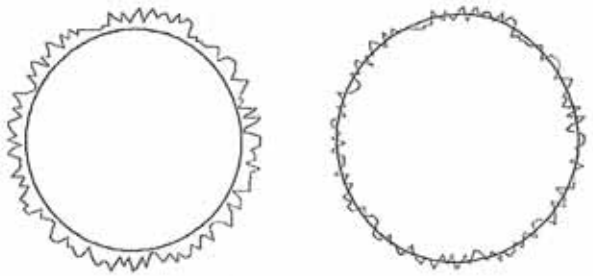


Figure 4.37 Curve smoothing with a Gaussian kernel (Lowe 1988) © 1998 IEEE: (a) without a shrinkage correction term; (b) with a shrinkage correction term.

ure 4.35a. We start at one point (the dot at $(1.0, 0.5)$ in Figure 4.35a) and plot it at coordinate $s = 0$ (Figure 4.35b). The next point at $(2.0, 0.5)$ gets plotted at $s = 1$, and the next point at $(2.5, 1.0)$ gets plotted at $s = 1.7071$, i.e., we increment s by the length of each edge segment. The resulting plot can be resampled on a regular (say, integral) s grid before further processing.

The advantage of the arc-length parameterization is that it makes matching and processing (e.g., smoothing) operations much easier. Consider the two curves describing similar shapes shown in Figure 4.36. To compare the curves, we first subtract the average values $\boldsymbol{x}_0 = \int_s \boldsymbol{x}(s)$ from each descriptor. Next, we rescale each descriptor so that s goes from 0 to 1 instead of 0 to S, i.e., we divide $\boldsymbol{x}(s)$ by S. Finally, we take the Fourier transform of each normalized descriptor, treating each $\boldsymbol{x} = (x, y)$ value as a complex number. If the original curves are the same (up to an unknown scale and rotation), the resulting Fourier transforms should differ only by a scale change in magnitude plus a constant complex phase shift, due to rotation, and a linear phase shift in the domain, due to different starting points for s (see Exercise 4.9).

Arc-length parameterization can also be used to smooth curves in order to remove digitization noise. However, if we just apply a regular smoothing filter, the curve tends to shrink

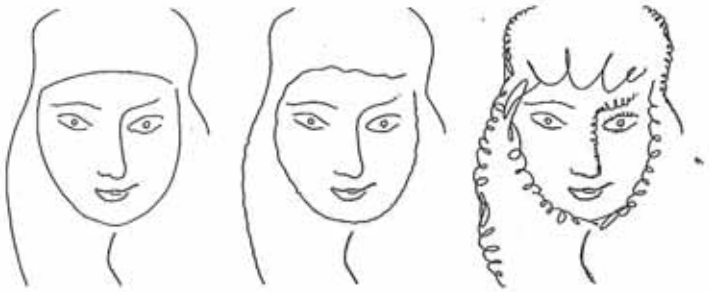


Figure 4.38 Changing the character of a curve without affecting its sweep (Finkelstein and Salesin 1994) © 1994 ACM: higher frequency wavelets can be replaced with exemplars from a style library to effect different local appearances.

on itself (Figure 4.37a). Lowe (1989) and Taubin (1995) describe techniques that compensate for this shrinkage by adding an offset term based on second derivative estimates or a larger smoothing kernel (Figure 4.37b). An alternative approach, based on selectively modifying different frequencies in a wavelet decomposition, is presented by Finkelstein and Salesin (1994). In addition to controlling shrinkage without affecting its "sweep", wavelets allow the "character" of a curve to be interactively modified, as shown in Figure 4.38.

The evolution of curves as they are smoothed and simplified is related to "grassfire" (distance) transforms and region skeletons (Section 3.3.3) (Tek and Kimia 2003), and can be used to recognize objects based on their contour shape (Sebastian and Kimia 2005). More local descriptors of curve shape such as *shape contexts* (Belongie, Malik, and Puzicha 2002) can also be used for recognition and are potentially more robust to missing parts due to occlusions.

The field of contour detection and linking continues to evolve rapidly and now includes techniques for global contour grouping, boundary completion, and junction detection (Maire, Arbelaez, Fowlkes *et al.* 2008), as well as grouping contours into likely regions (Arbeláez, Maire, Fowlkes *et al.* 2010) and wide-baseline correspondence (Meltzer and Soatto 2008).

4.2.3 *Application*: Edge editing and enhancement

While edges can serve as components for object recognition or features for matching, they can also be used directly for image editing.

In fact, if the edge magnitude and blur estimate are kept along with each edge, a visually similar image can be reconstructed from this information (Elder 1999). Based on this principle, Elder and Goldberg (2001) propose a system for "image editing in the contour domain". Their system allows users to selectively remove edges corresponding to unwanted features such as specularities, shadows, or distracting visual elements. After reconstructing the image from the remaining edges, the undesirable visual features have been removed (Figure 4.39).

Another potential application is to enhance perceptually salient edges while simplifying the underlying image to produce a cartoon-like or "pen-and-ink" stylized image (DeCarlo and Santella 2002). This application is discussed in more detail in Section 10.5.2.

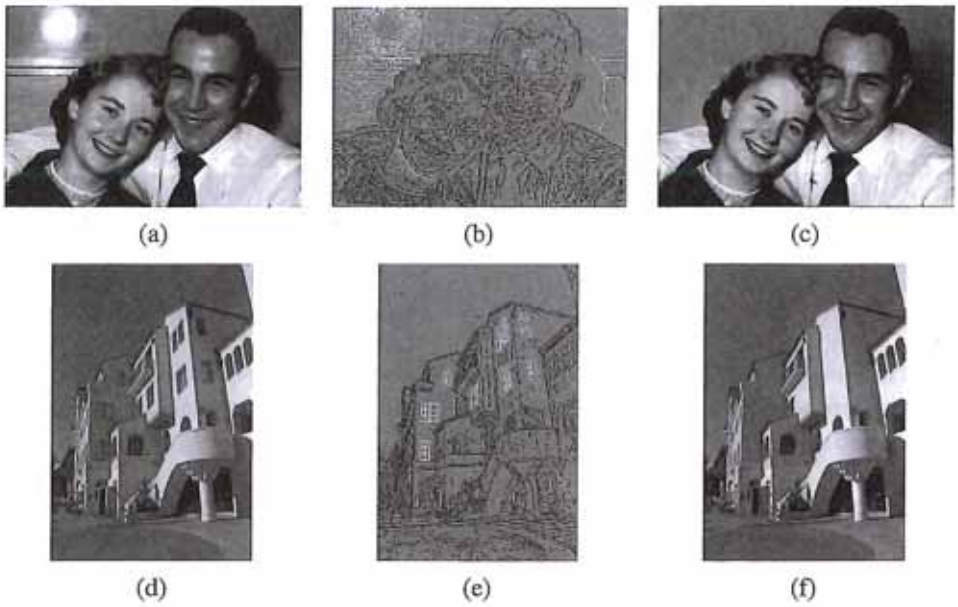


Figure 4.39 Image editing in the contour domain (Elder and Goldberg 2001) © 2001 IEEE: (a) and (d) original images; (b) and (e) extracted edges (edges to be deleted are marked in white); (c) and (f) reconstructed edited images.

4.3 Lines

While edges and general curves are suitable for describing the contours of natural objects, the man-made world is full of straight lines. Detecting and matching these lines can be useful in a variety of applications, including architectural modeling, pose estimation in urban environments, and the analysis of printed document layouts.

In this section, we present some techniques for extracting *piecewise linear* descriptions from the curves computed in the previous section. We begin with some algorithms for approximating a curve as a piecewise-linear polyline. We then describe the *Hough transform*, which can be used to group edgels into line segments even across gaps and occlusions. Finally, we describe how 3D lines with common *vanishing points* can be grouped together. These vanishing points can be used to calibrate a camera and to determine its orientation relative to a rectahedral scene, as described in Section 6.3.2.

4.3.1 Successive approximation

As we saw in Section 4.2.2, describing a curve as a series of 2D locations $\boldsymbol{x}_i = \boldsymbol{x}(s_i)$ provides a general representation suitable for matching and further processing. In many applications, however, it is preferable to approximate such a curve with a simpler representation, e.g., as a piecewise-linear polyline or as a B-spline curve (Farin 1996), as shown in Figure 4.40.

Many techniques have been developed over the years to perform this approximation, which is also known as *line simplification*. One of the oldest, and simplest, is the one proposed by Ramer (1972) and Douglas and Peucker (1973), who recursively subdivide the curve at

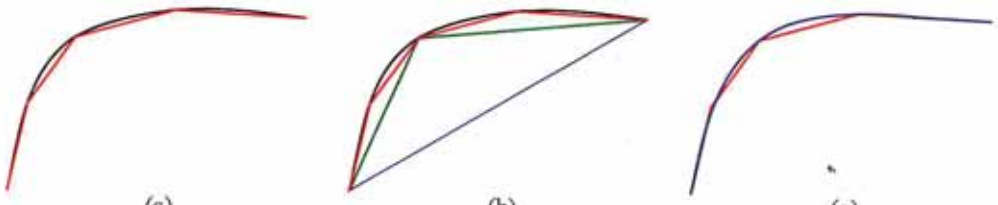


Figure 4.40 Approximating a curve (shown in black) as a polyline or B-spline: (a) original curve and a polyline approximation shown in red; (b) successive approximation by recursively finding points furthest away from the current approximation; (c) smooth interpolating spline, shown in dark blue, fit to the polyline vertices.

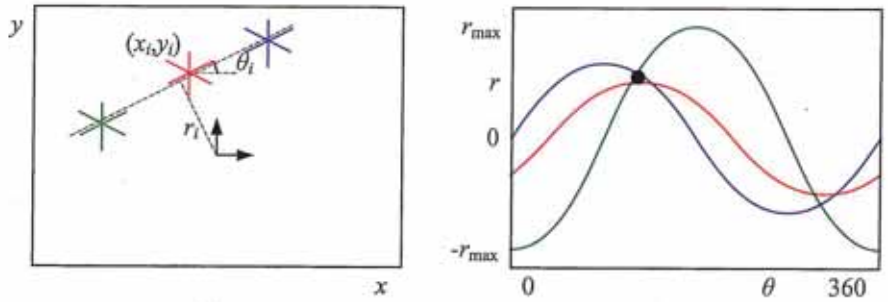


Figure 4.41 Original Hough transform: (a) each point votes for a complete family of potential lines $r_i(\theta) = x_i \cos\theta + y_i \sin\theta$; (b) each pencil of lines sweeps out a sinusoid in (r, θ); their intersection provides the desired line equation.

the point furthest away from the line joining the two endpoints (or the current coarse polyline approximation), as shown in Figure 4.40. Hershberger and Snoeyink (1992) provide a more efficient implementation and also cite some of the other related work in this area.

Once the line simplification has been computed, it can be used to approximate the original curve. If a smoother representation or visualization is desired, either approximating or interpolating splines or curves can be used (Sections 3.5.1 and 5.1.1) (Szeliski and Ito 1986; Bartels, Beatty, and Barsky 1987; Farin 1996), as shown in Figure 4.40c.

4.3.2 Hough transforms

While curve approximation with polylines can often lead to successful line extraction, lines in the real world are sometimes broken up into disconnected components or made up of many collinear line segments. In many cases, it is desirable to group such collinear segments into extended lines. At a further processing stage (described in Section 4.3.3), we can then group such lines into collections with common vanishing points.

The Hough transform, named after its original inventor (Hough 1962), is a well-known technique for having edges "vote" for plausible line locations (Duda and Hart 1972; Ballard 1981; Illingworth and Kittler 1988). In its original formulation (Figure 4.41), each edge point votes for *all* possible lines passing through it, and lines corresponding to high *accumulator* or

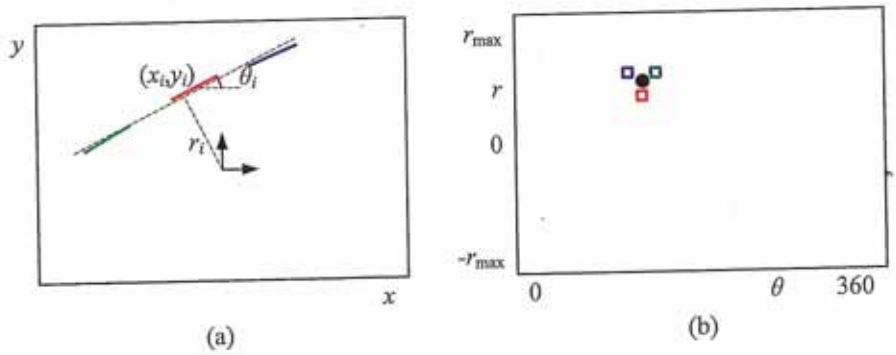


Figure 4.42 Oriented Hough transform: (a) an edgel re-parameterized in polar (r, θ) coordinates, with $\hat{\boldsymbol{n}}_i = (\cos\theta_i, \sin\theta_i)$ and $r_i = \hat{\boldsymbol{n}}_i \cdot \boldsymbol{x}_i$; (b) (r, θ) accumulator array, showing the votes for the three edgels marked in red, green, and blue.

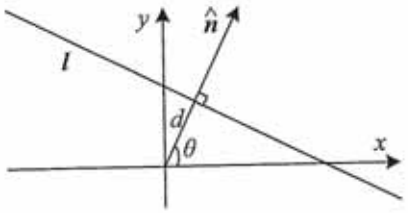


Figure 4.43 2D line equation expressed in terms of the normal $\hat{\boldsymbol{n}}$ and distance to the origin d.

bin values are examined for potential line fits.[9] Unless the points on a line are truly punctate, a better approach (in my experience) is to use the local orientation information at each edgel to vote for a *single* accumulator cell (Figure 4.42), as described below. A hybrid strategy, where each edgel votes for a number of possible orientation or location pairs centered around the estimate orientation, may be desirable in some cases.

Before we can vote for line hypotheses, we must first choose a suitable representation. Figure 4.43 (copied from Figure 2.2a) shows the normal-distance $(\hat{\boldsymbol{n}}, d)$ parameterization for a line. Since lines are made up of edge segments, we adopt the convention that the line normal $\hat{\boldsymbol{n}}$ points in the same direction (i.e., has the same sign) as the image gradient $\boldsymbol{J}(\boldsymbol{x}) = \nabla I(\boldsymbol{x})$ (4.19). To obtain a minimal two-parameter representation for lines, we convert the normal vector into an angle

$$\theta = \tan^{-1} n_y / n_x, \tag{4.26}$$

as shown in Figure 4.43. The range of possible (θ, d) values is $[-180°, 180°] \times [-\sqrt{2}, \sqrt{2}]$, assuming that we are using normalized pixel coordinates (2.61) that lie in $[-1, 1]$. The number of bins to use along each axis depends on the accuracy of the position and orientation estimate available at each edgel and the expected line density, and is best set experimentally with some test runs on sample imagery.

Given the line parameterization, the Hough transform proceeds as shown in Algorithm 4.2.

[9] The Hough transform can also be *generalized* to look for other geometric features such as circles (Ballard 1981), but we do not cover such extensions in this book.

procedure *Hough*$(\{(x, y, \theta)\})$:

1. Clear the accumulator array.
2. For each detected edgel at location (x, y) and orientation $\theta = \tan^{-1} n_y/n_x$, compute the value of
$$d = x\, n_x + y\, n_y$$
and increment the accumulator corresponding to (θ, d).
3. Find the peaks in the accumulator corresponding to lines.
4. Optionally re-fit the lines to the constituent edgels.

Algorithm 4.2 Outline of a Hough transform algorithm based on oriented edge segments.

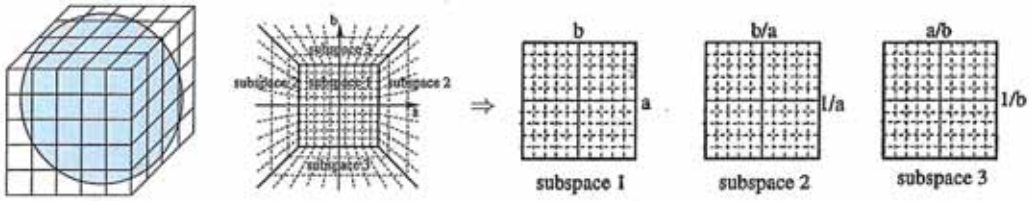


Figure 4.44 Cube map representation for line equations and vanishing points: (a) a cube map surrounding the unit sphere; (b) projecting the half-cube onto three subspaces (Tuytelaars, Van Gool, and Proesmans 1997) © 1997 IEEE.

Note that the original formulation of the Hough transform, which assumed no knowledge of the edgel orientation θ, has an additional loop inside Step 2 that iterates over all possible values of θ and increments a whole series of accumulators.

There are a lot of details in getting the Hough transform to work well, but these are best worked out by writing an implementation and testing it out on sample data. Exercise 4.12 describes some of these steps in more detail, including using edge segment lengths or strengths during the voting process, keeping a list of constituent edgels in the accumulator array for easier post-processing, and optionally combining edges of different "polarity" into the same line segments.

An alternative to the 2D polar (θ, d) representation for lines is to use the full 3D $\boldsymbol{m} = (\hat{\boldsymbol{n}}, d)$ line equation, projected onto the unit sphere. While the sphere can be parameterized using spherical coordinates (2.8),

$$\hat{\boldsymbol{m}} = (\cos\theta\cos\phi, \sin\theta\cos\phi, \sin\phi), \tag{4.27}$$

this does not uniformly sample the sphere and still requires the use of trigonometry.

An alternative representation can be obtained by using a *cube map*, i.e., projecting $\boldsymbol{m}$ onto the face of a unit cube (Figure 4.44a). To compute the cube map coordinate of a 3D vector $\boldsymbol{m}$, first find the largest (absolute value) component of $\boldsymbol{m}$, i.e., $m = \pm\max(|n_x|, |n_y|, |d|)$,

and use this to select one of the six cube faces. Divide the remaining two coordinates by m and use these as indices into the cube face. While this avoids the use of trigonometry, it does require some decision logic.

One advantage of using the cube map, first pointed out by Tuytelaars, Van Gool, and Proesmans (1997), is that all of the lines passing through a point correspond to line segments on the cube faces, which is useful if the original (full voting) variant of the Hough transform is being used. In their work, they represent the line equation as $ax + b + y = 0$, which does not treat the x and y axes symmetrically. Note that if we restrict $d \geq 0$ by ignoring the polarity of the edge orientation (gradient sign), we can use a half-cube instead, which can be represented using only three cube faces, as shown in Figure 4.44b (Tuytelaars, Van Gool, and Proesmans 1997).

RANSAC-based line detection. Another alternative to the Hough transform is the RANdom SAmple Consensus (RANSAC) algorithm described in more detail in Section 6.1.4. In brief, RANSAC randomly chooses pairs of edgels to form a line hypothesis and then tests how many other edgels fall onto this line. (If the edge orientations are accurate enough, a single edgel can produce this hypothesis.) Lines with sufficiently large numbers of *inliers* (matching edgels) are then selected as the desired line segments.

An advantage of RANSAC is that no accumulator array is needed and so the algorithm can be more space efficient and potentially less prone to the choice of bin size. The disadvantage is that many more hypotheses may need to be generated and tested than those obtained by finding peaks in the accumulator array.

In general, there is no clear consensus on which line estimation technique performs best. It is therefore a good idea to think carefully about the problem at hand and to implement several approaches (successive approximation, Hough, and RANSAC) to determine the one that works best for your application.

4.3.3 Vanishing points

In many scenes, structurally important lines have the same vanishing point because they are parallel in 3D. Examples of such lines are horizontal and vertical building edges, zebra crossings, railway tracks, the edges of furniture such as tables and dressers, and of course, the ubiquitous calibration pattern (Figure 4.45). Finding the vanishing points common to such line sets can help refine their position in the image and, in certain cases, help determine the intrinsic and extrinsic orientation of the camera (Section 6.3.2).

Over the years, a large number of techniques have been developed for finding vanishing points, including (Quan and Mohr 1989; Collins and Weiss 1990; Brillaut-O'Mahoney 1991; McLean and Kotturi 1995; Becker and Bove 1995; Shufelt 1999; Tuytelaars, Van Gool, and Proesmans 1997; Schaffalitzky and Zisserman 2000; Antone and Teller 2002; Rother 2002; Košecká and Zhang 2005; Pflugfelder 2008; Tardif 2009)—see some of the more recent papers for additional references. In this section, we present a simple Hough technique based on having line pairs vote for potential vanishing point locations, followed by a robust least squares fitting stage. For alternative approaches, please see some of the more recent papers listed above.

The first stage in my vanishing point detection algorithm uses a Hough transform to accumulate votes for likely vanishing point candidates. As with line fitting, one possible approach

(a) (b) (c)

Figure 4.45 Real-world vanishing points: (a) architecture (Sinha, Steedly, Szeliski *et al.* 2008), (b) furniture (Mičušík, Wildenauer, and Košecká 2008) © 2008 IEEE, and (c) calibration patterns (Zhang 2000).

is to have each line vote for *all* possible vanishing point directions, either using a cube map (Tuytelaars, Van Gool, and Proesmans 1997; Antone and Teller 2002) or a Gaussian sphere (Collins and Weiss 1990), optionally using knowledge about the uncertainty in the vanishing point location to perform a weighted vote (Collins and Weiss 1990; Brillaut-O'Mahoney 1991; Shufelt 1999). My preferred approach is to use pairs of detected line segments to form candidate vanishing point locations. Let $\hat{\boldsymbol{m}}_i$ and $\hat{\boldsymbol{m}}_j$ be the (unit norm) line equations for a pair of line segments and l_i and l_j be their corresponding segment lengths. The location of the corresponding vanishing point hypothesis can be computed as

$$\boldsymbol{v}_{ij} = \hat{\boldsymbol{m}}_i \times \hat{\boldsymbol{m}}_j \tag{4.28}$$

and the corresponding weight set to

$$w_{ij} = \|\boldsymbol{v}_{ij}\| l_i l_j. \tag{4.29}$$

This has the desirable effect of downweighting (near-)collinear line segments and short line segments. The Hough space itself can either be represented using spherical coordinates (4.27) or as a cube map (Figure 4.44a).

Once the Hough accumulator space has been populated, peaks can be detected in a manner similar to that previously discussed for line detection. Given a set of candidate line segments that voted for a vanishing point, which can optionally be kept as a list at each Hough accumulator cell, I then use a robust least squares fit to estimate a more accurate location for each vanishing point.

Consider the relationship between the two line segment endpoints $\{\boldsymbol{p}_{i0}, \boldsymbol{p}_{i1}\}$ and the vanishing point $\boldsymbol{v}$, as shown in Figure 4.46. The area A of the triangle given by these three points, which is the magnitude of their triple product

$$A_i = |(\boldsymbol{p}_{i0} \times \boldsymbol{p}_{i1}) \cdot \boldsymbol{v}|, \tag{4.30}$$

is proportional to the perpendicular distance d_1 between each endpoint and the line through $\boldsymbol{v}$ and the other endpoint, as well as the distance between $\boldsymbol{p}_{i0}$ and $\boldsymbol{v}$. Assuming that the accuracy of a fitted line segment is proportional to its endpoint accuracy (Exercise 4.13), this therefore serves as an optimal metric for how well a vanishing point fits a set of extracted lines (Leibowitz (2001, Section 3.6.1) and Pflugfelder (2008, Section 2.1.1.3)). A robustified

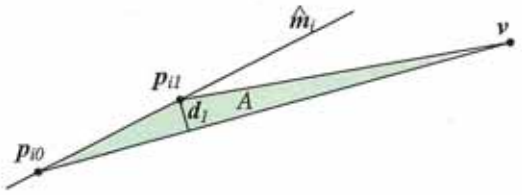


Figure 4.46 Triple product of the line segments endpoints p_{i0} and p_{i1} and the vanishing point v. The area A is proportional to the perpendicular distance d_1 and the distance between the other endpoint p_{i0} and the vanishing point.

least squares estimate (Appendix B.3) for the vanishing point can therefore be written as

$$\mathcal{E} = \sum_i \rho(A_i) = \boldsymbol{v}^T \left(\sum_i w_i(A_i) \boldsymbol{m}_i \boldsymbol{m}_i^T \right) \boldsymbol{v} = \boldsymbol{v}^T \boldsymbol{M} \boldsymbol{v}, \tag{4.31}$$

where $\boldsymbol{m}_i = \boldsymbol{p}_{i0} \times \boldsymbol{p}_{i1}$ is the segment line equation weighted by its length l_i, and $w_i = \rho'(A_i)/A_i$ is the *influence* of each robustified (reweighted) measurement on the final error (Appendix B.3). Notice how this metric is closely related to the original formula for the pairwise weighted Hough transform accumulation step. The final desired value for $\boldsymbol{v}$ is computed as the least eigenvector of $\boldsymbol{M}$.

While the technique described above proceeds in two discrete stages, better results may be obtained by alternating between assigning lines to vanishing points and refitting the vanishing point locations (Antone and Teller 2002; Košecká and Zhang 2005; Pflugfelder 2008). The results of detecting individual vanishing points can also be made more robust by simultaneously searching for pairs or triplets of mutually orthogonal vanishing points (Shufelt 1999; Antone and Teller 2002; Rother 2002; Sinha, Steedly, Szeliski *et al.* 2008). Some results of such vanishing point detection algorithms can be seen in Figure 4.45.

4.3.4 *Application*: Rectangle detection

Once sets of mutually orthogonal vanishing points have been detected, it now becomes possible to search for 3D rectangular structures in the image (Figure 4.47). Over the last decade, a variety of techniques have been developed to find such rectangles, primarily focused on architectural scenes (Košecká and Zhang 2005; Han and Zhu 2005; Shaw and Barnes 2006; Mičušík, Wildenauer, and Košecká 2008; Schindler, Krishnamurthy, Lublinerman *et al.* 2008).

After detecting orthogonal vanishing directions, Košecká and Zhang (2005) refine the fitted line equations, search for corners near line intersections, and then verify rectangle hypotheses by rectifying the corresponding patches and looking for a preponderance of horizontal and vertical edges (Figures 4.47a–b). In follow-on work, Mičušík, Wildenauer, and Košecká (2008) use a Markov random field (MRF) to disambiguate between potentially overlapping rectangle hypotheses. They also use a plane sweep algorithm to match rectangles between different views (Figures 4.47d–f).

A different approach is proposed by Han and Zhu (2005), who use a grammar of potential rectangle shapes and nesting structures (between rectangles and vanishing points) to infer the most likely assignment of line segments to rectangles (Figure 4.47c).

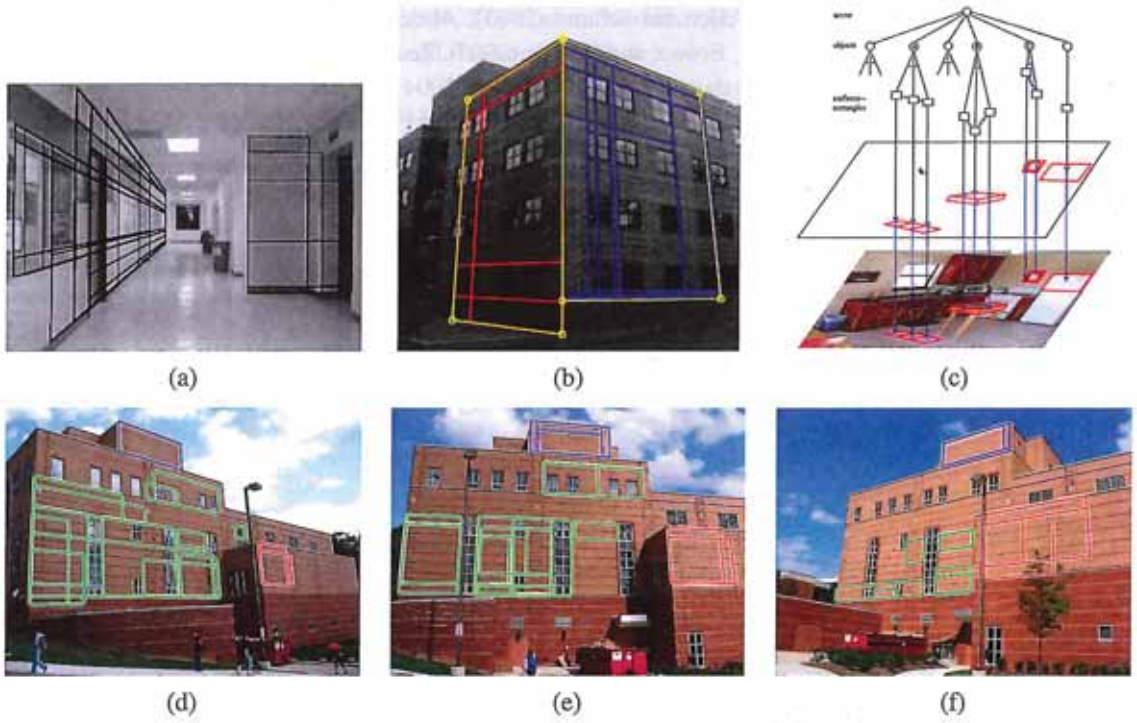


Figure 4.47 Rectangle detection: (a) indoor corridor and (b) building exterior with grouped facades (Košecká and Zhang 2005) © 2005 Elsevier; (c) grammar-based recognition (Han and Zhu 2005) © 2005 IEEE; (d–f) rectangle matching using a plane sweep algorithm (Mičušík, Wildenauer, and Košecká 2008) © 2008 IEEE.

4.4 Additional reading

One of the seminal papers on feature detection, description, and matching is by Lowe (2004). Comprehensive surveys and evaluations of such techniques have been made by Schmid, Mohr, and Bauckhage (2000); Mikolajczyk and Schmid (2005); Mikolajczyk, Tuytelaars, Schmid *et al.* (2005); Tuytelaars and Mikolajczyk (2007) while Shi and Tomasi (1994) and Triggs (2004) also provide nice reviews.

In the area of feature detectors (Mikolajczyk, Tuytelaars, Schmid *et al.* 2005), in addition to such classic approaches as Förstner–Harris (Förstner 1986; Harris and Stephens 1988) and difference of Gaussians (Lindeberg 1993, 1998b; Lowe 2004), maximally stable extremal regions (MSERs) are widely used for applications that require affine invariance (Matas, Chum, Urban *et al.* 2004; Nistér and Stewénius 2008). More recent interest point detectors are discussed by Xiao and Shah (2003); Koethe (2003); Carneiro and Jepson (2005); Kenney, Zuliani, and Manjunath (2005); Bay, Tuytelaars, and Van Gool (2006); Platel, Balmachnova, Florack *et al.* (2006); Rosten and Drummond (2006), as well as techniques based on line matching (Zoghlami, Faugeras, and Deriche 1997; Bartoli, Coquerelle, and Sturm 2004) and region detection (Kadir, Zisserman, and Brady 2004; Matas, Chum, Urban *et al.* 2004; Tuytelaars and Van Gool 2004; Corso and Hager 2005).

A variety of local feature descriptors (and matching heuristics) are surveyed and compared by Mikolajczyk and Schmid (2005). More recent publications in this area include

those by van de Weijer and Schmid (2006); Abdel-Hakim and Farag (2006); Winder and Brown (2007); Hua, Brown, and Winder (2007). Techniques for efficiently matching features include k-d trees (Beis and Lowe 1999; Lowe 2004; Muja and Lowe 2009), pyramid matching kernels (Grauman and Darrell 2005), metric (vocabulary) trees (Nistér and Stewénius 2006), and a variety of multi-dimensional hashing techniques (Shakhnarovich, Viola, and Darrell 2003; Torralba, Weiss, and Fergus 2008; Weiss, Torralba, and Fergus 2008; Kulis and Grauman 2009; Raginsky and Lazebnik 2009).

The classic reference on feature detection and tracking is (Shi and Tomasi 1994). More recent work in this field has focused on learning better matching functions for specific features (Avidan 2001; Jurie and Dhome 2002; Williams, Blake, and Cipolla 2003; Lepetit and Fua 2005; Lepetit, Pilet, and Fua 2006; Hinterstoisser, Benhimane, Navab *et al.* 2008; Rogez, Rihan, Ramalingam *et al.* 2008; Özuysal, Calonder, Lepetit *et al.* 2010).

A highly cited and widely used edge detector is the one developed by Canny (1986). Alternative edge detectors as well as experimental comparisons can be found in publications by Nalwa and Binford (1986); Nalwa (1987); Deriche (1987); Freeman and Adelson (1991); Nalwa (1993); Heath, Sarkar, Sanocki *et al.* (1998); Crane (1997); Ritter and Wilson (2000); Bowyer, Kranenburg, and Dougherty (2001); Arbeláez, Maire, Fowlkes *et al.* (2010). The topic of scale selection in edge detection is nicely treated by Elder and Zucker (1998), while approaches to color and texture edge detection can be found in (Ruzon and Tomasi 2001; Martin, Fowlkes, and Malik 2004; Gevers, van de Weijer, and Stokman 2006). Edge detectors have also recently been combined with region segmentation techniques to further improve the detection of semantically salient boundaries (Maire, Arbelaez, Fowlkes *et al.* 2008; Arbeláez, Maire, Fowlkes *et al.* 2010). Edges linked into contours can be smoothed and manipulated for artistic effect (Lowe 1989; Finkelstein and Salesin 1994; Taubin 1995) and used for recognition (Belongie, Malik, and Puzicha 2002; Tek and Kimia 2003; Sebastian and Kimia 2005).

An early, well-regarded paper on straight line extraction in images was written by Burns, Hanson, and Riseman (1986). More recent techniques often combine line detection with vanishing point detection (Quan and Mohr 1989; Collins and Weiss 1990; Brillaut-O'Mahoney 1991; McLean and Kotturi 1995; Becker and Bove 1995; Shufelt 1999; Tuytelaars, Van Gool, and Proesmans 1997; Schaffalitzky and Zisserman 2000; Antone and Teller 2002; Rother 2002; Košecká and Zhang 2005; Pflugfelder 2008; Sinha, Steedly, Szeliski *et al.* 2008; Tardif 2009).

4.5 Exercises

Ex 4.1: Interest point detector Implement one or more keypoint detectors and compare their performance (with your own or with a classmate's detector).

Possible detectors:

- Laplacian or Difference of Gaussian;
- Förstner–Harris Hessian (try different formula variants given in (4.9–4.11));
- oriented/steerable filter, looking for either second-order high second response or two edges in a window (Koethe 2003), as discussed in Section 4.1.1.

Other detectors are described by Mikolajczyk, Tuytelaars, Schmid *et al.* (2005); Tuytelaars and Mikolajczyk (2007). Additional optional steps could include:

1. Compute the detections on a sub-octave pyramid and find 3D maxima.
2. Find local orientation estimates using steerable filter responses or a gradient histogramming method.
3. Implement non-maximal suppression, such as the adaptive technique of Brown, Szeliski, and Winder (2005).
4. Vary the window shape and size (pre-filter and aggregation).

To test for repeatability, download the code from http://www.robots.ox.ac.uk/~vgg/research/affine/ (Mikolajczyk, Tuytelaars, Schmid *et al.* 2005; Tuytelaars and Mikolajczyk 2007) or simply rotate or shear your own test images. (Pick a domain you may want to use later, e.g., for outdoor stitching.)

Be sure to measure and report the stability of your scale and orientation estimates.

Ex 4.2: Interest point descriptor Implement one or more descriptors (steered to local scale and orientation) and compare their performance (with your own or with a classmate's detector).

Some possible descriptors include

- contrast-normalized patches (Brown, Szeliski, and Winder 2005);
- SIFT (Lowe 2004);
- GLOH (Mikolajczyk and Schmid 2005);
- DAISY (Winder and Brown 2007; Tola, Lepetit, and Fua 2010).

Other detectors are described by Mikolajczyk and Schmid (2005).

Ex 4.3: ROC curve computation Given a pair of curves (histograms) plotting the number of matching and non-matching features as a function of Euclidean distance d as shown in Figure 4.23b, derive an algorithm for plotting a ROC curve (Figure 4.23a). In particular, let $t(d)$ be the distribution of true matches and $f(d)$ be the distribution of (false) non-matches. Write down the equations for the ROC, i.e., TPR(FPR), and the AUC.

(Hint: Plot the cumulative distributions $T(d) = \int t(d)$ and $F(d) = \int f(d)$ and see if these help you derive the TPR and FPR at a given threshold θ.)

Ex 4.4: Feature matcher After extracting features from a collection of overlapping or distorted images,[10] match them up by their descriptors either using nearest neighbor matching or a more efficient matching strategy such as a k-d tree.

See whether you can improve the accuracy of your matches using techniques such as the nearest neighbor distance ratio.

[10] http://www.robots.ox.ac.uk/~vgg/research/affine/.

Ex 4.5: Feature tracker Instead of finding feature points independently in multiple images and then matching them, find features in the first image of a video or image sequence and then re-locate the corresponding points in the next frames using either search and gradient descent (Shi and Tomasi 1994) or learned feature detectors (Lepetit, Pilet, and Fua 2006; Fossati, Dimitrijevic, Lepetit *et al.* 2007). When the number of tracked points drops below a threshold or new regions in the image become visible, find additional points to track.

(Optional) Winnow out incorrect matches by estimating a homography (6.19–6.23) or fundamental matrix (Section 7.2.1).

(Optional) Refine the accuracy of your matches using the iterative registration algorithm described in Section 8.2 and Exercise 8.2.

Ex 4.6: Facial feature tracker Apply your feature tracker to tracking points on a person's face, either manually initialized to interesting locations such as eye corners or automatically initialized at interest points.

(Optional) Match features between two people and use these features to perform image morphing (Exercise 3.25).

Ex 4.7: Edge detector Implement an edge detector of your choice. Compare its performance to that of your classmates' detectors or code downloaded from the Internet.

A simple but well-performing sub-pixel edge detector can be created as follows:

1. Blur the input image a little,
$$B_\sigma(\boldsymbol{x}) = G_\sigma(\boldsymbol{x}) * I(\boldsymbol{x}).$$
2. Construct a Gaussian pyramid (Exercise 3.19),
$$P = \text{Pyramid}\{B_\sigma(\boldsymbol{x})\}$$
3. Subtract an interpolated coarser-level pyramid image from the original resolution blurred image,
$$S(\boldsymbol{x}) = B_\sigma(\boldsymbol{x}) - P.\text{InterpolatedLevel}(L).$$
4. For each quad of pixels, $\{(i,j),(i+1,j),(i,j+1),(i+1,j+1)\}$, count the number of zero crossings along the four edges.
5. When there are exactly two zero crossings, compute their locations using (4.25) and store these edgel endpoints along with the midpoint in the edgel structure (Figure 4.48).
6. For each edgel, compute the local gradient by taking the horizontal and vertical differences between the values of S along the zero crossing edges.
7. Store the magnitude of this gradient as the edge strength and either its orientation or that of the segment joining the edgel endpoints as the edge orientation.
8. Add the edgel to a list of edgels or store it in a 2D array of edgels (addressed by pixel coordinates).

Figure 4.48 shows a possible representation for each computed edgel.

```
struct SEdgel {
    float e[2][2];      // edgel endpoints (zero crossing)
    float x, y;         // sub-pixel edge position (midpoint)
    float n_x, n_y;     // orientation, as normal vector
    float theta;        // orientation, as angle (degrees)
    float length;       // length of edgel
    float strength;     // strength of edgel (gradient magnitude)
};

struct SLine : public SEdgel {
    float line_length;  // length of line (est. from ellipsoid)
    float sigma;        // estimated std. dev. of edgel noise
    float r;            // line equation: x * n_y - y * n_x = r
};
```

Figure 4.48 A potential C++ structure for edgel and line elements.

Ex 4.8: Edge linking and thresholding Link up the edges computed in the previous exercise into chains and optionally perform thresholding with hysteresis.

The steps may include:

1. Store the edgels either in a 2D array (say, an integer image with indices into the edgel list) or pre-sort the edgel list first by (integer) x coordinates and then y coordinates, for faster neighbor finding.

2. Pick up an edgel from the list of unlinked edgels and find its neighbors in both directions until no neighbor is found or a closed contour is obtained. Flag edgels as linked as you visit them and push them onto your list of linked edgels.

3. Alternatively, generalize a previously developed connected component algorithm (Exercise 3.14) to perform the linking in just two raster passes.

4. (Optional) Perform hysteresis-based thresholding (Canny 1986). Use two thresholds "hi" and "lo" for the edge strength. A candidate edgel is considered an edge if either its strength is above the "hi" threshold or its strength is above the "lo" threshold and it is (recursively) connected to a previously detected edge.

5. (Optional) Link together contours that have small gaps but whose endpoints have similar orientations.

6. (Optional) Find junctions between adjacent contours, e.g., using some of the ideas (or references) from Maire, Arbelaez, Fowlkes *et al.* (2008).

Ex 4.9: Contour matching Convert a closed contour (linked edgel list) into its arc-length parameterization and use this to match object outlines.

The steps may include:

1. Walk along the contour and create a list of (x_i, y_i, s_i) triplets, using the arc-length formula
$$s_{i+1} = s_i + \|x_{i+1} - x_i\|. \tag{4.32}$$
2. Resample this list onto a regular set of (x_j, y_j, j) samples using linear interpolation of each segment.
3. Compute the average values of x and y, i.e., $\overline{x}$ and $\overline{y}$ and subtract them from your sampled curve points.
4. Resample the original (x_i, y_i, s_i) piecewise-linear function onto a length-independent set of samples, say $j \in [0, 1023]$. (Using a length which is a power of two makes subsequent Fourier transforms more convenient.)
5. Compute the Fourier transform of the curve, treating each (x, y) pair as a complex number.
6. To compare two curves, fit a linear equation to the phase difference between the two curves. (Careful: phase wraps around at 360°. Also, you may wish to weight samples by their Fourier spectrum magnitude—see Section 8.1.2.)
7. (Optional) Prove that the constant phase component corresponds to the temporal shift in s, while the linear component corresponds to rotation.

Of course, feel free to try any other curve descriptor and matching technique from the computer vision literature (Tek and Kimia 2003; Sebastian and Kimia 2005).

Ex 4.10: Jigsaw puzzle solver—challenging Write a program to automatically solve a jigsaw puzzle from a set of scanned puzzle pieces. Your software may include the following components:

1. Scan the pieces (either face up or face down) on a flatbed scanner with a distinctively colored background.
2. (Optional) Scan in the box top to use as a low-resolution reference image.
3. Use color-based thresholding to isolate the pieces.
4. Extract the contour of each piece using edge finding and linking.
5. (Optional) Re-represent each contour using an arc-length or some other re-parameterization. Break up the contours into meaningful matchable pieces. (Is this hard?)
6. (Optional) Associate color values with each contour to help in the matching.
7. (Optional) Match pieces to the reference image using some rotationally invariant feature descriptors.
8. Solve a global optimization or (backtracking) search problem to snap pieces together and place them in the correct location relative to the reference image.
9. Test your algorithm on a succession of more difficult puzzles and compare your results with those of others.

Ex 4.11: Successive approximation line detector Implement a line simplification algorithm (Section 4.3.1) (Ramer 1972; Douglas and Peucker 1973) to convert a hand-drawn curve (or linked edge image) into a small set of polylines.

(Optional) Re-render this curve using either an approximating or interpolating spline or Bezier curve (Szeliski and Ito 1986; Bartels, Beatty, and Barsky 1987; Farin 1996).

Ex 4.12: Hough transform line detector Implement a Hough transform for finding lines in images:

1. Create an accumulator array of the appropriate user-specified size and clear it. The user can specify the spacing in degrees between orientation bins and in pixels between distance bins. The array can be allocated as integer (for simple counts), floating point (for weighted counts), or as an array of vectors for keeping back pointers to the constituent edges.

2. For each detected edgel at location (x, y) and orientation $\theta = \tan^{-1} n_y/n_x$, compute the value of

$$d = xn_x + yn_y \tag{4.33}$$

   and increment the accumulator corresponding to (θ, d).

   (Optional) Weight the vote of each edge by its length (see Exercise 4.7) or the strength of its gradient.

3. (Optional) Smooth the scalar accumulator array by adding in values from its immediate neighbors. This can help counteract the *discretization* effect of voting for only a single bin—see Exercise 3.7.

4. Find the largest peaks (local maxima) in the accumulator corresponding to lines.

5. (Optional) For each peak, re-fit the lines to the constituent edgels, using *total least squares* (Appendix A.2). Use the original edgel lengths or strength weights to weight the least squares fit, as well as the agreement between the hypothesized line orientation and the edgel orientation. Determine whether these heuristics help increase the accuracy of the fit.

6. After fitting each peak, zero-out or eliminate that peak and its adjacent bins in the array, and move on to the next largest peak.

Test out your Hough transform on a variety of images taken indoors and outdoors, as well as checkerboard calibration patterns.

For checkerboard patterns, you can modify your Hough transform by collapsing *antipodal* bins $(\theta \pm 180°, -d)$ with (θ, d) to find lines that do not care about polarity changes. Can you think of examples in real-world images where this might be desirable as well?

Ex 4.13: Line fitting uncertainty Estimate the uncertainty (covariance) in your line fit using uncertainty analysis.

1. After determining which edgels belong to the line segment (using either successive approximation or Hough transform), re-fit the line segment using total least squares (Van Huffel and Vandewalle 1991; Van Huffel and Lemmerling 2002), i.e., find the

mean or centroid of the edgels and then use eigenvalue analysis to find the dominant orientation.

2. Compute the perpendicular errors (deviations) to the line and robustly estimate the variance of the fitting noise using an estimator such as MAD (Appendix B.3).
3. (Optional) re-fit the line parameters by throwing away outliers or using a robust norm or influence function.
4. Estimate the error in the perpendicular location of the line segment and its orientation.

Ex 4.14: Vanishing points Compute the vanishing points in an image using one of the techniques described in Section 4.3.3 and optionally refine the original line equations associated with each vanishing point. Your results can be used later to track a target (Exercise 6.5) or reconstruct architecture (Section 12.6.1).

Ex 4.15: Vanishing point uncertainty Perform an uncertainty analysis on your estimated vanishing points. You will need to decide how to represent your vanishing point, e.g., homogeneous coordinates on a sphere, to handle vanishing points near infinity.

See the discussion of Bingham distributions by Collins and Weiss (1990) for some ideas.

Chapter 6

Feature-based alignment

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_6, © Springer-Verlag London Limited 2011

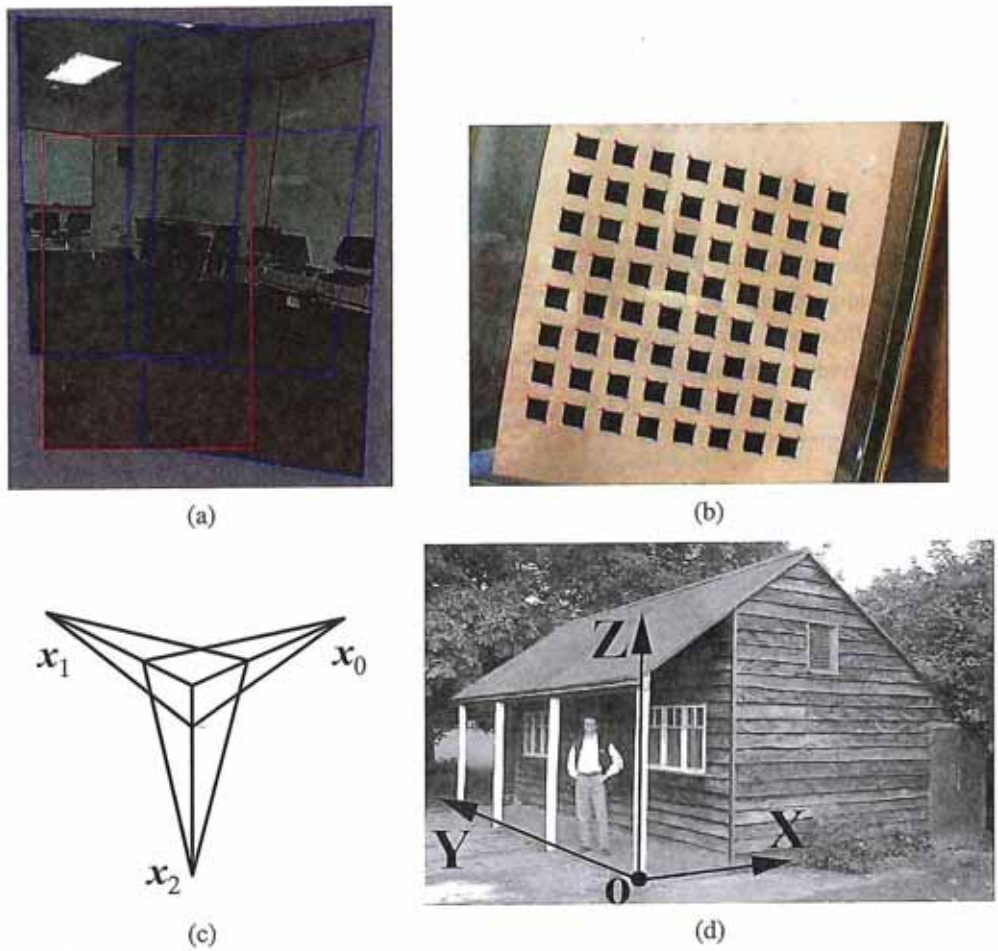


Figure 6.1 Geometric alignment and calibration: (a) geometric alignment of 2D images for stitching (Szeliski and Shum 1997) © 1997 ACM; (b) a two-dimensional calibration target (Zhang 2000) © 2000 IEEE; (c) calibration from vanishing points; (d) scene with easy-to-find lines and vanishing directions (Criminisi, Reid, and Zisserman 2000) © 2000 Springer.

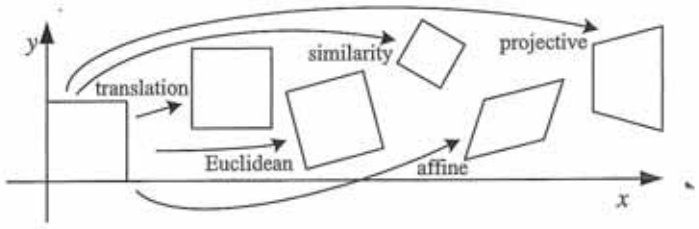


Figure 6.2 Basic set of 2D planar transformations

Once we have extracted features from images, the next stage in many vision algorithms is to match these features across different images (Section 4.1.3). An important component of this matching is to verify whether the set of matching features is geometrically consistent, e.g., whether the feature displacements can be described by a simple 2D or 3D geometric transformation. The computed motions can then be used in other applications such as image stitching (Chapter 9) or augmented reality (Section 6.2.3).

In this chapter, we look at the topic of geometric image registration, i.e., the computation of 2D and 3D transformations that map features in one image to another (Section 6.1). One special case of this problem is *pose estimation*, which is determining a camera's position relative to a known 3D object or scene (Section 6.2). Another case is the computation of a camera's *intrinsic calibration*, which consists of the internal parameters such as focal length and radial distortion (Section 6.3). In Chapter 7, we look at the related problems of how to estimate 3D point structure from 2D matches (*triangulation*) and how to simultaneously estimate 3D geometry and camera motion (*structure from motion*).

6.1 2D and 3D feature-based alignment

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points. In this section, we restrict ourselves to global *parametric* transformations, such as those described in Section 2.1.2 and shown in Table 2.1 and Figure 6.2, or higher order transformation for curved surfaces (Shashua and Toelg 1997; Can, Stewart, Roysam *et al.* 2002). Applications to non-rigid or elastic deformations (Bookstein 1989; Szeliski and Lavallée 1996; Torresani, Hertzmann, and Bregler 2008) are examined in Sections 8.3 and 12.6.4.

6.1.1 2D alignment using least squares

Given a set of matched feature points $\{(\boldsymbol{x}_i, \boldsymbol{x}'_i)\}$ and a planar parametric transformation[1] of the form

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}), \tag{6.1}$$

[1] For examples of non-planar parametric models, such as quadrics, see the work of Shashua and Toelg (1997); Shashua and Wexler (2001).

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Table 6.1 Jacobians of the 2D coordinate transformations $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})$ shown in Table 2.1, where we have re-parameterized the motions so that they are identity for $\boldsymbol{p} = 0$.

how can we produce the best estimate of the motion parameters $\boldsymbol{p}$? The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals

$$E_{\text{LS}} = \sum_i \|\boldsymbol{r}_i\|^2 = \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}) - \boldsymbol{x}_i'\|^2, \tag{6.2}$$

where

$$\boldsymbol{r}_i = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}) - \boldsymbol{x}_i' = \hat{\boldsymbol{x}}_i' - \tilde{\boldsymbol{x}}_i' \tag{6.3}$$

is the *residual* between the measured location $\hat{\boldsymbol{x}}_i'$ and its corresponding current *predicted* location $\tilde{\boldsymbol{x}}_i' = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p})$. (See Appendix A.2 for more on least squares and Appendix B.2 for a statistical justification.)

Many of the motion models presented in Section 2.1.2 and Table 2.1, i.e., translation, similarity, and affine, have a *linear* relationship between the amount of motion $\Delta \boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{x}$ and the unknown parameters $\boldsymbol{p}$,

$$\Delta \boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{x} = \boldsymbol{J}(\boldsymbol{x})\boldsymbol{p}, \tag{6.4}$$

where $\boldsymbol{J} = \partial \boldsymbol{f} / \partial \boldsymbol{p}$ is the *Jacobian* of the transformation $\boldsymbol{f}$ with respect to the motion parameters $\boldsymbol{p}$ (see Table 6.1). In this case, a simple *linear* regression (linear least squares problem) can be formulated as

$$\begin{aligned} E_{\text{LLS}} &= \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{p} - \Delta \boldsymbol{x}_i\|^2 && (6.5) \\ &= \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)\right] \boldsymbol{p} - 2\boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\Delta \boldsymbol{x}_i\right] + \sum_i \|\Delta \boldsymbol{x}_i\|^2 && (6.6) \\ &= \boldsymbol{p}^T \boldsymbol{A}\boldsymbol{p} - 2\boldsymbol{p}^T \boldsymbol{b} + c. && (6.7) \end{aligned}$$

The minimum can be found by solving the symmetric positive definite (SPD) system of *normal equations*[2]

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}, \tag{6.8}$$

where

$$\boldsymbol{A} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i) \tag{6.9}$$

is called the *Hessian* and $\boldsymbol{b} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\Delta\boldsymbol{x}_i$. For the case of pure translation, the resulting equations have a particularly simple form, i.e., the translation is the average translation between corresponding points or, equivalently, the translation of the point centroids.

Uncertainty weighting. The above least squares formulation assumes that all feature points are matched with the same accuracy. This is often not the case, since certain points may fall into more textured regions than others. If we associate a scalar variance estimate σ_i^2 with each correspondence, we can minimize the *weighted least squares* problem instead,[3]

$$E_{\text{WLS}} = \sum_i \sigma_i^{-2}\|\boldsymbol{r}_i\|^2. \tag{6.10}$$

As shown in Section 8.1.3, a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the *patch Hessian* $\boldsymbol{A}_i$ (8.55) with the per-pixel noise covariance σ_n^2 (8.44). Weighting each squared residual by its inverse covariance $\Sigma_i^{-1} = \sigma_n^{-2}\boldsymbol{A}_i$ (which is called the *information matrix*), we obtain

$$E_{\text{CWLS}} = \sum_i \|\boldsymbol{r}_i\|^2_{\Sigma_i^{-1}} = \sum_i \boldsymbol{r}_i^T\Sigma_i^{-1}\boldsymbol{r}_i = \sum_i \sigma_n^{-2}\boldsymbol{r}_i^T\boldsymbol{A}_i\boldsymbol{r}_i. \tag{6.11}$$

6.1.2 *Application*: Panography

One of the simplest (and most fun) applications of image alignment is a special form of image stitching called *panography*. In a panograph, images are translated and optionally rotated and scaled before being blended with simple averaging (Figure 6.3). This process mimics the photographic collages created by artist David Hockney, although his compositions use an opaque overlay model, being created out of regular photographs.

In most of the examples seen on the Web, the images are aligned by hand for best artistic effect.[4] However, it is also possible to use feature matching and alignment techniques to perform the registration automatically (Nomura, Zhang, and Nayar 2007; Zelnik-Manor and Perona 2007).

Consider a simple translational model. We want all the corresponding features in different images to line up as best as possible. Let $\boldsymbol{t}_j$ be the location of the jth image coordinate frame in the global composite frame and $\boldsymbol{x}_{ij}$ be the location of the ith matched feature in the jth image. In order to align the images, we wish to minimize the least squares error

$$E_{\text{PLS}} = \sum_{ij} \|(\boldsymbol{t}_j + \boldsymbol{x}_{ij}) - \boldsymbol{x}_i\|^2, \tag{6.12}$$

[2] For poorly conditioned problems, it is better to use QR decomposition on the set of linear equations $\boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{p} = \Delta\boldsymbol{x}_i$ instead of the normal equations (Björck 1996; Golub and Van Loan 1996). However, such conditions rarely arise in image registration.

[3] Problems where each measurement can have a different variance or certainty are called *heteroscedastic models*.

[4] http://www.flickr.com/groups/panography/.

Figure 6.3 A simple panograph consisting of three images automatically aligned with a translational model and then averaged together.

where $\boldsymbol{x}_i$ is the consensus (average) position of feature i in the global coordinate frame. (An alternative approach is to register each pair of overlapping images separately and then compute a consensus location for each frame—see Exercise 6.2.)

The above least squares problem is indeterminate (you can add a constant offset to all the frame and point locations $\boldsymbol{t}_j$ and $\boldsymbol{x}_i$). To fix this, either pick one frame as being at the origin or add a constraint to make the average frame offsets be 0.

The formulas for adding rotation and scale transformations are straightforward and are left as an exercise (Exercise 6.2). See if you can create some collages that you would be happy to share with others on the Web.

6.1.3 Iterative algorithms

While linear least squares is the simplest method for estimating parameters, most problems in computer vision do not have a simple linear relationship between the measurements and the unknowns. In this case, the resulting problem is called *non-linear least squares* or *non-linear regression*.

Consider, for example, the problem of estimating a rigid Euclidean 2D transformation (translation plus rotation) between two sets of points. If we parameterize this transformation by the translation amount (t_x, t_y) and the rotation angle θ, as in Table 2.1, the Jacobian of this transformation, given in Table 6.1, depends on the current value of θ. Notice how in Table 6.1, we have re-parameterized the motion matrices so that they are always the identity at the origin $\boldsymbol{p} = 0$, which makes it easier to initialize the motion parameters.

To minimize the non-linear least squares problem, we iteratively find an update $\Delta\boldsymbol{p}$ to the current parameter estimate $\boldsymbol{p}$ by minimizing

$$E_{\mathrm{NLS}}(\Delta\boldsymbol{p}) = \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta\boldsymbol{p}) - \boldsymbol{x}_i'\|^2 \tag{6.13}$$

$$\approx \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i; \boldsymbol{p})\Delta\boldsymbol{p} - \boldsymbol{r}_i\|^2 \tag{6.14}$$

$$= \Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{J} \right] \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{r}_i \right] + \sum_i \|\boldsymbol{r}_i\|^2 \quad (6.15)$$

$$= \Delta \boldsymbol{p}^T \boldsymbol{A} \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \boldsymbol{b} + c, \quad (6.16)$$

where the "Hessian"[5] $\boldsymbol{A}$ is the same as Equation (6.9) and the right hand side vector

$$\boldsymbol{b} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i) \boldsymbol{r}_i \quad (6.17)$$

is now a Jacobian-weighted sum of residual vectors. This makes intuitive sense, as the parameters are pulled in the direction of the prediction error with a strength proportional to the Jacobian.

Once $\boldsymbol{A}$ and $\boldsymbol{b}$ have been computed, we solve for $\Delta \boldsymbol{p}$ using

$$(\boldsymbol{A} + \lambda \text{diag}(\boldsymbol{A})) \Delta \boldsymbol{p} = \boldsymbol{b}, \quad (6.18)$$

and update the parameter vector $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta \boldsymbol{p}$ accordingly. The parameter λ is an additional damping parameter used to ensure that the system takes a "downhill" step in energy (squared error) and is an essential component of the Levenberg–Marquardt algorithm (described in more detail in Appendix A.3). In many applications, it can be set to 0 if the system is successfully converging.

For the case of our 2D translation+rotation, we end up with a 3×3 set of normal equations in the unknowns $(\delta t_x, \delta t_y, \delta \theta)$. An initial guess for (t_x, t_y, θ) can be obtained by fitting a four-parameter similarity transform in (t_x, t_y, c, s) and then setting $\theta = \tan^{-1}(s/c)$. An alternative approach is to estimate the translation parameters using the centroids of the 2D points and to then estimate the rotation angle using polar coordinates (Exercise 6.3).

For the other 2D motion models, the derivatives in Table 6.1 are all fairly straightforward, except for the projective 2D motion (homography), which arises in image-stitching applications (Chapter 9). These equations can be re-written from (2.21) in their new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \text{ and } y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \quad (6.19)$$

The Jacobian is therefore

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \quad (6.20)$$

where $D = h_{20}x + h_{21}y + 1$ is the denominator in (6.19), which depends on the current parameter settings (as do x' and y').

An initial guess for the eight unknowns $\{h_{00}, h_{01}, \ldots, h_{21}\}$ can be obtained by multiplying both sides of the equations in (6.19) through by the denominator, which yields the linear set of equations,

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \quad (6.21)$$

[5] The "Hessian" $\boldsymbol{A}$ is not the true Hessian (second derivative) of the non-linear least squares problem (6.13). Instead, it is the approximate Hessian, which neglects second (and higher) order derivatives of $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta \boldsymbol{p})$.

However, this is not optimal from a statistical point of view, since the denominator D, which was used to multiply each equation, can vary quite a bit from point to point.[6]

One way to compensate for this is to *reweight* each equation by the inverse of the current estimate of the denominator, D,

$$\frac{1}{D}\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{6.22}$$

While this may at first seem to be the exact same set of equations as (6.21), because least squares is being used to solve the over-determined set of equations, the weightings *do* matter and produce a different set of normal equations that performs better in practice.

The most principled way to do the estimation, however, is to directly minimize the squared residual equations (6.13) using the Gauss–Newton approximation, i.e., performing a first-order Taylor series expansion in $\boldsymbol{p}$, as shown in (6.14), which yields the set of equations

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \tag{6.23}$$

While these look similar to (6.22), they differ in two important respects. First, the left hand side consists of unweighted *prediction errors* rather than point displacements and the solution vector is a *perturbation* to the parameter vector $\boldsymbol{p}$. Second, the quantities inside $\boldsymbol{J}$ involve *predicted* feature locations $(\tilde{x}', \tilde{y}')$ instead of *sensed* feature locations $(\hat{x}', \hat{y}')$. Both of these differences are subtle and yet they lead to an algorithm that, when combined with proper checking for downhill steps (as in the Levenberg–Marquardt algorithm), will converge to a local minimum. Note that iterating Equations (6.22) is not guaranteed to converge, since it is not minimizing a well-defined energy function.

Equation (6.23) is analogous to the *additive* algorithm for direct intensity-based registration (Section 8.2), since the change to the full transformation is being computed. If we prepend an incremental homography to the current homography instead, i.e., we use a *compositional* algorithm (described in Section 8.2), we get $D = 1$ (since $\boldsymbol{p} = 0$) and the above formula simplifies to

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}, \tag{6.24}$$

where we have replaced $(\tilde{x}', \tilde{y}')$ with (x, y) for conciseness. (Notice how this results in the same Jacobian as (8.63).)

[6] Hartley and Zisserman (2004) call this strategy of forming linear equations from rational equations the *direct linear transform*, but that term is more commonly associated with pose estimation (Section 6.2). Note also that our definition of the h_{ij} parameters differs from that used in their book, since we define h_{ii} to be the *difference* from unity and we do not leave h_{22} as a free parameter, which means that we cannot handle certain extreme homographies.

6.1.4 Robust least squares and RANSAC

While regular least squares is the method of choice for measurements where the noise follows a normal (Gaussian) distribution, more robust versions of least squares are required when there are outliers among the correspondences (as there almost always are). In this case, it is preferable to use an *M-estimator* (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Black and Rangarajan 1996; Stewart 1999), which involves applying a robust penalty function $\rho(r)$ to the residuals

$$E_{\mathrm{RLS}}(\Delta \boldsymbol{p}) = \sum_i \rho(\|\boldsymbol{r}_i\|) \tag{6.25}$$

instead of squaring them.

We can take the derivative of this function with respect to $\boldsymbol{p}$ and set it to 0,

$$\sum_i \psi(\|\boldsymbol{r}_i\|) \frac{\partial \|\boldsymbol{r}_i\|}{\partial \boldsymbol{p}} = \sum_i \frac{\psi(\|\boldsymbol{r}_i\|)}{\|\boldsymbol{r}_i\|} \boldsymbol{r}_i^T \frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{p}} = 0, \tag{6.26}$$

where $\psi(r) = \rho'(r)$ is the derivative of ρ and is called the *influence function*. If we introduce a *weight function*, $w(r) = \Psi(r)/r$, we observe that finding the stationary point of (6.25) using (6.26) is equivalent to minimizing the *iteratively reweighted least squares* (IRLS) problem

$$E_{\mathrm{IRLS}} = \sum_i w(\|\boldsymbol{r}_i\|) \|\boldsymbol{r}_i\|^2, \tag{6.27}$$

where the $w(\|\boldsymbol{r}_i\|)$ play the same local weighting role as σ_i^{-2} in (6.10). The IRLS algorithm alternates between computing the influence functions $w(\|\boldsymbol{r}_i\|)$ and solving the resulting weighted least squares problem (with fixed w values). Other incremental robust least squares algorithms can be found in the work of Sawhney and Ayer (1996); Black and Anandan (1996); Black and Rangarajan (1996); Baker, Gross, Ishikawa *et al.* (2003) and textbooks and tutorials on robust statistics (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Rousseeuw and Leroy 1987; Stewart 1999).

While M-estimators can definitely help reduce the influence of outliers, in some cases, starting with too many outliers will prevent IRLS (or other gradient descent algorithms) from converging to the global optimum. A better approach is often to find a starting set of *inlier* correspondences, i.e., points that are consistent with a dominant motion estimate.[7]

Two widely used approaches to this problem are called RANdom SAmple Consensus, or RANSAC for short (Fischler and Bolles 1981), and *least median of squares* (LMS) (Rousseeuw 1984). Both techniques start by selecting (at random) a subset of k correspondences, which is then used to compute an initial estimate for $\boldsymbol{p}$. The *residuals* of the full set of correspondences are then computed as

$$\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i; \boldsymbol{p}) - \hat{\boldsymbol{x}}_i', \tag{6.28}$$

where $\tilde{\boldsymbol{x}}_i'$ are the *estimated* (mapped) locations and $\hat{\boldsymbol{x}}_i'$ are the sensed (detected) feature point locations.

The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location, i.e., whose $\|\boldsymbol{r}_i\| \leq \epsilon$. (The ϵ value is application dependent but is often around 1–3 pixels.) Least median of squares finds the median value of the $\|\boldsymbol{r}_i\|^2$ values. The

[7] For pixel-based alignment methods (Section 8.1.1), hierarchical (coarse-to-fine) techniques are often used to lock onto the *dominant motion* in a scene.

k	p	S
3	0.5	35
6	0.6	97
6	0.5	293

Table 6.2 Number of trials S to attain a 99% probability of success (Stewart 1999).

random selection process is repeated S times and the sample set with the largest number of inliers (or with the smallest median residual) is kept as the final solution. Either the initial parameter guess $\boldsymbol{p}$ or the full set of computed inliers is then passed on to the next data fitting stage.

When the number of measurements is quite large, it may be preferable to only score a subset of the measurements in an initial round that selects the most plausible hypotheses for additional scoring and selection. This modification of RANSAC, which can significantly speed up its performance, is called *Preemptive RANSAC* (Nistér 2003). In another variant on RANSAC called PROSAC (PROgressive SAmple Consensus), random samples are initially added from the most "confident" matches, thereby speeding up the process of finding a (statistically) likely good set of inliers (Chum and Matas 2005).

To ensure that the random sampling has a good chance of finding a true set of inliers, a sufficient number of trials S must be tried. Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials. The likelihood in one trial that all k random samples are inliers is p^k. Therefore, the likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S \tag{6.29}$$

and the required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}. \tag{6.30}$$

Stewart (1999) gives examples of the required number of trials S to attain a 99% probability of success. As you can see from Table 6.2, the number of trials grows quickly with the number of sample points used. This provides a strong incentive to use the *minimum* number of sample points k possible for any given trial, which is how RANSAC is normally used in practice.

Uncertainty modeling

In addition to robustly computing a good alignment, some applications require the computation of uncertainty (see Appendix B.6). For linear problems, this estimate can be obtained by inverting the Hessian matrix (6.9) and multiplying it by the feature position noise (if these have not already been used to weight the individual measurements, as in Equations (6.10) and 6.11)). In statistics, the Hessian, which is the inverse covariance, is sometimes called the (Fisher) *information matrix* (Appendix B.1.1).

When the problem involves non-linear least squares, the inverse of the Hessian matrix provides the *Cramer–Rao lower bound* on the covariance matrix, i.e., it provides the *minimum*

amount of covariance in a given solution, which can actually have a wider spread ("longer tails") if the energy flattens out away from the local minimum where the optimal solution is found.

6.1.5 3D alignment

Instead of aligning 2D sets of image features, many computer vision applications require the alignment of 3D points. In the case where the 3D transformations are linear in the motion parameters, e.g., for translation, similarity, and affine, regular least squares (6.5) can be used.

The case of rigid (Euclidean) motion,

$$E_{\mathrm{R3D}} = \sum_i \|\boldsymbol{x}_i' - \boldsymbol{R}\boldsymbol{x}_i - \boldsymbol{t}\|^2, \tag{6.31}$$

which arises more frequently and is often called the *absolute orientation* problem (Horn 1987), requires slightly different techniques. If only scalar weightings are being used (as opposed to full 3D per-point anisotropic covariance estimates), the weighted centroids of the two point clouds $\boldsymbol{c}$ and $\boldsymbol{c}'$ can be used to estimate the translation $\boldsymbol{t} = \boldsymbol{c}' - \boldsymbol{R}\boldsymbol{c}$.[8] We are then left with the problem of estimating the rotation between two sets of points $\{\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{c}\}$ and $\{\hat{\boldsymbol{x}}_i' = \boldsymbol{x}_i' - \boldsymbol{c}'\}$ that are both centered at the origin.

One commonly used technique is called the *orthogonal Procrustes algorithm* (Golub and Van Loan 1996, p. 601) and involves computing the singular value decomposition (SVD) of the 3×3 correlation matrix

$$\boldsymbol{C} = \sum_i \hat{\boldsymbol{x}}'\hat{\boldsymbol{x}}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T. \tag{6.32}$$

The rotation matrix is then obtained as $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$. (Verify this for yourself when $\hat{\boldsymbol{x}}' = \boldsymbol{R}\hat{\boldsymbol{x}}$.)

Another technique is the absolute orientation algorithm (Horn 1987) for estimating the unit quaternion corresponding to the rotation matrix $\boldsymbol{R}$, which involves forming a 4×4 matrix from the entries in $\boldsymbol{C}$ and then finding the eigenvector associated with its largest positive eigenvalue.

Lorusso, Eggert, and Fisher (1995) experimentally compare these two techniques to two additional techniques proposed in the literature, but find that the difference in accuracy is negligible (well below the effects of measurement noise).

In situations where these closed-form algorithms are not applicable, e.g., when full 3D covariances are being used or when the 3D alignment is part of some larger optimization, the incremental rotation update introduced in Section 2.1.4 (2.35–2.36), which is parameterized by an instantaneous rotation vector $\boldsymbol{\omega}$, can be used (See Section 9.1.3 for an application to image stitching.)

In some situations, e.g., when merging range data maps, the correspondence between data points is not known *a priori*. In this case, iterative algorithms that start by matching nearby points and then update the most likely correspondence can be used (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan, Lu *et al.* 1998; David, DeMenthon, Duraiswami *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). These techniques are discussed in more detail in Section 12.2.1.

[8] When full covariances are used, they are transformed by the rotation and so a closed-form solution for translation is not possible.

6.2 Pose estimation

A particular instance of feature-based alignment, which occurs very often, is estimating an object's 3D pose from a set of 2D point projections. This *pose estimation* problem is also known as *extrinsic* calibration, as opposed to the *intrinsic* calibration of internal camera parameters such as focal length, which we discuss in Section 6.3. The problem of recovering pose from three correspondences, which is the minimal amount of information necessary, is known as the *perspective-3-point-problem* (P3P), with extensions to larger numbers of points collectively known as PnP (Haralick, Lee, Ottenberg *et al.* 1994; Quan and Lan 1999; Moreno-Noguer, Lepetit, and Fua 2007).

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

6.2.1 Linear algorithms

The simplest way to recover the pose of the camera is to form a set of linear equations analogous to those used for 2D motion estimation (6.19) from the camera matrix form of perspective projection (2.55–2.56),

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}} \tag{6.33}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \tag{6.34}$$

where (x_i, y_i) are the measured 2D feature locations and (X_i, Y_i, Z_i) are the known 3D feature locations (Figure 6.4). As with (6.21), this system of equations can be solved in a linear fashion for the unknowns in the camera matrix $\boldsymbol{P}$ by multiplying the denominator on both sides of the equation.[9] The resulting algorithm is called the *direct linear transform* (DLT) and is commonly attributed to Sutherland (1974). (For a more in-depth discussion, refer to the work of Hartley and Zisserman (2004).) In order to compute the 12 (or 11) unknowns in $\boldsymbol{P}$, at least six correspondences between 3D and 2D locations must be known.

As with the case of estimating homographies (6.21–6.23), more accurate results for the entries in $\boldsymbol{P}$ can be obtained by directly minimizing the set of Equations (6.33–6.34) using non-linear least squares with a small number of iterations.

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

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]. \tag{6.35}$$

Since $\boldsymbol{K}$ is by convention upper-triangular (see the discussion in Section 2.1.5), both $\boldsymbol{K}$ and $\boldsymbol{R}$ can be obtained from the front 3×3 sub-matrix of $\boldsymbol{P}$ using RQ factorization (Golub and Van Loan 1996).[10]

[9] Because $\boldsymbol{P}$ is unknown up to a scale, we can either fix one of the entries, e.g., $p_{23} = 1$, or find the smallest singular vector of the set of linear equations.

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

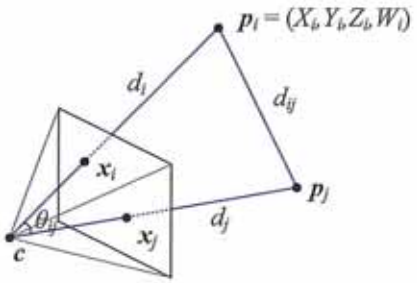


Figure 6.4 Pose estimation by the direct linear transform and by measuring visual angles and distances between pairs of points.

In most applications, however, we have some prior knowledge about the intrinsic calibration matrix $\boldsymbol{K}$, e.g., that the pixels are square, the skew is very small, and the optical center is near the center of the image (2.57–2.59). Such constraints can be incorporated into a non-linear minimization of the parameters in $\boldsymbol{K}$ and $(\boldsymbol{R}, \boldsymbol{t})$, as described in Section 6.2.2.

In the case where the camera is already calibrated, i.e., the matrix $\boldsymbol{K}$ is known (Section 6.3), we can perform pose estimation using as few as three points (Fischler and Bolles 1981; Haralick, Lee, Ottenberg *et al.* 1994; Quan and Lan 1999). The basic observation that these *linear PnP* (*perspective n-point*) algorithms employ is that the visual angle between any pair of 2D points $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{x}}_j$ must be the same as the angle between their corresponding 3D points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ (Figure 6.4).

Given a set of corresponding 2D and 3D points $\{(\hat{\boldsymbol{x}}_i, \boldsymbol{p}_i)\}$, where the $\hat{\boldsymbol{x}}_i$ are unit directions obtained by transforming 2D pixel measurements $\boldsymbol{x}_i$ to unit norm 3D directions $\hat{\boldsymbol{x}}_i$ through the inverse calibration matrix $\boldsymbol{K}$,

$$\hat{\boldsymbol{x}}_i = \mathcal{N}(\boldsymbol{K}^{-1}\boldsymbol{x}_i) = \boldsymbol{K}^{-1}\boldsymbol{x}_i / \|\boldsymbol{K}^{-1}\boldsymbol{x}_i\|, \tag{6.36}$$

the unknowns are the distances d_i from the camera origin $\boldsymbol{c}$ to the 3D points $\boldsymbol{p}_i$, where

$$\boldsymbol{p}_i = d_i\hat{\boldsymbol{x}}_i + \boldsymbol{c} \tag{6.37}$$

(Figure 6.4). The cosine law for triangle $\Delta(\boldsymbol{c}, \boldsymbol{p}_i, \boldsymbol{p}_j)$ gives us

$$f_{ij}(d_i, d_j) = d_i^2 + d_j^2 - 2d_i d_j c_{ij} - d_{ij}^2 = 0, \tag{6.38}$$

where

$$c_{ij} = \cos\theta_{ij} = \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j \tag{6.39}$$

and

$$d_{ij}^2 = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2. \tag{6.40}$$

We can take any triplet of constraints (f_{ij}, f_{ik}, f_{jk}) and eliminate the d_j and d_k using Sylvester resultants (Cox, Little, and O'Shea 2007) to obtain a quartic equation in d_i^2,

$$g_{ijk}(d_i^2) = a_4 d_i^8 + a_3 d_i^6 + a_2 d_i^4 + a_1 d_i^2 + a_0 = 0. \tag{6.41}$$

Given five or more correspondences, we can generate $\frac{(n-1)(n-2)}{2}$ triplets to obtain a linear estimate (using SVD) for the values of $(d_i^8, d_i^6, d_i^4, d_i^2)$ (Quan and Lan 1999). Estimates for

d_i^2 can computed as ratios of successive d_i^{2n+2}/d_i^{2n} estimates and these can be averaged to obtain a final estimate of d_i^2 (and hence d_i).

Once the individual estimates of the d_i distances have been computed, we can generate a 3D structure consisting of the scaled point directions $d_i\hat{\boldsymbol{x}}_i$, which can then be aligned with the 3D point cloud $\{\boldsymbol{p}_i\}$ using absolute orientation (Section 6.1.5) to obtained the desired pose estimate. Quan and Lan (1999) give accuracy results for this and other techniques, which use fewer points but require more complicated algebraic manipulations. The paper by Moreno-Noguer, Lepetit, and Fua (2007) reviews more recent alternatives and also gives a lower complexity algorithm that typically produces more accurate results.

Unfortunately, because minimal PnP solutions can be quite noise sensitive and also suffer from *bas-relief ambiguities* (e.g., depth reversals) (Section 7.4.3), it is often preferable to use the linear six-point algorithm to guess an initial pose and then optimize this estimate using the iterative technique described in Section 6.2.2.

An alternative pose estimation algorithm involves starting with a scaled orthographic projection model and then iteratively refining this initial estimate using a more accurate perspective projection model (DeMenthon and Davis 1995). The attraction of this model, as stated in the paper's title, is that it can be implemented "in 25 lines of [Mathematica] code".

6.2.2 Iterative algorithms

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

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{p}_i; \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{K}) \tag{6.42}$$

and iteratively minimize the robustified linearized reprojection errors

$$E_{\text{NLP}} = \sum_i \rho\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}}\Delta\boldsymbol{R} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}}\Delta\boldsymbol{t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{K}}\Delta\boldsymbol{K} - \boldsymbol{r}_i\right), \tag{6.43}$$

where $\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_i$ is the current residual vector (2D error in predicted position) and the partial derivatives are with respect to the unknown pose parameters (rotation, translation, and optionally calibration). Note that if full 2D covariance estimates are available for the 2D feature locations, the above squared norm can be weighted by the inverse point covariance matrix, as in Equation (6.11).

An easier to understand (and implement) version of the above non-linear regression problem can be constructed by re-writing the projection equations as a concatenation of simpler steps, each of which transforms a 4D homogeneous coordinate $\boldsymbol{p}_i$ by a simple transformation such as translation, rotation, or perspective division (Figure 6.5). The resulting projection equations can be written as

$$\boldsymbol{y}^{(1)} = \boldsymbol{f}_{\text{T}}(\boldsymbol{p}_i; \boldsymbol{c}_j) = \boldsymbol{p}_i - \boldsymbol{c}_j, \tag{6.44}$$

$$\boldsymbol{y}^{(2)} = \boldsymbol{f}_{\text{R}}(\boldsymbol{y}^{(1)}; \boldsymbol{q}_j) = \boldsymbol{R}(\boldsymbol{q}_j)\,\boldsymbol{y}^{(1)}, \tag{6.45}$$

$$\boldsymbol{y}^{(3)} = \boldsymbol{f}_{\text{P}}(\boldsymbol{y}^{(2)}) = \frac{\boldsymbol{y}^{(2)}}{z^{(2)}}, \tag{6.46}$$

$$\boldsymbol{x}_i = \boldsymbol{f}_{\text{C}}(\boldsymbol{y}^{(3)}; \boldsymbol{k}) = \boldsymbol{K}(\boldsymbol{k})\,\boldsymbol{y}^{(3)}. \tag{6.47}$$

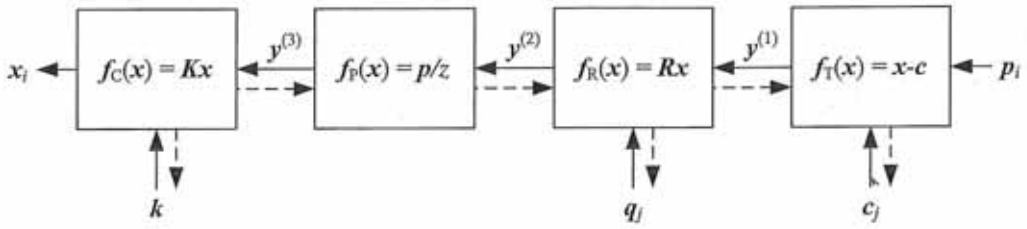


Figure 6.5 A set of chained transforms for projecting a 3D point $\boldsymbol{p}_i$ to a 2D measurement $\boldsymbol{x}_i$ through a series of transformations $\boldsymbol{f}^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

Note that in these equations, we have indexed the camera centers $\boldsymbol{c}_j$ and camera rotation quaternions $\boldsymbol{q}_j$ by an index j, in case more than one pose of the calibration object is being used (see also Section 7.4.) We are also using the camera center $\boldsymbol{c}_j$ instead of the world translation $\boldsymbol{t}_j$, since this is a more natural parameter to estimate.

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

$$\frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{p}^{(k)}} = \frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{y}^{(k)}} \frac{\partial \boldsymbol{y}^{(k)}}{\partial \boldsymbol{p}^{(k)}}, \tag{6.48}$$

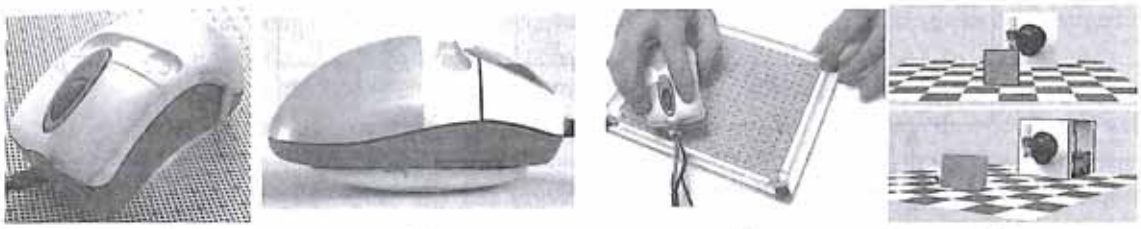
where $\boldsymbol{p}^{(k)}$ indicates one of the parameter vectors that is being optimized. (This same "trick" is used in neural networks as part of the *backpropagation* algorithm (Bishop 2006).)

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

6.2.3 *Application*: Augmented reality

A widely used application of pose estimation is *augmented reality*, where virtual 3D images or annotations are superimposed on top of a live video feed, either through the use of see-through glasses (a head-mounted display) or on a regular computer or mobile device screen (Azuma, Baillot, Behringer *et al.* 2001; Haller, Billinghurst, and Thomas 2007). In some applications, a special pattern printed on cards or in a book is tracked to perform the augmentation (Kato, Billinghurst, Poupyrev *et al.* 2000; Billinghurst, Kato, and Poupyrev 2001). For a desktop application, a grid of dots printed on a mouse pad can be tracked by a camera embedded in an augmented mouse to give the user control of a full six degrees of freedom over their position and orientation in a 3D space (Hinckley, Sinclair, Hanson *et al.* 1999), as shown in Figure 6.6.

Sometimes, the scene itself provides a convenient object to track, such as the rectangle defining a desktop used in *through-the-lens camera control* (Gleicher and Witkin 1992). In



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

Figure 6.6 The VideoMouse can sense six degrees of freedom relative to a specially printed mouse pad using its embedded camera (Hinckley, Sinclair, Hanson *et al.* 1999) © 1999 ACM: (a) top view of the mouse; (b) view of the mouse showing the curved base for rocking; (c) moving the mouse pad with the other hand extends the interaction capabilities; (d) the resulting movement seen on the screen.

outdoor locations, such as film sets, it is more common to place special markers such as brightly colored balls in the scene to make it easier to find and track them (Bogart 1991). In older applications, surveying techniques were used to determine the locations of these balls before filming. Today, it is more common to apply structure-from-motion directly to the film footage itself (Section 7.4.2).

Rapid pose estimation is also central to tracking the position and orientation of the hand-held remote controls used in Nintendo's Wii game systems. A high-speed camera embedded in the remote control is used to track the locations of the infrared (IR) LEDs in the bar that is mounted on the TV monitor. Pose estimation is then used to infer the remote control's location and orientation at very high frame rates. The Wii system can be extended to a variety of other user interaction applications by mounting the bar on a hand-held device, as described by Johnny Lee.[11]

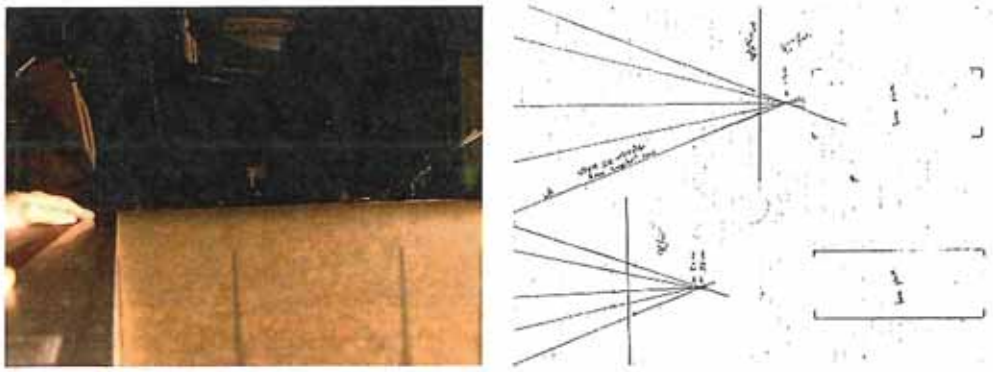
Exercises 6.4 and 6.5 have you implement two different tracking and pose estimation systems for augmented-reality applications. The first system tracks the outline of a rectangular object, such as a book cover or magazine page, and the second has you track the pose of a hand-held Rubik's cube.

6.3 Geometric intrinsic calibration

As described above in Equations (6.42–6.43), the computation of the internal (intrinsic) camera calibration parameters can occur simultaneously with the estimation of the (extrinsic) pose of the camera with respect to a known calibration target. This, indeed, is the "classic" approach to camera calibration used in both the photogrammetry (Slama 1980) and the computer vision (Tsai 1987) communities. In this section, we look at alternative formulations (which may not involve the full solution of a non-linear regression problem), the use of alternative calibration targets, and the estimation of the non-linear part of camera optics such as radial distortion.[12]

[11] http://johnnylee.net/projects/wii/.

[12] In some applications, you can use the EXIF tags associated with a JPEG image to obtain a rough estimate of a camera's focal length but this technique should be used with caution as the results are often inaccurate.



(a) (b)

Figure 6.7 Calibrating a lens by drawing straight lines on cardboard (Debevec, Wenger, Tchou *et al.* 2002) © 2002 ACM: (a) an image taken by the video camera showing a hand holding a metal ruler whose right edge appears vertical in the image; (b) the set of lines drawn on the cardboard converging on the front nodal point (center of projection) of the lens and indicating the horizontal field of view.

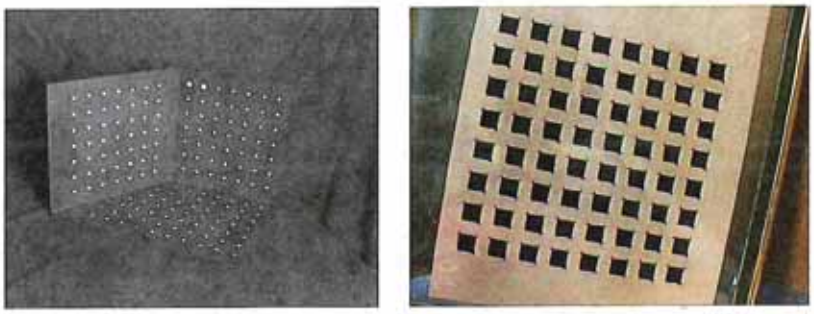
6.3.1 Calibration patterns

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

If a smaller calibration rig needs to be used, e.g., for indoor robotics applications or for mobile robots that carry their own calibration target, it is best if the calibration object can span as much of the workspace as possible (Figure 6.8a), as planar targets often fail to accurately predict the components of the pose that lie far away from the plane. A good way to determine if the calibration has been successfully performed is to estimate the covariance in the parameters (Section 6.1.4) and then project 3D points from various points in the workspace into the image in order to estimate their 2D positional uncertainty.

An alternative method for estimating the focal length and center of projection of a lens is to place the camera on a large flat piece of cardboard and use a long metal ruler to draw lines on the cardboard that appear vertical in the image, as shown in Figure 6.7a (Debevec, Wenger, Tchou *et al.* 2002). Such lines lie on planes that are parallel to the vertical axis of the camera sensor and also pass through the lens' front nodal point. The location of the nodal point (projected vertically onto the cardboard plane) and the horizontal field of view (determined from lines that graze the left and right edges of the visible image) can be recovered by intersecting these lines and measuring their angular extent (Figure 6.7b).

If no calibration pattern is available, it is also possible to perform calibration simultaneously with structure and pose recovery (Sections 6.3.4 and 7.4), which is known as *self-calibration* (Faugeras, Luong, and Maybank 1992; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). However, such an approach requires a large amount of imagery to be accurate.



(a) (b)

Figure 6.8 Calibration patterns: (a) a three-dimensional target (Quan and Lan 1999) © 1999 IEEE; (b) a two-dimensional target (Zhang 2000) © 2000 IEEE. Note that radial distortion needs to be removed from such images before the feature points can be used for calibration.

Planar calibration patterns

When a finite workspace is being used and accurate machining and motion control platforms are available, a good way to perform calibration is to move a planar calibration target in a controlled fashion through the workspace volume. This approach is sometimes called the *N-planes* calibration approach (Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée, Szeliski *et al.* 1992; Grossberg and Nayar 2001) and has the advantage that each camera pixel can be mapped to a unique 3D ray in space, which takes care of both linear effects modeled by the calibration matrix $\boldsymbol{K}$ and non-linear effects such as radial distortion (Section 6.3.5).

A less cumbersome but also less accurate calibration can be obtained by waving a planar calibration pattern in front of a camera (Figure 6.8b). In this case, the pattern's pose has (in principle) to be recovered in conjunction with the intrinsics. In this technique, each input image is used to compute a separate homography (6.19–6.23) $\tilde{\boldsymbol{H}}$ mapping the plane's calibration points $(X_i, Y_i, 0)$ into image coordinates (x_i, y_i),

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\boldsymbol{H}} \boldsymbol{p}_i, \tag{6.49}$$

where the $\boldsymbol{r}_i$ are the first two columns of $\boldsymbol{R}$ and $\sim$ indicates equality up to scale. From these, Zhang (2000) shows how to form linear constraints on the nine entries in the $\boldsymbol{B} = \boldsymbol{K}^{-T}\boldsymbol{K}^{-1}$ matrix, from which the calibration matrix $\boldsymbol{K}$ can be recovered using a matrix square root and inversion. (The matrix $\boldsymbol{B}$ is known as the *image of the absolute conic* (IAC) in projective geometry and is commonly used for camera calibration (Hartley and Zisserman 2004, Section 7.5).) If only the focal length is being recovered, the even simpler approach of using vanishing points can be used instead.

6.3.2 Vanishing points

A common case for calibration that occurs often in practice is when the camera is looking at a man-made scene with strong extended rectahedral objects such as boxes or room walls. In this case, we can intersect the 2D lines corresponding to 3D parallel lines to compute their

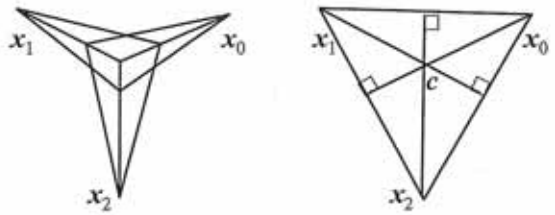


Figure 6.9 Calibration from vanishing points: (a) any pair of finite vanishing points $(\hat{x}_i, \hat{x}_j)$ can be used to estimate the focal length; (b) the orthocenter of the vanishing point triangle gives the optical center of the image $\boldsymbol{c}$.

vanishing points, as described in Section 4.3.3, and use these to determine the intrinsic and extrinsic calibration parameters (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008).

Let us assume that we have detected two or more orthogonal vanishing points, all of which are *finite*, i.e., they are not obtained from lines that appear to be parallel in the image plane (Figure 6.9a). Let us also assume a simplified form for the calibration matrix $\boldsymbol{K}$ where only the focal length is unknown (2.59). (It is often safe for rough 3D modeling to assume that the optical center is at the center of the image, that the aspect ratio is 1, and that there is no skew.) In this case, the projection equation for the vanishing points can be written as

$$\hat{\boldsymbol{x}}_i = \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ f \end{bmatrix} \sim \boldsymbol{R}\boldsymbol{p}_i = \boldsymbol{r}_i, \tag{6.50}$$

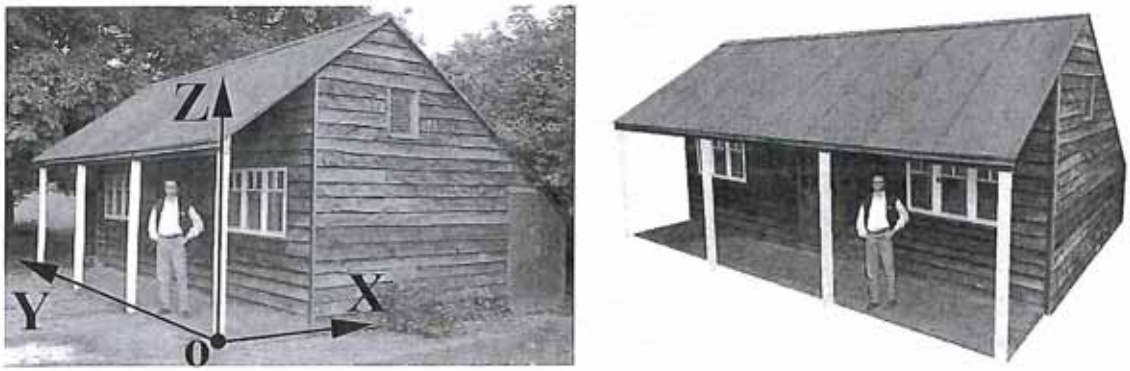
where $\boldsymbol{p}_i$ corresponds to one of the cardinal directions $(1, 0, 0)$, $(0, 1, 0)$, or $(0, 0, 1)$, and $\boldsymbol{r}_i$ is the ith column of the rotation matrix $\boldsymbol{R}$.

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

$$\boldsymbol{r}_i \cdot \boldsymbol{r}_j \sim (x_i - c_x)(x_j - c_y) + (y_i - c_y)(y_j - c_y) + f^2 = 0 \tag{6.51}$$

from which we can obtain an estimate for f^2. Note that the accuracy of this estimate increases as the vanishing points move closer to the center of the image. In other words, it is best to tilt the calibration pattern a decent amount around the 45° axis, as in Figure 6.9a. Once the focal length f has been determined, the individual columns of $\boldsymbol{R}$ can be estimated by normalizing the left hand side of (6.50) and taking cross products. Alternatively, an SVD of the initial $\boldsymbol{R}$ estimate, which is a variant on orthogonal Procrustes (6.32), can be used.

If all three vanishing points are visible and finite in the same image, it is also possible to estimate the optical center as the orthocenter of the triangle formed by the three vanishing points (Caprile and Torre 1990; Hartley and Zisserman 2004, Section 7.6) (Figure 6.9b). In practice, however, it is more accurate to re-estimate any unknown intrinsic calibration parameters using non-linear least squares (6.42).



(a) (b)

Figure 6.10 Single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer: (a) input image showing the three coordinate axes computed from the two horizontal vanishing points (which can be determined from the sidings on the shed); (b) a new view of the 3D reconstruction.

6.3.3 *Application*: Single view metrology

A fun application of vanishing point estimation and camera calibration is the *single view metrology* system developed by Criminisi, Reid, and Zisserman (2000). Their system allows people to interactively measure heights and other dimensions as well as to build piecewise-planar 3D models, as shown in Figure 6.10.

The first step in their system is to identify two orthogonal vanishing points on the ground plane and the vanishing point for the vertical direction, which can be done by drawing some parallel sets of lines in the image. (Alternatively, automated techniques such as those discussed in Section 4.3.3 or by Schaffalitzky and Zisserman (2000) could be used.) The user then marks a few dimensions in the image, such as the height of a reference object, and the system can automatically compute the height of another object. Walls and other planar impostors (geometry) can also be sketched and reconstructed.

In the formulation originally developed by Criminisi, Reid, and Zisserman (2000), the system produces an *affine* reconstruction, i.e., one that is only known up to a set of independent scaling factors along each axis. A potentially more useful system can be constructed by assuming that the camera is calibrated up to an unknown focal length, which can be recovered from orthogonal (finite) vanishing directions, as we just described in Section 6.3.2. Once this is done, the user can indicate an origin on the ground plane and another point a known distance away. From this, points on the ground plane can be directly projected into 3D and points above the ground plane, when paired with their ground plane projections, can also be recovered. A fully metric reconstruction of the scene then becomes possible.

Exercise 6.9 has you implement such a system and then use it to model some simple 3D scenes. Section 12.6.1 describes other, potentially multi-view, approaches to architectural reconstruction, including an interactive piecewise-planar modeling system that uses vanishing points to establish 3D line directions and plane normals (Sinha, Steedly, Szeliski *et al.* 2008).

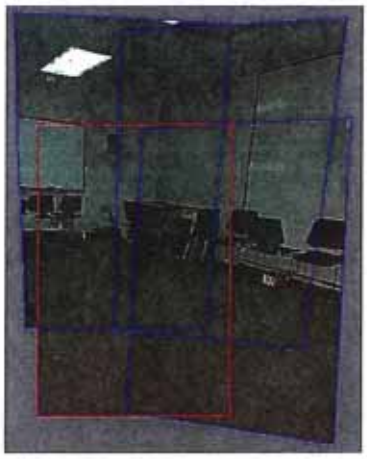


Figure 6.11 Four images taken with a hand-held camera registered using a 3D rotation motion model, which can be used to estimate the focal length of the camera (Szeliski and Shum 1997) © 2000 ACM.

6.3.4 Rotational motion

When no calibration targets or known structures are available but you can rotate the camera around its front nodal point (or, equivalently, work in a large open environment where all objects are distant), the camera can be calibrated from a set of overlapping images by assuming that it is undergoing pure rotational motion, as shown in Figure 6.11 (Stein 1995; Hartley 1997b; Hartley, Hayman, de Agapito *et al.* 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003). When a full 360° motion is used to perform this calibration, a very accurate estimate of the focal length f can be obtained, as the accuracy in this estimate is proportional to the total number of pixels in the resulting cylindrical panorama (Section 9.1.6) (Stein 1995; Shum and Szeliski 2000).

To use this technique, we first compute the homographies $\tilde{\boldsymbol{H}}_{ij}$ between all overlapping pairs of images, as explained in Equations (6.19–6.23). Then, we use the observation, first made in Equation (2.72) and explored in more detail in Section 9.1.3 (9.5), that each homography is related to the inter-camera rotation $\boldsymbol{R}_{ij}$ through the (unknown) calibration matrices $\boldsymbol{K}_i$ and $\boldsymbol{K}_j$,

$$\tilde{\boldsymbol{H}}_{ij} = \boldsymbol{K}_i \boldsymbol{R}_i \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} = \boldsymbol{K}_i \boldsymbol{R}_{ij} \boldsymbol{K}_j^{-1}. \tag{6.52}$$

The simplest way to obtain the calibration is to use the simplified form of the calibration matrix (2.59), where we assume that the pixels are square and the optical center lies at the center of the image, i.e., $\boldsymbol{K}_k = \mathrm{diag}(f_k, f_k, 1)$. (We number the pixel coordinates accordingly, i.e., place pixel $(x, y) = (0, 0)$ at the center of the image.) We can then rewrite Equation (6.52) as

$$\boldsymbol{R}_{10} \sim \boldsymbol{K}_1^{-1} \tilde{\boldsymbol{H}}_{10} \boldsymbol{K}_0 \sim \begin{bmatrix} h_{00} & h_{01} & f_0^{-1} h_{02} \\ h_{10} & h_{11} & f_0^{-1} h_{12} \\ f_1 h_{20} & f_1 h_{21} & f_0^{-1} f_1 h_{22} \end{bmatrix}, \tag{6.53}$$

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

Using the orthonormality properties of the rotation matrix $\boldsymbol{R}_{10}$ and the fact that the right hand side of (6.53) is known only up to a scale, we obtain

$$h_{00}^2 + h_{01}^2 + f_0^{-2} h_{02}^2 = h_{10}^2 + h_{11}^2 + f_0^{-2} h_{12}^2 \tag{6.54}$$

and

$$h_{00}h_{10} + h_{01}h_{11} + f_0^{-2} h_{02}h_{12} = 0. \tag{6.55}$$

From this, we can compute estimates for f_0 of

$$f_0^2 = \frac{h_{12}^2 - h_{02}^2}{h_{00}^2 + h_{01}^2 - h_{10}^2 - h_{11}^2} \text{ if } \mathrm{h}_{00}^2 + \mathrm{h}_{01}^2 \neq \mathrm{h}_{10}^2 + \mathrm{h}_{11}^2 \tag{6.56}$$

or

$$f_0^2 = -\frac{h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \text{ if } \mathrm{h}_{00}\mathrm{h}_{10} \neq -\mathrm{h}_{01}\mathrm{h}_{11}. \tag{6.57}$$

(Note that the equations originally given by Szeliski and Shum (1997) are erroneous; the correct equations are given by Shum and Szeliski (2000).) If neither of these conditions holds, we can also take the dot products between the first (or second) row and the third one. Similar results can be obtained for f_1 as well, by analyzing the columns of $\tilde{\boldsymbol{H}}_{10}$. If the focal length is the same for both images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, e.g., from different homographies, the median value can be used as the final estimate.

A more general (upper-triangular) estimate of $\boldsymbol{K}$ can be obtained in the case of a fixed-parameter camera $\boldsymbol{K}_i = \boldsymbol{K}$ using the technique of Hartley (1997b). Observe from (6.52) that $\boldsymbol{R}_{ij} \sim \boldsymbol{K}^{-1}\tilde{\boldsymbol{H}}_{ij}\boldsymbol{K}$ and $\boldsymbol{R}_{ij}^{-T} \sim \boldsymbol{K}^T \tilde{\boldsymbol{H}}_{ij}^{-T} \boldsymbol{K}^{-T}$. Equating $\boldsymbol{R}_{ij} = \boldsymbol{R}_{ij}^{-T}$ we obtain $\boldsymbol{K}^{-1}\tilde{\boldsymbol{H}}_{ij}\boldsymbol{K} \sim \boldsymbol{K}^T \tilde{\boldsymbol{H}}_{ij}^{-T} \boldsymbol{K}^{-T}$, from which we get

$$\tilde{\boldsymbol{H}}_{ij}(\boldsymbol{K}\boldsymbol{K}^T) \sim (\boldsymbol{K}\boldsymbol{K}^T)\tilde{\boldsymbol{H}}_{ij}^{-T}. \tag{6.58}$$

This provides us with some homogeneous linear constraints on the entries in $\boldsymbol{A} = \boldsymbol{K}\boldsymbol{K}^T$, which is known as the *dual of the image of the absolute conic* (Hartley 1997b; Hartley and Zisserman 2004). (Recall that when we estimate a homography, we can only recover it up to an unknown scale.) Given a sufficient number of independent homography estimates $\tilde{\boldsymbol{H}}_{ij}$, we can recover $\boldsymbol{A}$ (up to a scale) using either SVD or eigenvalue analysis and then recover $\boldsymbol{K}$ through Cholesky decomposition (Appendix A.1.4). Extensions to the cases of temporally varying calibration parameters and non-stationary cameras are discussed by Hartley, Hayman, de Agapito *et al.* (2000) and de Agapito, Hayman, and Reid (2001).

The quality of the intrinsic camera parameters can be greatly increased by constructing a full 360° panorama, since mis-estimating the focal length will result in a gap (or excessive overlap) when the first image in the sequence is stitched to itself (Figure 9.5). The resulting mis-alignment can be used to improve the estimate of the focal length and to re-adjust the rotation estimates, as described in Section 9.1.4. Rotating the camera by 90° around its optic axis and re-shooting the panorama is a good way to check for aspect ratio and skew pixel problems, as is generating a full hemi-spherical panorama when there is sufficient texture.

Ultimately, however, the most accurate estimate of the calibration parameters (including radial distortion) can be obtained using a full simultaneous non-linear minimization of the intrinsic and extrinsic (rotation) parameters, as described in Section 9.2.

6.3.5 Radial distortion

When images are taken with wide-angle lenses, it is often necessary to model *lens distortions* such as *radial distortion*. As discussed in Section 2.1.6, the radial distortion model says that coordinates in the observed images are displaced away from (*barrel* distortion) or towards (*pincushion* distortion) the image center by an amount proportional to their radial distance (Figure 2.13a–b). The simplest radial distortion models use low-order polynomials (c.f. Equation (2.78)),

$$\begin{aligned} \hat{x} &= x(1+\kappa_1 r^2 + \kappa_2 r^4) \\ \hat{y} &= y(1+\kappa_1 r^2 + \kappa_2 r^4), \end{aligned} \tag{6.59}$$

where $r^2 = x^2 + y^2$ and κ_1 and κ_2 are called the *radial distortion parameters* (Brown 1971; Slama 1980).[13]

A variety of techniques can be used to estimate the radial distortion parameters for a given lens.[14] One of the simplest and most useful is to take an image of a scene with a lot of straight lines, especially lines aligned with and near the edges of the image. The radial distortion parameters can then be adjusted until all of the lines in the image are straight, which is commonly called the *plumb-line method* (Brown 1971; Kang 2001; El-Melegy and Farag 2003). Exercise 6.10 gives some more details on how to implement such a technique.

Another approach is to use several overlapping images and to combine the estimation of the radial distortion parameters with the image alignment process, i.e., by extending the pipeline used for stitching in Section 9.2.1. Sawhney and Kumar (1999) use a hierarchy of motion models (translation, affine, projective) in a coarse-to-fine strategy coupled with a quadratic radial distortion correction term. They use direct (intensity-based) minimization to compute the alignment. Stein (1997) uses a feature-based approach combined with a general 3D motion model (and quadratic radial distortion), which requires more matches than a parallax-free rotational panorama but is potentially more general. More recent approaches sometimes simultaneously compute both the unknown intrinsic parameters and the radial distortion coefficients, which may include higher-order terms or more complex rational or non-parametric forms (Claus and Fitzgibbon 2005; Sturm 2005; Thirthala and Pollefeys 2005; Barreto and Daniilidis 2005; Hartley and Kang 2005; Steele and Jaynes 2006; Tardif, Sturm, Trudeau *et al.* 2009).

When a known calibration target is being used (Figure 6.8), the radial distortion estimation can be folded into the estimation of the other intrinsic and extrinsic parameters (Zhang 2000; Hartley and Kang 2007; Tardif, Sturm, Trudeau *et al.* 2009). This can be viewed as adding another stage to the general non-linear minimization pipeline shown in Figure 6.5 between the intrinsic parameter multiplication box $\boldsymbol{f}_{\mathrm{C}}$ and the perspective division box $\boldsymbol{f}_{\mathrm{P}}$. (See Exercise 6.11 on more details for the case of a planar calibration target.)

Of course, as discussed in Section 2.1.6, more general models of lens distortion, such as fisheye and non-central projection, may sometimes be required. While the parameterization of such lenses may be more complicated (Section 2.1.6), the general approach of either using calibration rigs with known 3D positions or self-calibration through the use of multiple

[13] Sometimes the relationship between x and $\hat{x}$ is expressed the other way around, i.e., using primed (final) coordinates on the right-hand side, $x = \hat{x}(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)$. This is convenient if we map image pixels into (warped) rays and then undistort the rays to obtain 3D rays in space, i.e., if we are using inverse warping.

[14] Some of today's digital cameras are starting to remove radial distortion using software in the camera itself.

overlapping images of a scene can both be used (Hartley and Kang 2007; Tardif, Sturm, and Roy 2007). The same techniques used to calibrate for radial distortion can also be used to reduce the amount of chromatic aberration by separately calibrating each color channel and then warping the channels to put them back into alignment (Exercise 6.12).

6.4 Additional reading

Hartley and Zisserman (2004) provide a wonderful introduction to the topics of feature-based alignment and optimal motion estimation, as well as an in-depth discussion of camera calibration and pose estimation techniques.

Techniques for robust estimation are discussed in more detail in Appendix B.3 and in monographs and review articles on this topic (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Rousseeuw and Leroy 1987; Black and Rangarajan 1996; Stewart 1999). The most commonly used robust initialization technique in computer vision is RANdom SAmple Consensus (RANSAC) (Fischler and Bolles 1981), which has spawned a series of more efficient variants (Nistér 2003; Chum and Matas 2005).

The topic of registering 3D point data sets is called *absolute orientation* (Horn 1987) and *3D pose estimation* (Lorusso, Eggert, and Fisher 1995). A variety of techniques has been developed for simultaneously computing 3D point correspondences and their corresponding rigid transformations (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan, Lu *et al.* 1998; David, DeMenthon, Duraiswami *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009).

Camera calibration was first studied in photogrammetry (Brown 1971; Slama 1980; Atkinson 1996; Kraus 1997) but it has also been widely studied in computer vision (Tsai 1987; Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée, Szeliski *et al.* 1992; Zhang 2000; Grossberg and Nayar 2001). Vanishing points observed either from rectahedral calibration objects or man-made architecture are often used to perform rudimentary calibration (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008). Performing camera calibration without using known targets is known as *self-calibration* and is discussed in textbooks and surveys on structure from motion (Faugeras, Luong, and Maybank 1992; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). One popular subset of such techniques uses pure rotational motion (Stein 1995; Hartley 1997b; Hartley, Hayman, de Agapito *et al.* 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003).

6.5 Exercises

Ex 6.1: Feature-based image alignment for flip-book animations Take a set of photos of an action scene or portrait (preferably in motor-drive—continuous shooting—mode) and align them to make a composite or flip-book animation.

1. Extract features and feature descriptors using some of the techniques described in Sections 4.1.1–4.1.2.

2. Match your features using nearest neighbor matching with a nearest neighbor distance ratio test (4.18).
3. Compute an optimal 2D translation and rotation between the first image and all subsequent images, using least squares (Section 6.1.1) with optional RANSAC for robustness (Section 6.1.4).
4. Resample all of the images onto the first image's coordinate frame (Section 3.6.1) using either bilinear or bicubic resampling and optionally crop them to their common area.
5. Convert the resulting images into an animated GIF (using software available from the Web) or optionally implement cross-dissolves to turn them into a "slo-mo" video.
6. (Optional) Combine this technique with feature-based (Exercise 3.25) morphing.

Ex 6.2: Panography Create the kind of panograph discussed in Section 6.1.2 and commonly found on the Web.

1. Take a series of interesting overlapping photos.
2. Use the feature detector, descriptor, and matcher developed in Exercises 4.1–4.4 (or existing software) to match features among the images.
3. Turn each connected component of matching features into a *track*, i.e., assign a unique index i to each track, discarding any tracks that are inconsistent (contain two different features in the same image).
4. Compute a global translation for each image using Equation (6.12).
5. Since your matches probably contain errors, turn the above least square metric into a robust metric (6.25) and re-solve your system using iteratively reweighted least squares.
6. Compute the size of the resulting composite canvas and resample each image into its final position on the canvas. (Keeping track of bounding boxes will make this more efficient.)
7. Average all of the images, or choose some kind of ordering and implement translucent *over* compositing (3.8).
8. (Optional) Extend your parametric motion model to include rotations and scale, i.e., the similarity transform given in Table 6.1. Discuss how you could handle the case of translations and rotations only (no scale).
9. (Optional) Write a simple tool to let the user adjust the ordering and opacity, and add or remove images.
10. (Optional) Write down a different least squares problem that involves pairwise matching of images. Discuss why this might be better or worse than the global matching formula given in (6.12).

Ex 6.3: 2D rigid/Euclidean matching Several alternative approaches are given in Section 6.1.3 for estimating a 2D rigid (Euclidean) alignment.

1. Implement the various alternatives and compare their accuracy on synthetic data, i.e., random 2D point clouds with noisy feature positions.
2. One approach is to estimate the translations from the centroids and then estimate rotation in polar coordinates. Do you need to weight the angles obtained from a polar decomposition in some way to get the statistically correct estimate?
3. How can you modify your techniques to take into account either scalar (6.10) or full two-dimensional point covariance weightings (6.11)? Do all of the previously developed "shortcuts" still work or does full weighting require iterative optimization?

Ex 6.4: 2D match move/augmented reality Replace a picture in a magazine or a book with a different image or video.

1. With a webcam, take a picture of a magazine or book page.
2. Outline a figure or picture on the page with a rectangle, i.e., draw over the four sides as they appear in the image.
3. Match features in this area with each new image frame.
4. Replace the original image with an "advertising" insert, warping the new image with the appropriate homography.
5. Try your approach on a clip from a sporting event (e.g., indoor or outdoor soccer) to implement a billboard replacement.

Ex 6.5: 3D joystick Track a Rubik's cube to implement a 3D joystick/mouse control.

1. Get out an old Rubik's cube (or get one from your parents).
2. Write a program to detect the center of each colored square.
3. Group these centers into lines and then find the vanishing points for each face.
4. Estimate the rotation angle and focal length from the vanishing points.
5. Estimate the full 3D pose (including translation) by finding one or more 3×3 grids and recovering the plane's full equation from this known homography using the technique developed by Zhang (2000).
6. Alternatively, since you already know the rotation, simply estimate the unknown translation from the known 3D corner points on the cube and their measured 2D locations using either linear or non-linear least squares.
7. Use the 3D rotation and position to control a VRML or 3D game viewer.

Ex 6.6: Rotation-based calibration Take an outdoor or indoor sequence from a rotating camera with very little parallax and use it to calibrate the focal length of your camera using the techniques described in Section 6.3.4 or Sections 9.1.3–9.2.1.

1. Take out any radial distortion in the images using one of the techniques from Exercises 6.10–6.11 or using parameters supplied for a given camera by your instructor.

2. Detect and match feature points across neighboring frames and chain them into feature tracks.
3. Compute homographies between overlapping frames and use Equations (6.56–6.57) to get an estimate of the focal length.
4. Compute a full 360° panorama and update your focal length estimate to close the gap (Section 9.1.4).
5. (Optional) Perform a complete bundle adjustment in the rotation matrices and focal length to obtain the highest quality estimate (Section 9.2.1).

Ex 6.7: Target-based calibration Use a three-dimensional target to calibrate your camera.

1. Construct a three-dimensional calibration pattern with known 3D locations. It is not easy to get high accuracy unless you use a machine shop, but you can get close using heavy plywood and printed patterns.
2. Find the corners, e.g, using a line finder and intersecting the lines.
3. Implement one of the iterative calibration and pose estimation algorithms described in Tsai (1987); Bogart (1991); Gleicher and Witkin (1992) or the system described in Section 6.2.2.
4. Take many pictures at different distances and orientations relative to the calibration target and report on both your re-projection errors and accuracy. (To do the latter, you may need to use simulated data.)

Ex 6.8: Calibration accuracy Compare the three calibration techniques (plane-based, rotation-based, and 3D-target-based).

One approach is to have a different student implement each one and to compare the results. Another approach is to use synthetic data, potentially re-using the software you developed for Exercise 2.3. The advantage of using synthetic data is that you know the ground truth for the calibration and pose parameters, you can easily run lots of experiments, and you can synthetically vary the noise in your measurements.

Here are some possible guidelines for constructing your test sets:

1. Assume a medium-wide focal length (say, 50° field of view).
2. For the plane-based technique, generate a 2D grid target and project it at different inclinations.
3. For a 3D target, create an inner cube corner and position it so that it fills most of field of view.
4. For the rotation technique, scatter points uniformly on a sphere until you get a similar number of points as for other techniques.

Before comparing your techniques, predict which one will be the most accurate (normalize your results by the square root of the number of points used).

Add varying amounts of noise to your measurements and describe the noise sensitivity of your various techniques.

Ex 6.9: Single view metrology Implement a system to measure dimensions and reconstruct a 3D model from a single image of a man-made scene using visible vanishing directions (Section 6.3.3) (Criminisi, Reid, and Zisserman 2000).

1. Find the three orthogonal vanishing points from parallel lines and use them to establish the three coordinate axes (rotation matrix $\boldsymbol{R}$ of the camera relative to the scene). If two of the vanishing points are finite (not at infinity), use them to compute the focal length, assuming a known optical center. Otherwise, find some other way to calibrate your camera; you could use some of the techniques described by Schaffalitzky and Zisserman (2000).

2. Click on a ground plane point to establish your origin and click on a point a known distance away to establish the scene scale. This lets you compute the translation $\boldsymbol{t}$ between the camera and the scene. As an alternative, click on a pair of points, one on the ground plane and one above it, and use the known height to establish the scene scale.

3. Write a user interface that lets you click on ground plane points to recover their 3D locations. (Hint: you already know the camera matrix, so knowledge of a point's z value is sufficient to recover its 3D location.) Click on pairs of points (one on the ground plane, one above it) to measure vertical heights.

4. Extend your system to let you draw quadrilaterals in the scene that correspond to axis-aligned rectangles in the world, using some of the techniques described by Sinha, Steedly, Szeliski *et al.* (2008). Export your 3D rectangles to a VRML or PLY[15] file.

5. (Optional) Warp the pixels enclosed by the quadrilateral using the correct homography to produce a texture map for each planar polygon.

Ex 6.10: Radial distortion with plumb lines Implement a plumb-line algorithm to determine the radial distortion parameters.

1. Take some images of scenes with lots of straight lines, e.g., hallways in your home or office, and try to get some of the lines as close to the edges of the image as possible.

2. Extract the edges and link them into curves, as described in Section 4.2.2 and Exercise 4.8.

3. Fit quadratic or elliptic curves to the linked edges using a generalization of the successive line approximation algorithm described in Section 4.3.1 and Exercise 4.11 and keep the curves that fit this form well.

4. For each curved segment, fit a straight line and minimize the perpendicular distance between the curve and the line while adjusting the radial distortion parameters.

5. Alternate between re-fitting the straight line and adjusting the radial distortion parameters until convergence.

[15] http://meshlab.sf.net.

Ex 6.11: Radial distortion with a calibration target Use a grid calibration target to determine the radial distortion parameters.

1. Print out a planar calibration target, mount it on a stiff board, and get it to fill your field of view.
2. Detect the squares, lines, or dots in your calibration target.
3. Estimate the homography mapping the target to the camera from the central portion of the image that does not have any radial distortion.
4. Predict the positions of the remaining targets and use the differences between the observed and predicted positions to estimate the radial distortion.
5. (Optional) Fit a general spline model (for severe distortion) instead of the quartic distortion model.
6. (Optional) Extend your technique to calibrate a fisheye lens.

Ex 6.12: Chromatic aberration Use the radial distortion estimates for each color channel computed in the previous exercise to clean up wide-angle lens images by warping all of the channels into alignment. (Optional) Straighten out the images at the same time.

Can you think of any reasons why this warping strategy may not always work?

Chapter 9

Image stitching

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_9, © Springer-Verlag London Limited 2011

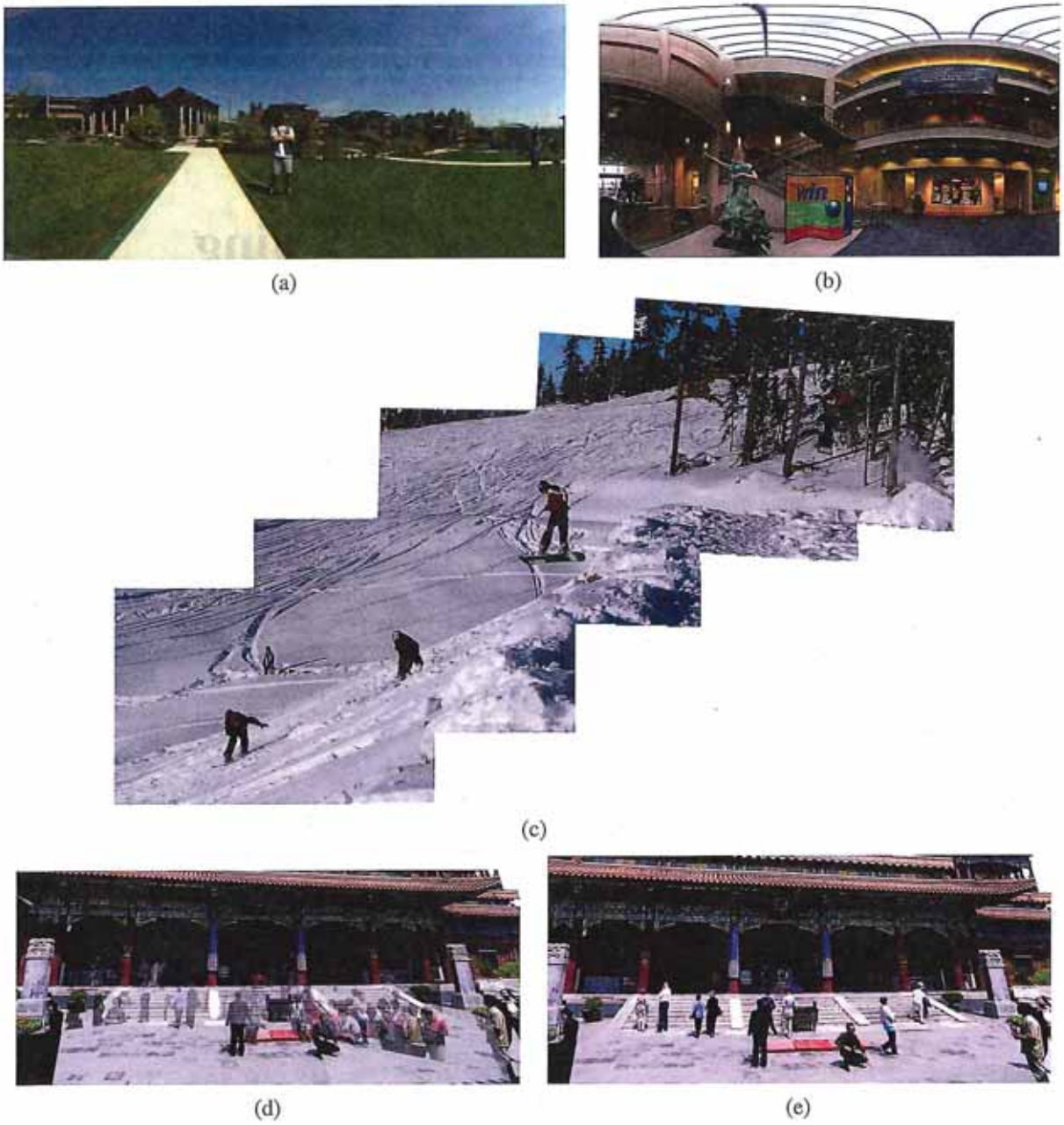


Figure 9.1 Image stitching: (a) portion of a cylindrical panorama and (b) a spherical panorama constructed from 54 photographs (Szeliski and Shum 1997) © 1997 ACM; (c) a multi-image panorama automatically assembled from an unordered photo collection; a multi-image stitch (d) without and (e) with moving object removal (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE.

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision (Milgram 1975; Peleg 1981). image stitching algorithms create the high-resolution photo-mosaics used to produce today's digital maps and satellite photos. They also come bundled with most digital cameras and can be used to create beautiful ultra wide-angle panoramas.

image stitching originated in the photogrammetry community, where more manually intensive methods based on surveyed *ground control points* or manually registered *tie points* have long been used to register aerial photos into large-scale photo-mosaics (Slama 1980). One of the key advances in this community was the development of *bundle adjustment* algorithms (Section 7.4), which could simultaneously solve for the locations of all of the camera positions, thus yielding globally consistent solutions (Triggs, McLauchlan, Hartley *et al.* 1999). Another recurring problem in creating photo-mosaics is the elimination of visible seams, for which a variety of techniques have been developed over the years (Milgram 1975, 1977; Peleg 1981; Davis 1998; Agarwala, Dontcheva, Agrawala *et al.* 2004)

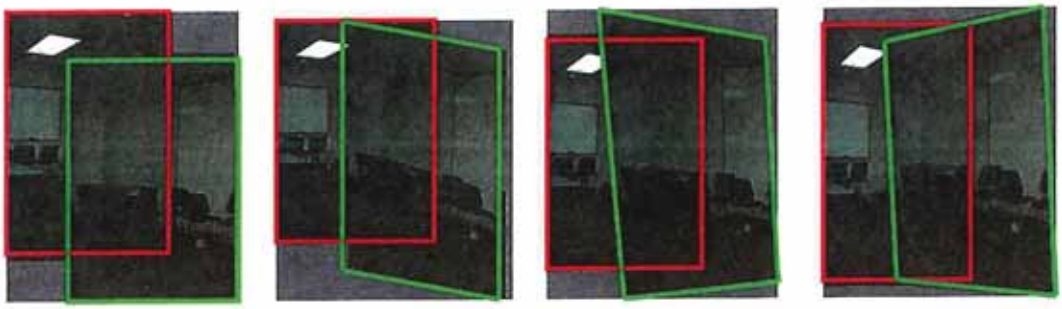
In film photography, special cameras were developed in the 1990s to take ultra-wide-angle panoramas, often by exposing the film through a vertical slit as the camera rotated on its axis (Meehan 1990). In the mid-1990s, image alignment techniques started being applied to the construction of wide-angle seamless panoramas from regular hand-held cameras (Mann and Picard 1994; Chen 1995; Szeliski 1996). More recent work in this area has addressed the need to compute globally consistent alignments (Szeliski and Shum 1997; Sawhney and Kumar 1999; Shum and Szeliski 2000), to remove "ghosts" due to parallax and object movement (Davis 1998; Shum and Szeliski 2000; Uyttendaele, Eden, and Szeliski 2001; Agarwala, Dontcheva, Agrawala *et al.* 2004), and to deal with varying exposures (Mann and Picard 1994; Uyttendaele, Eden, and Szeliski 2001; Levin, Zomet, Peleg *et al.* 2004; Agarwala, Dontcheva, Agrawala *et al.* 2004; Eden, Uyttendaele, and Szeliski 2006; Kopf, Uyttendaele, Deussen *et al.* 2007).[1] These techniques have spawned a large number of commercial stitching products (Chen 1995; Sawhney, Kumar, Gendel *et al.* 1998), of which reviews and comparisons can be found on the Web.[2]

While most of the earlier techniques worked by directly minimizing pixel-to-pixel dissimilarities, more recent algorithms usually extract a sparse set of features and match them to each other, as described in Chapter 4. Such feature-based approaches to image stitching have the advantage of being more robust against scene movement and are potentially faster, if implemented the right way. Their biggest advantage, however, is the ability to "recognize panoramas", i.e., to automatically discover the adjacency (overlap) relationships among an unordered set of images, which makes them ideally suited for fully automated stitching of panoramas taken by casual users (Brown and Lowe 2007).

What, then, are the essential problems in image stitching? As with image alignment, we must first determine the appropriate mathematical model relating pixel coordinates in one image to pixel coordinates in another; Section 9.1 reviews the basic models we have studied and presents some new motion models related specifically to panoramic image stitching. Next, we must somehow estimate the correct alignments relating various pairs (or collections) of images. Chapter 4 discussed how distinctive features can be found in each image and then

[1] A collection of some of these papers was compiled by Benosman and Kang (2001) and they are surveyed by Szeliski (2006a).

[2] The Photosynth Web site, http://photosynth.net, allows people to create and upload panoramas for free.



(a) translation [2 dof] (b) affine [6 dof] (c) perspective [8 dof] (d) 3D rotation [3+ dof]

Figure 9.2 Two-dimensional motion models and how they can be used for image stitching.

efficiently matched to rapidly establish correspondences between pairs of images. Chapter 8 discussed how direct pixel-to-pixel comparisons combined with gradient descent (and other optimization techniques) can also be used to estimate these parameters. When multiple images exist in a panorama, bundle adjustment (Section 7.4) can be used to compute a globally consistent set of alignments and to efficiently discover which images overlap one another. In Section 9.2, we look at how each of these previously developed techniques can be modified to take advantage of the imaging setups commonly used to create panoramas.

Once we have aligned the images, we must choose a final compositing surface for warping the aligned images (Section 9.3.1). We also need algorithms to seamlessly cut and blend overlapping images, even in the presence of parallax, lens distortion, scene motion, and exposure differences (Section 9.3.2–9.3.4).

9.1 Motion models

Before we can register and align images, we need to establish the mathematical relationships that map pixel coordinates from one image to another. A variety of such *parametric motion models* are possible, from simple 2D transforms, to planar perspective models, 3D camera rotations, lens distortions, and mapping to non-planar (e.g., cylindrical) surfaces.

We already covered several of these models in Sections 2.1 and 6.1. In particular, we saw in Section 2.1.5 how the parametric motion describing the deformation of a planar surfaced as viewed from different positions can be described with an eight-parameter homography (2.71) (Mann and Picard 1994; Szeliski 1996). We also saw how a camera undergoing a pure rotation induces a different kind of homography (2.72).

In this section, we review both of these models and show how they can be applied to different stitching situations. We also introduce spherical and cylindrical compositing surfaces and show how, under favorable circumstances, they can be used to perform alignment using pure translations (Section 9.1.6). Deciding which alignment model is most appropriate for a given situation or set of data is a *model selection* problem (Hastie, Tibshirani, and Friedman 2001; Torr 2002; Bishop 2006; Robert 2007), an important topic we do not cover in this book.

9.1.1 Planar perspective motion

The simplest possible motion model to use when aligning images is to simply translate and rotate them in 2D (Figure 9.2a). This is exactly the same kind of motion that you would use if you had overlapping photographic prints. It is also the kind of technique favored by David Hockney to create the collages that he calls *joiners* (Zelnik-Manor and Perona 2007; Nomura, Zhang, and Nayar 2007). Creating such collages, which show visible seams and inconsistencies that add to the artistic effect, is popular on Web sites such as Flickr, where they more commonly go under the name *panography* (Section 6.1.2). Translation and rotation are also usually adequate motion models to compensate for small camera motions in applications such as photo and video stabilization and merging (Exercise 6.1 and Section 8.2.1).

In Section 6.1.3, we saw how the mapping between two cameras viewing a common plane can be described using a 3×3 homography (2.71). Consider the matrix M_{10} that arises when mapping a pixel in one image to a 3D point and then back onto a second image,

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{P}}_1 \tilde{\boldsymbol{P}}_0^{-1} \tilde{\boldsymbol{x}}_0 = \boldsymbol{M}_{10} \tilde{\boldsymbol{x}}_0. \tag{9.1}$$

When the last row of the $\boldsymbol{P}_0$ matrix is replaced with a plane equation $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + c_0$ and points are assumed to lie on this plane, i.e., their disparity is $d_0 = 0$, we can ignore the last column of $\boldsymbol{M}_{10}$ and also its last row, since we do not care about the final z-buffer depth. The resulting homography matrix $\tilde{\boldsymbol{H}}_{10}$ (the upper left 3×3 sub-matrix of $\boldsymbol{M}_{10}$) describes the mapping between pixels in the two images,

$$\tilde{\boldsymbol{x}}_1 \sim \tilde{\boldsymbol{H}}_{10} \tilde{\boldsymbol{x}}_0. \tag{9.2}$$

This observation formed the basis of some of the earliest automated image stitching algorithms (Mann and Picard 1994; Szeliski 1994, 1996). Because reliable feature matching techniques had not yet been developed, these algorithms used direct pixel value matching, i.e., direct parametric motion estimation, as described in Section 8.2 and Equations (6.19–6.20).

More recent stitching algorithms first extract features and then match them up, often using robust techniques such as RANSAC (Section 6.1.4) to compute a good set of inliers. The final computation of the homography (9.2), i.e., the solution of the least squares fitting problem given pairs of corresponding features,

$$x_1 = \frac{(1+h_{00})x_0 + h_{01}y_0 + h_{02}}{h_{20}x_0 + h_{21}y_0 + 1} \text{ and } y_1 = \frac{h_{10}x_0 + (1+h_{11})y_0 + h_{12}}{h_{20}x_0 + h_{21}y_0 + 1}, \tag{9.3}$$

uses iterative least squares, as described in Section 6.1.3 and Equations (6.21–6.23).

9.1.2 *Application*: Whiteboard and document scanning

The simplest image-stitching application is to stitch together a number of image scans taken on a flatbed scanner. Say you have a large map, or a piece of child's artwork, that is too large to fit on your scanner. Simply take multiple scans of the document, making sure to overlap the scans by a large enough amount to ensure that there are enough common features. Next, take successive pairs of images that you know overlap, extract features, match them up, and estimate the 2D rigid transform (2.16),

$$\boldsymbol{x}_{k+1} = \boldsymbol{R}_k \boldsymbol{x}_k + \boldsymbol{t}_k, \tag{9.4}$$

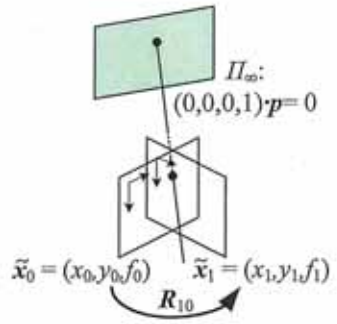


Figure 9.3 Pure 3D camera rotation. The form of the homography (mapping) is particularly simple and depends only on the 3D rotation matrix and focal lengths.

that best matches the features, using two-point RANSAC, if necessary, to find a good set of inliers. Then, on a final compositing surface (aligned with the first scan, for example), resample your images (Section 3.6.1) and average them together. Can you see any potential problems with this scheme?

One complication is that a 2D rigid transformation is non-linear in the rotation angle θ, so you will have to either use non-linear least squares or constrain $\boldsymbol{R}$ to be orthonormal, as described in Section 6.1.3.

A bigger problem lies in the pairwise alignment process. As you align more and more pairs, the solution may drift so that it is no longer globally consistent. In this case, a global optimization procedure, as described in Section 9.2, may be required. Such global optimization often requires a large system of non-linear equations to be solved, although in some cases, such as linearized homographies (Section 9.1.3) or similarity transforms (Section 6.1.2), regular least squares may be an option.

A slightly more complex scenario is when you take multiple overlapping handheld pictures of a whiteboard or other large planar object (He and Zhang 2005; Zhang and He 2007). Here, the natural motion model to use is a homography, although a more complex model that estimates the 3D rigid motion relative to the plane (plus the focal length, if unknown), could in principle be used.

9.1.3 Rotational panoramas

The most typical case for panoramic image stitching is when the camera undergoes a pure rotation. Think of standing at the rim of the Grand Canyon. Relative to the distant geometry in the scene, as you snap away, the camera is undergoing a pure rotation, which is equivalent to assuming that all points are very far from the camera, i.e., on the *plane at infinity* (Figure 9.3). Setting $\boldsymbol{t}_0 = \boldsymbol{t}_1 = 0$, we get the simplified 3×3 homography

$$\tilde{\boldsymbol{H}}_{10} = \boldsymbol{K}_1 \boldsymbol{R}_1 \boldsymbol{R}_0^{-1} \boldsymbol{K}_0^{-1} = \boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1}, \tag{9.5}$$

where $\boldsymbol{K}_k = \mathrm{diag}(f_k, f_k, 1)$ is the simplified camera intrinsic matrix (2.59), assuming that $c_x = c_y = 0$, i.e., we are indexing the pixels starting from the optical center (Szeliski 1996).

This can also be re-written as

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_1 & & \\ & f_1 & \\ & & 1 \end{bmatrix} \boldsymbol{R}_{10} \begin{bmatrix} f_0^{-1} & & \\ & f_0^{-1} & \\ & & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{9.6}$$

or

$$\begin{bmatrix} x_1 \\ y_1 \\ f_1 \end{bmatrix} \sim \boldsymbol{R}_{10} \begin{bmatrix} x_0 \\ y_0 \\ f_0 \end{bmatrix}, \tag{9.7}$$

which reveals the simplicity of the mapping equations and makes all of the motion parameters explicit. Thus, instead of the general eight-parameter homography relating a pair of images, we get the three-, four-, or five-parameter *3D rotation* motion models corresponding to the cases where the focal length f is known, fixed, or variable (Szeliski and Shum 1997).[3] Estimating the 3D rotation matrix (and, optionally, focal length) associated with each image is intrinsically more stable than estimating a homography with a full eight degrees of freedom, which makes this the method of choice for large-scale image stitching algorithms (Szeliski and Shum 1997; Shum and Szeliski 2000; Brown and Lowe 2007).

Given this representation, how do we update the rotation matrices to best align two overlapping images? Given a current estimate for the homography $\tilde{\boldsymbol{H}}_{10}$ in (9.5), the best way to update $\boldsymbol{R}_{10}$ is to prepend an *incremental* rotation matrix $\boldsymbol{R}(\boldsymbol{\omega})$ to the current estimate $\boldsymbol{R}_{10}$ (Szeliski and Shum 1997; Shum and Szeliski 2000),

$$\tilde{\boldsymbol{H}}(\boldsymbol{\omega}) = \boldsymbol{K}_1 \boldsymbol{R}(\boldsymbol{\omega}) \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1} = [\boldsymbol{K}_1 \boldsymbol{R}(\boldsymbol{\omega}) \boldsymbol{K}_1^{-1}][\boldsymbol{K}_1 \boldsymbol{R}_{10} \boldsymbol{K}_0^{-1}] = \boldsymbol{D} \tilde{\boldsymbol{H}}_{10}. \tag{9.8}$$

Note that here we have written the update rule in the *compositional* form, where the incremental update $\boldsymbol{D}$ is *prepended* to the current homography $\tilde{\boldsymbol{H}}_{10}$. Using the small-angle approximation to $\boldsymbol{R}(\boldsymbol{\omega})$ given in (2.35), we can write the incremental update matrix as

$$\boldsymbol{D} = \boldsymbol{K}_1 \boldsymbol{R}(\boldsymbol{\omega}) \boldsymbol{K}_1^{-1} \approx \boldsymbol{K}_1 (\boldsymbol{I} + [\boldsymbol{\omega}]_\times) \boldsymbol{K}_1^{-1} = \begin{bmatrix} 1 & -\omega_z & f_1 \omega_y \\ \omega_z & 1 & -f_1 \omega_x \\ -\omega_y / f_1 & \omega_x / f_1 & 1 \end{bmatrix}. \tag{9.9}$$

Notice how there is now a nice one-to-one correspondence between the entries in the $\boldsymbol{D}$ matrix and the $h_{00}, \ldots, h_{21}$ parameters used in Table 6.1 and Equation (6.19), i.e.,

$$(h_{00}, h_{01}, h_{02}, h_{00}, h_{11}, h_{12}, h_{20}, h_{21}) = (0, -\omega_z, f_1 \omega_y, \omega_z, 0, -f_1 \omega_x, -\omega_y / f_1, \omega_x / f_1). \tag{9.10}$$

We can therefore apply the chain rule to Equations (6.24 and 9.10) to obtain

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} -xy/f_1 & f_1 + x^2/f_1 & -y \\ -(f_1 + y^2/f_1) & xy/f_1 & x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{9.11}$$

which give us the linearized update equations needed to estimate $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$.[4] Notice that this update rule depends on the focal length f_1 of the *target* view and is independent

[3] An initial estimate of the focal lengths can be obtained using the intrinsic calibration techniques described in Section 6.3.4 or from EXIF tags.

[4] This is the same as the rotational component of instantaneous rigid flow (Bergen, Anandan, Hanna *et al.* 1992) and the update equations given by Szeliski and Shum (1997) and Shum and Szeliski (2000).

Figure 9.4 Four images taken with a hand-held camera registered using a 3D rotation motion model (Szeliski and Shum 1997) © 1997 ACM. Notice how the homographies, rather than being arbitrary, have a well-defined keystone shape whose width increases away from the origin.

of the focal length f_0 of the *template* view. This is because the compositional algorithm essentially makes small perturbations to the target. Once the incremental rotation vector $\boldsymbol{\omega}$ has been computed, the $\boldsymbol{R}_1$ rotation matrix can be updated using $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}(\boldsymbol{\omega})\boldsymbol{R}_1$.

The formulas for updating the focal length estimates are a little more involved and are given in (Shum and Szeliski 2000). We will not repeat them here, since an alternative update rule, based on minimizing the difference between back-projected 3D rays, is given in Section 9.2.1. Figure 9.4 shows the alignment of four images under the 3D rotation motion model.

9.1.4 Gap closing

The techniques presented in this section can be used to estimate a series of rotation matrices and focal lengths, which can be chained together to create large panoramas. Unfortunately, because of accumulated errors, this approach will rarely produce a closed 360° panorama. Instead, there will invariably be either a gap or an overlap (Figure 9.5).

We can solve this problem by matching the first image in the sequence with the last one. The difference between the two rotation matrix estimates associated with the repeated first indicates the amount of misregistration. This error can be distributed evenly across the whole sequence by taking the quotient of the two quaternions associated with these rotations and dividing this "error quaternion" by the number of images in the sequence (assuming relatively constant inter-frame rotations). We can also update the estimated focal length based on the amount of misregistration. To do this, we first convert the error quaternion into a *gap angle*, θ_g and then update the focal length using the equation $f' = f(1 - \theta_g/360°)$.

Figure 9.5a shows the end of registered image sequence and the first image. There is a big gap between the last image and the first which are in fact the same image. The gap is 32° because the wrong estimate of focal length ($f = 510$) was used. Figure 9.5b shows the registration after closing the gap with the correct focal length ($f = 468$). Notice that both

(a) (b)

Figure 9.5 Gap closing (Szeliski and Shum 1997) © 1997 ACM: (a) A gap is visible when the focal length is wrong ($f = 510$). (b) No gap is visible for the correct focal length ($f = 468$).

mosaics show very little visual misregistration (except at the gap), yet Figure 9.5a has been computed using a focal length that has 9% error. Related approaches have been developed by Hartley (1994b), McMillan and Bishop (1995), Stein (1995), and Kang and Weiss (1997) to solve the focal length estimation problem using pure panning motion and cylindrical images.

Unfortunately, this particular gap-closing heuristic only works for the kind of "one-dimensional" panorama where the camera is continuously turning in the same direction. In Section 9.2, we describe a different approach to removing gaps and overlaps that works for arbitrary camera motions.

9.1.5 *Application*: Video summarization and compression

An interesting application of image stitching is the ability to summarize and compress videos taken with a panning camera. This application was first suggested by Teodosio and Bender (1993), who called their mosaic-based summaries *salient stills*. These ideas were then extended by Irani, Hsu, and Anandan (1995), Kumar, Anandan, Irani *et al.* (1995), and Irani and Anandan (1998) to additional applications, such as video compression and video indexing. While these early approaches used affine motion models and were therefore restricted to long focal lengths, the techniques were generalized by Lee, ge Chen, lung Bruce Lin *et al.* (1997) to full eight-parameter homographies and incorporated into the MPEG-4 video compression standard, where the stitched background layers were called *video sprites* (Figure 9.6).

While video stitching is in many ways a straightforward generalization of multiple-image stitching (Steedly, Pal, and Szeliski 2005; Baudisch, Tan, Steedly *et al.* 2006), the potential presence of large amounts of independent motion, camera zoom, and the desire to visualize dynamic events impose additional challenges. For example, moving foreground objects can often be removed using *median filtering*. Alternatively, foreground objects can be extracted into a separate layer (Sawhney and Ayer 1996) and later composited back into the stitched panoramas, sometimes as multiple instances to give the impressions of a "Chronophotograph" (Massey and Bender 1996) and sometimes as video overlays (Irani and Anandan 1998).

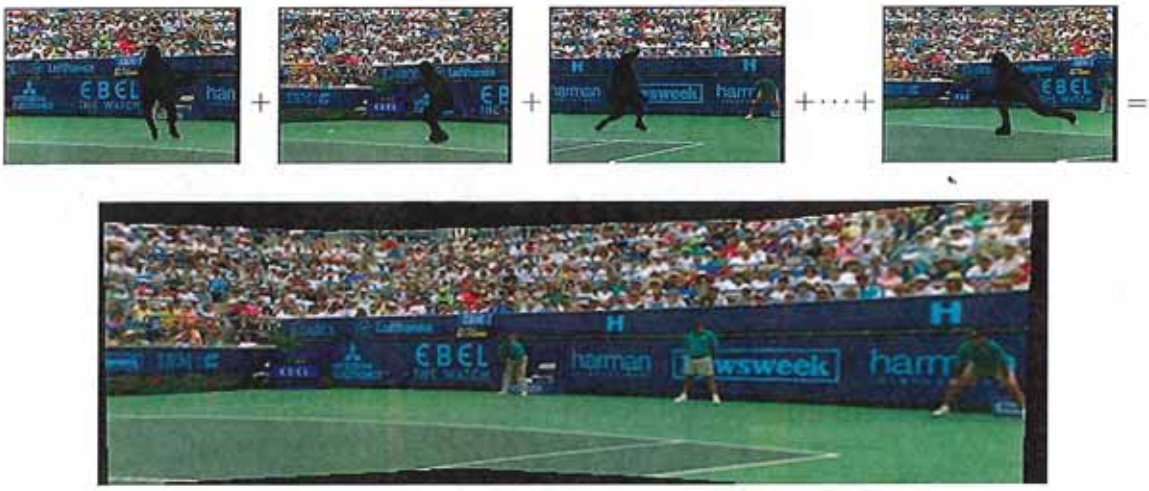
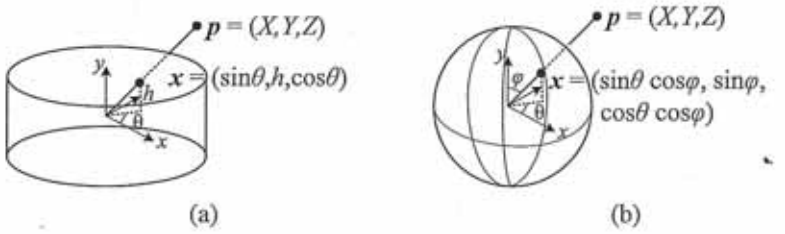


Figure 9.6 Video stitching the background scene to create a single *sprite* image that can be transmitted and used to re-create the background in each frame (Lee, ge Chen, lung Bruce Lin *et al.* 1997) © 1997 IEEE.

Videos can also be used to create animated *panoramic video textures* (Section 13.5.2), in which different portions of a panoramic scene are animated with independently moving video loops (Agarwala, Zheng, Pal *et al.* 2005; Rav-Acha, Pritch, Lischinski *et al.* 2005), or to shine "video flashlights" onto a composite mosaic of a scene (Sawhney, Arpa, Kumar *et al.* 2002).

Video can also provide an interesting source of content for creating panoramas taken from moving cameras. While this invalidates the usual assumption of a single point of view (optical center), interesting results can still be obtained. For example, the VideoBrush system of Sawhney, Kumar, Gendel *et al.* (1998) uses thin strips taken from the center of the image to create a panorama taken from a horizontally moving camera. This idea can be generalized to other camera motions and compositing surfaces using the concept of mosaics on adaptive manifold (Peleg, Rousso, Rav-Acha *et al.* 2000), and also used to generate panoramic stereograms (Peleg, Ben-Ezra, and Pritch 2001). Related ideas have been used to create panoramic matte paintings for multi-plane cel animation (Wood, Finkelstein, Hughes *et al.* 1997), for creating stitched images of scenes with parallax (Kumar, Anandan, Irani *et al.* 1995), and as 3D representations of more complex scenes using *multiple-center-of-projection images* (Rademacher and Bishop 1998) and *multi-perspective panoramas* (Román, Garg, and Levoy 2004; Román and Lensch 2006; Agarwala, Agrawala, Cohen *et al.* 2006).

Another interesting variant on video-based panoramas are *concentric mosaics* (Section 13.3.3) (Shum and He 1999). Here, rather than trying to produce a single panoramic image, the complete original video is kept and used to re-synthesize views (from different camera origins) using ray remapping (light field rendering), thus endowing the panorama with a sense of 3D depth. The same data set can also be used to explicitly reconstruct the depth using multi-baseline stereo (Peleg, Ben-Ezra, and Pritch 2001; Li, Shum, Tang *et al.* 2004; Zheng, Kang, Cohen *et al.* 2007).



(a) (b)

Figure 9.7 Projection from 3D to (a) cylindrical and (b) spherical coordinates.

9.1.6 Cylindrical and spherical coordinates

An alternative to using homographies or 3D motions to align images is to first warp the images into *cylindrical* coordinates and then use a pure translational model to align them (Chen 1995; Szeliski 1996). Unfortunately, this only works if the images are all taken with a level camera or with a known tilt angle.

Assume for now that the camera is in its canonical position, i.e., its rotation matrix is the identity, $\boldsymbol{R} = \boldsymbol{I}$, so that the optical axis is aligned with the z axis and the y axis is aligned vertically. The 3D ray corresponding to an (x, y) pixel is therefore (x, y, f).

We wish to project this image onto a *cylindrical surface* of unit radius (Szeliski 1996). Points on this surface are parameterized by an angle θ and a height h, with the 3D cylindrical coordinates corresponding to (θ, h) given by

$$(\sin\theta, h, \cos\theta) \propto (x, y, f), \tag{9.12}$$

as shown in Figure 9.7a. From this correspondence, we can compute the formula for the *warped* or *mapped* coordinates (Szeliski and Shum 1997),

$$x' = s\theta = s\tan^{-1}\frac{x}{f}, \tag{9.13}$$

$$y' = sh = s\frac{y}{\sqrt{x^2+f^2}}, \tag{9.14}$$

where s is an arbitrary scaling factor (sometimes called the *radius* of the cylinder) that can be set to $s = f$ to minimize the distortion (scaling) near the center of the image.[5] The inverse of this mapping equation is given by

$$x = f\tan\theta = f\tan\frac{x'}{s}, \tag{9.15}$$

$$y = h\sqrt{x^2+f^2} = \frac{y'}{s}f\sqrt{1+\tan^2 x'/s} = f\frac{y'}{s}\sec\frac{x'}{s}. \tag{9.16}$$

Images can also be projected onto a *spherical surface* (Szeliski and Shum 1997), which is useful if the final panorama includes a full sphere or hemisphere of views, instead of just a cylindrical strip. In this case, the sphere is parameterized by two angles (θ, ϕ), with 3D spherical coordinates given by

$$(\sin\theta\cos\phi, \sin\phi, \cos\theta\cos\phi) \propto (x, y, f), \tag{9.17}$$

[5] The scale can also be set to a larger or smaller value for the final compositing surface, depending on the desired output panorama resolution—see Section 9.3.

(a) (b)

Figure 9.8 A cylindrical panorama (Szeliski and Shum 1997) © 1997 ACM: (a) two cylindrically warped images related by a horizontal translation; (b) part of a cylindrical panorama composited from a sequence of images.

as shown in Figure 9.7b.[6] The correspondence between coordinates is now given by (Szeliski and Shum 1997):

$$x' = s\theta = s\tan^{-1}\frac{x}{f}, \tag{9.18}$$

$$y' = s\phi = s\tan^{-1}\frac{y}{\sqrt{x^2+f^2}}, \tag{9.19}$$

while the inverse is given by

$$x = f\tan\theta = f\tan\frac{x'}{s}, \tag{9.20}$$

$$y = \sqrt{x^2+f^2}\tan\phi = \tan\frac{y'}{s} f\sqrt{1+\tan^2 x'/s} = f\tan\frac{y'}{s}\sec\frac{x'}{s}. \tag{9.21}$$

Note that it may be simpler to generate a scaled (x, y, z) direction from Equation (9.17) followed by a perspective division by z and a scaling by f.

Cylindrical image stitching algorithms are most commonly used when the camera is known to be level and only rotating around its vertical axis (Chen 1995). Under these conditions, images at different rotations are related by a pure horizontal translation.[7] This makes it attractive as an initial class project in an introductory computer vision course, since the full complexity of the perspective alignment algorithm (Sections 6.1, 8.2, and 9.1.3) can be avoided. Figure 9.8 shows how two cylindrically warped images from a leveled rotational panorama are related by a pure translation (Szeliski and Shum 1997).

Professional panoramic photographers often use pan-tilt heads that make it easy to control the tilt and to stop at specific *detents* in the rotation angle. Motorized rotation heads are also sometimes used for the acquisition of larger panoramas (Kopf, Uyttendaele, Deussen *et al.* 2007).[8] Not only do they ensure a uniform coverage of the visual field with a desired amount of image overlap but they also make it possible to stitch the images using cylindrical or spherical coordinates and pure translations. In this case, pixel coordinates (x, y, f) must first

[6] Note that these are not the usual spherical coordinates, first presented in Equation (2.8). Here, the y axis points at the north pole instead of the z axis, since we are used to viewing images taken horizontally, i.e., with the y axis pointing in the direction of the gravity vector.

[7] Small vertical tilts can sometimes be compensated for with vertical translations.

[8] See also http://gigapan.org.

Figure 9.9 A spherical panorama constructed from 54 photographs (Szeliski and Shum 1997) © 1997 ACM.

be rotated using the known tilt and panning angles before being projected into cylindrical or spherical coordinates (Chen 1995). Having a roughly known panning angle also makes it easier to compute the alignment, since the rough relative positioning of all the input images is known ahead of time, enabling a reduced search range for alignment. Figure 9.9 shows a full 3D rotational panorama unwrapped onto the surface of a sphere (Szeliski and Shum 1997).

One final coordinate mapping worth mentioning is the *polar* mapping, where the north pole lies along the optical axis rather than the vertical axis,

$$(\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\phi) = s\,(x, y, z). \tag{9.22}$$

In this case, the mapping equations become

$$x' = s\phi\cos\theta = s\frac{x}{r}\tan^{-1}\frac{r}{z}, \tag{9.23}$$

$$y' = s\phi\sin\theta = s\frac{y}{r}\tan^{-1}\frac{r}{z}, \tag{9.24}$$

where $r = \sqrt{x^2 + y^2}$ is the *radial distance* in the (x, y) plane and $s\phi$ plays a similar role in the (x', y') plane. This mapping provides an attractive visualization surface for certain kinds of wide-angle panoramas and is also a good model for the distortion induced by *fisheye lenses*, as discussed in Section 2.1.6. Note how for small values of (x, y), the mapping equations reduce to $x' \approx sx/z$, which suggests that s plays a role similar to the focal length f.

9.2 Global alignment

So far, we have discussed how to register pairs of images using a variety of motion models. In most applications, we are given more than a single pair of images to register. The goal is then to find a globally consistent set of alignment parameters that minimize the mis-registration between all pairs of images (Szeliski and Shum 1997; Shum and Szeliski 2000; Sawhney and Kumar 1999; Coorg and Teller 2000).

In this section, we extend the pairwise matching criteria (6.2, 8.1, and 8.50) to a global energy function that involves all of the per-image pose parameters (Section 9.2.1). Once we have computed the global alignment, we often need to perform *local adjustments*, such as *parallax removal*, to reduce double images and blurring due to local mis-registrations (Section 9.2.2). Finally, if we are given an unordered set of images to register, we need to discover which images go together to form one or more panoramas. This process of *panorama recognition* is described in Section 9.2.3.

9.2.1 Bundle adjustment

One way to register a large number of images is to add new images to the panorama one at a time, aligning the most recent image with the previous ones already in the collection (Szeliski and Shum 1997) and discovering, if necessary, which images it overlaps (Sawhney and Kumar 1999). In the case of 360° panoramas, accumulated error may lead to the presence of a gap (or excessive overlap) between the two ends of the panorama, which can be fixed by stretching the alignment of all the images using a process called *gap closing* (Szeliski and Shum 1997). However, a better alternative is to simultaneously align all the images using a least-squares framework to correctly distribute any mis-registration errors.

The process of simultaneously adjusting pose parameters for a large collection of overlapping images is called *bundle adjustment* in the photogrammetry community (Triggs, McLauchlan, Hartley *et al.* 1999). In computer vision, it was first applied to the general structure from motion problem (Szeliski and Kang 1994) and then later specialized for panoramic image stitching (Shum and Szeliski 2000; Sawhney and Kumar 1999; Coorg and Teller 2000).

In this section, we formulate the problem of global alignment using a feature-based approach, since this results in a simpler system. An equivalent direct approach can be obtained either by dividing images into patches and creating a virtual feature correspondence for each one (as discussed in Section 9.2.4 and by Shum and Szeliski (2000)) or by replacing the per-feature error metrics with per-pixel metrics.

Consider the feature-based alignment problem given in Equation (6.2), i.e.,

$$E_{\text{pairwise-LS}} = \sum_i \|\boldsymbol{r}_i\|^2 = \|\tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i; \boldsymbol{p}) - \hat{\boldsymbol{x}}_i'\|^2. \tag{9.25}$$

For multi-image alignment, instead of having a single collection of pairwise feature correspondences, $\{(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i')\}$, we have a collection of n features, with the location of the ith feature point in the jth image denoted by $\boldsymbol{x}_{ij}$ and its scalar confidence (i.e., inverse variance) denoted by c_{ij}.[9] Each image also has some associated pose parameters.

In this section, we assume that this pose consists of a rotation matrix $\boldsymbol{R}_j$ and a focal length f_j, although formulations in terms of homographies are also possible (Szeliski and Shum 1997; Sawhney and Kumar 1999). The equation mapping a 3D point $\boldsymbol{x}_i$ into a point $\boldsymbol{x}_{ij}$ in frame j can be re-written from Equations (2.68) and (9.5) as

$$\tilde{\boldsymbol{x}}_{ij} \sim \boldsymbol{K}_j \boldsymbol{R}_j \boldsymbol{x}_i \text{ and } \boldsymbol{x}_i \sim \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} \tilde{\boldsymbol{x}}_{ij}, \tag{9.26}$$

[9] Features that are not seen in image j have $c_{ij} = 0$. We can also use 2×2 inverse covariance matrices Σ_{ij}^{-1} in place of c_{ij}, as shown in Equation (6.11).

where $\boldsymbol{K}_j = \text{diag}(f_j, f_j, 1)$ is the simplified form of the calibration matrix. The motion mapping a point $\boldsymbol{x}_{ij}$ from frame j into a point $\boldsymbol{x}_{ik}$ in frame k is similarly given by

$$\tilde{\boldsymbol{x}}_{ik} \sim \tilde{\boldsymbol{H}}_{kj}\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{R}_j^{-1}\boldsymbol{K}_j^{-1}\tilde{\boldsymbol{x}}_{ij}. \tag{9.27}$$

Given an initial set of $\{(\boldsymbol{R}_j, f_j)\}$ estimates obtained from chaining pairwise alignments, how do we refine these estimates?

One approach is to directly extend the pairwise energy $E_{\text{pairwise-LS}}$ (9.25) to a multiview formulation,

$$E_{\text{all-pairs-2D}} = \sum_i \sum_{jk} c_{ij}c_{ik}\|\tilde{\boldsymbol{x}}_{ik}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j, \boldsymbol{R}_k, f_k) - \hat{\boldsymbol{x}}_{ik}\|^2, \tag{9.28}$$

where the $\tilde{\boldsymbol{x}}_{ik}$ function is the *predicted* location of feature i in frame k given by (9.27), $\hat{\boldsymbol{x}}_{ij}$ is the *observed* location, and the "2D" in the subscript indicates that an image-plane error is being minimized (Shum and Szeliski 2000). Note that since $\tilde{\boldsymbol{x}}_{ik}$ depends on the $\hat{\boldsymbol{x}}_{ij}$ observed value, we actually have an *errors-in-variable* problem, which in principle requires more sophisticated techniques than least squares to solve (Van Huffel and Lemmerling 2002; Matei and Meer 2006). However, in practice, if we have enough features, we can directly minimize the above quantity using regular non-linear least squares and obtain an accurate multi-frame alignment.

While this approach works well in practice, it suffers from two potential disadvantages. First, since a summation is taken over all pairs with corresponding features, features that are observed many times are overweighted in the final solution. (In effect, a feature observed m times gets counted $\binom{m}{2}$ times instead of m times.) Second, the derivatives of $\tilde{\boldsymbol{x}}_{ik}$ with respect to the $\{(\boldsymbol{R}_j, f_j)\}$ are a little cumbersome, although using the incremental correction to $\boldsymbol{R}_j$ introduced in Section 9.1.3 makes this more tractable.

An alternative way to formulate the optimization is to use true bundle adjustment, i.e., to solve not only for the pose parameters $\{(\boldsymbol{R}_j, f_j)\}$ but also for the 3D point positions $\{\boldsymbol{x}_i\}$,

$$E_{\text{BA-2D}} = \sum_i \sum_j c_{ij}\|\tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j) - \hat{\boldsymbol{x}}_{ij}\|^2, \tag{9.29}$$

where $\tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j)$ is given by (9.26). The disadvantage of full bundle adjustment is that there are more variables to solve for, so each iteration and also the overall convergence may be slower. (Imagine how the 3D points need to "shift" each time some rotation matrices are updated.) However, the computational complexity of each linearized Gauss–Newton step can be reduced using sparse matrix techniques (Section 7.4.1) (Szeliski and Kang 1994; Triggs, McLauchlan, Hartley *et al.* 1999; Hartley and Zisserman 2004).

An alternative formulation is to minimize the error in 3D projected ray directions (Shum and Szeliski 2000), i.e.,

$$E_{\text{BA-3D}} = \sum_i \sum_j c_{ij}\|\tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \boldsymbol{x}_i\|^2, \tag{9.30}$$

where $\tilde{\boldsymbol{x}}_i(\boldsymbol{x}_{ij}; \boldsymbol{R}_j, f_j)$ is given by the second half of (9.26). This has no particular advantage over (9.29). In fact, since errors are being minimized in 3D ray space, there is a bias towards estimating longer focal lengths, since the angles between rays become smaller as f increases.

However, if we eliminate the 3D rays $\boldsymbol{x}_i$, we can derive a pairwise energy formulated in 3D ray space (Shum and Szeliski 2000),

$$E_{\text{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ik}; \boldsymbol{R}_k, f_k)\|^2. \tag{9.31}$$

This results in the simplest set of update equations (Shum and Szeliski 2000), since the f_k can be folded into the creation of the homogeneous coordinate vector as in Equation (9.7). Thus, even though this formula over-weights features that occur more frequently, it is the method used by Shum and Szeliski (2000) and Brown, Szeliski, and Winder (2005). In order to reduce the bias towards longer focal lengths, we multiply each residual (3D error) by $\sqrt{f_j f_k}$, which is similar to projecting the 3D rays into a "virtual camera" of intermediate focal length.

Up vector selection. As mentioned above, there exists a global ambiguity in the pose of the 3D cameras computed by the above methods. While this may not appear to matter, people prefer that the final stitched image is "upright" rather than twisted or tilted. More concretely, people are used to seeing photographs displayed so that the vertical (gravity) axis points straight up in the image. Consider how you usually shoot photographs: while you may pan and tilt the camera any which way, you usually keep the horizontal edge of your camera (its x-axis) parallel to the ground plane (perpendicular to the world gravity direction).

Mathematically, this constraint on the rotation matrices can be expressed as follows. Recall from Equation (9.26) that the 3D to 2D projection is given by

$$\tilde{\boldsymbol{x}}_{ik} \sim \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{x}_i. \tag{9.32}$$

We wish to post-multiply each rotation matrix $\boldsymbol{R}_k$ by a global rotation $\boldsymbol{R}_{\text{g}}$ such that the projection of the global y-axis, $\hat{\boldsymbol{j}} = (0, 1, 0)$ is perpendicular to the image x-axis, $\hat{\boldsymbol{i}} = (1, 0, 0)$.[10]

This constraint can be written as

$$\hat{\boldsymbol{i}}^T \boldsymbol{R}_k \boldsymbol{R}_{\text{g}} \hat{\boldsymbol{j}} = 0 \tag{9.33}$$

(note that the scaling by the calibration matrix is irrelevant here). This is equivalent to requiring that the first row of $\boldsymbol{R}_k$, $\boldsymbol{r}_{k0} = \hat{\boldsymbol{i}}^T \boldsymbol{R}_k$ be perpendicular to the second column of $\boldsymbol{R}_{\text{g}}$, $\boldsymbol{r}_{\text{g1}} = \boldsymbol{R}_{\text{g}} \hat{\boldsymbol{j}}$. This set of constraints (one per input image) can be written as a least squares problem,

$$\boldsymbol{r}_{\text{g1}} = \arg\min_{\boldsymbol{r}} \sum_k (\boldsymbol{r}^T \boldsymbol{r}_{k0})^2 = \arg\min_{\boldsymbol{r}} \boldsymbol{r}^T \left[\sum_k \boldsymbol{r}_{k0} \boldsymbol{r}_{k0}^T \right] \boldsymbol{r}. \tag{9.34}$$

Thus, $\boldsymbol{r}_{\text{g1}}$ is the smallest eigenvector of the *scatter* or *moment* matrix spanned by the individual camera rotation x-vectors, which should generally be of the form $(c, 0, s)$ when the cameras are upright.

To fully specify the $\boldsymbol{R}_{\text{g}}$ global rotation, we need to specify one additional constraint. This is related to the *view selection* problem discussed in Section 9.3.1. One simple heuristic is to prefer the average z-axis of the individual rotation matrices, $\bar{\boldsymbol{k}} = \sum_k \hat{\boldsymbol{k}}^T \boldsymbol{R}_k$ to be close to the world z-axis, $\boldsymbol{r}_{\text{g2}} = \boldsymbol{R}_{\text{g}} \hat{\boldsymbol{k}}$. We can therefore compute the full rotation matrix $\boldsymbol{R}_{\text{g}}$ in three steps:

[10] Note that here we use the convention common in computer graphics that the vertical world axis corresponds to y. This is a natural choice if we wish the rotation matrix associated with a "regular" image taken horizontally to be the identity, rather than a 90° rotation around the x-axis.

1. $\boldsymbol{r}_{g1} = \min$ eigenvector $(\sum_k \boldsymbol{r}_{k0}\boldsymbol{r}_{k0}^T)$;
2. $\boldsymbol{r}_{g0} = \mathcal{N}((\sum_k \boldsymbol{r}_{k2}) \times \boldsymbol{r}_{g1})$;
3. $\boldsymbol{r}_{g2} = \boldsymbol{r}_{g0} \times \boldsymbol{r}_{g1}$,

where $\mathcal{N}(\boldsymbol{v}) = \boldsymbol{v}/\|\boldsymbol{v}\|$ normalizes a vector $\boldsymbol{v}$.

9.2.2 Parallax removal

Once we have optimized the global orientations and focal lengths of our cameras, we may find that the images are still not perfectly aligned, i.e., the resulting stitched image looks blurry or ghosted in some places. This can be caused by a variety of factors, including unmodeled radial distortion, 3D parallax (failure to rotate the camera around its optical center), small scene motions such as waving tree branches, and large-scale scene motions such as people moving in and out of pictures.

Each of these problems can be treated with a different approach. Radial distortion can be estimated (potentially ahead of time) using one of the techniques discussed in Section 2.1.6. For example, the *plumb-line method* (Brown 1971; Kang 2001; El-Melegy and Farag 2003) adjusts radial distortion parameters until slightly curved lines become straight, while mosaic-based approaches adjust them until mis-registration is reduced in image overlap areas (Stein 1997; Sawhney and Kumar 1999).

3D parallax can be handled by doing a full 3D bundle adjustment, i.e., by replacing the projection equation (9.26) used in Equation (9.29) with Equation (2.68), which models camera translations. The 3D positions of the matched feature points and cameras can then be simultaneously recovered, although this can be significantly more expensive than parallax-free image registration. Once the 3D structure has been recovered, the scene could (in theory) be projected to a single (central) viewpoint that contains no parallax. However, in order to do this, dense *stereo* correspondence needs to be performed (Section 11.3) (Li, Shum, Tang *et al.* 2004; Zheng, Kang, Cohen *et al.* 2007), which may not be possible if the images contain only partial overlap. In that case, it may be necessary to correct for parallax only in the overlap areas, which can be accomplished using a *multi-perspective plane sweep* (MPPS) algorithm (Kang, Szeliski, and Uyttendaele 2004; Uyttendaele, Criminisi, Kang *et al.* 2004).

When the motion in the scene is very large, i.e., when objects appear and disappear completely, a sensible solution is to simply *select* pixels from only one image at a time as the source for the final composite (Milgram 1977; Davis 1998; Agarwala, Dontcheva, Agrawala *et al.* 2004), as discussed in Section 9.3.2. However, when the motion is reasonably small (on the order of a few pixels), general 2D motion estimation (optical flow) can be used to perform an appropriate correction before blending using a process called *local alignment* (Shum and Szeliski 2000; Kang, Uyttendaele, Winder *et al.* 2003). This same process can also be used to compensate for radial distortion and 3D parallax, although it uses a weaker motion model than explicitly modeling the source of error and may, therefore, fail more often or introduce unwanted distortions.

The local alignment technique introduced by Shum and Szeliski (2000) starts with the global bundle adjustment (9.31) used to optimize the camera poses. Once these have been estimated, the *desired* location of a 3D point $\boldsymbol{x}_i$ can be estimated as the *average* of the back-

(a) (b) (c)

Figure 9.10 Deghosting a mosaic with motion parallax (Shum and Szeliski 2000) © 2000 IEEE: (a) composite with parallax; (b) after a single deghosting step (patch size 32); (c) after multiple steps (sizes 32, 16 and 8).

projected 3D locations,

$$\bar{\boldsymbol{x}}_i \sim \sum_j c_{ij} \bar{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) \Big/ \sum_j c_{ij} \,, \tag{9.35}$$

which can be projected into each image j to obtain a *target location* $\bar{\boldsymbol{x}}_{ij}$. The difference between the target locations $\bar{\boldsymbol{x}}_{ij}$ and the original features $\boldsymbol{x}_{ij}$ provide a set of local motion estimates

$$\boldsymbol{u}_{ij} = \bar{\boldsymbol{x}}_{ij} - \boldsymbol{x}_{ij}, \tag{9.36}$$

which can be interpolated to form a dense correction field $\boldsymbol{u}_j(\boldsymbol{x}_j)$. In their system, Shum and Szeliski (2000) use an *inverse warping* algorithm where the sparse $-\boldsymbol{u}_{ij}$ values are placed at the new target locations $\bar{\boldsymbol{x}}_{ij}$, interpolated using bilinear kernel functions (Nielson 1993) and then added to the original pixel coordinates when computing the warped (corrected) image. In order to get a reasonably dense set of features to interpolate, Shum and Szeliski (2000) place a feature point at the center of each patch (the patch size controls the smoothness in the local alignment stage), rather than relying of features extracted using an interest operator (Figure 9.10).

An alternative approach to motion-based de-ghosting was proposed by Kang, Uyttendaele, Winder *et al.* (2003), who estimate dense optical flow between each input image and a central *reference* image. The accuracy of the flow vector is checked using a photo-consistency measure before a given warped pixel is considered valid and is used to compute a high dynamic range radiance estimate, which is the goal of their overall algorithm. The requirement for a reference image makes their approach less applicable to general image mosaicing, although an extension to this case could certainly be envisaged.

9.2.3 Recognizing panoramas

The final piece needed to perform fully automated image stitching is a technique to recognize which images actually go together, which Brown and Lowe (2007) call *recognizing panoramas*. If the user takes images in sequence so that each image overlaps its predecessor and also specifies the first and last images to be stitched, bundle adjustment combined with the process of *topology inference* can be used to automatically assemble a panorama (Sawhney and Kumar 1999). However, users often jump around when taking panoramas, e.g., they may start a new row on top of a previous one, jump back to take a repeat shot, or create

360° panoramas where end-to-end overlaps need to be discovered. Furthermore, the ability to discover multiple panoramas taken by a user over an extended period of time can be a big convenience.

To recognize panoramas, Brown and Lowe (2007) first find all pairwise image overlaps using a feature-based method and then find connected components in the overlap graph to "recognize" individual panoramas (Figure 9.11). The feature-based matching stage first extracts scale invariant feature transform (SIFT) feature locations and feature descriptors (Lowe 2004) from all the input images and places them in an indexing structure, as described in Section 4.1.3. For each image pair under consideration, the nearest matching neighbor is found for each feature in the first image, using the indexing structure to rapidly find candidates and then comparing feature descriptors to find the best match. RANSAC is used to find a set of *inlier* matches; pairs of matches are used to hypothesize similarity motion models that are then used to count the number of inliers. (A more recent RANSAC algorithm tailored specifically for rotational panoramas is described by Brown, Hartley, and Nistér (2007).)

In practice, the most difficult part of getting a fully automated stitching algorithm to work is deciding which pairs of images actually correspond to the same parts of the scene. Repeated structures such as windows (Figure 9.12) can lead to false matches when using a feature-based approach. One way to mitigate this problem is to perform a direct pixel-based comparison between the registered images to determine if they actually are different views of the same scene. Unfortunately, this heuristic may fail if there are moving objects in the scene (Figure 9.13). While there is no magic bullet for this problem, short of full scene understanding, further improvements can likely be made by applying domain-specific heuristics, such as priors on typical camera motions as well as machine learning techniques applied to the problem of match validation.

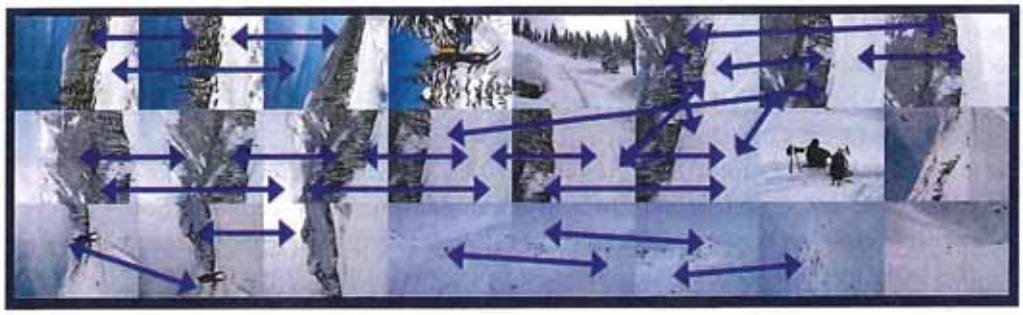
9.2.4 Direct vs. feature-based alignment

Given that there exist these two approaches to aligning images, which is preferable?

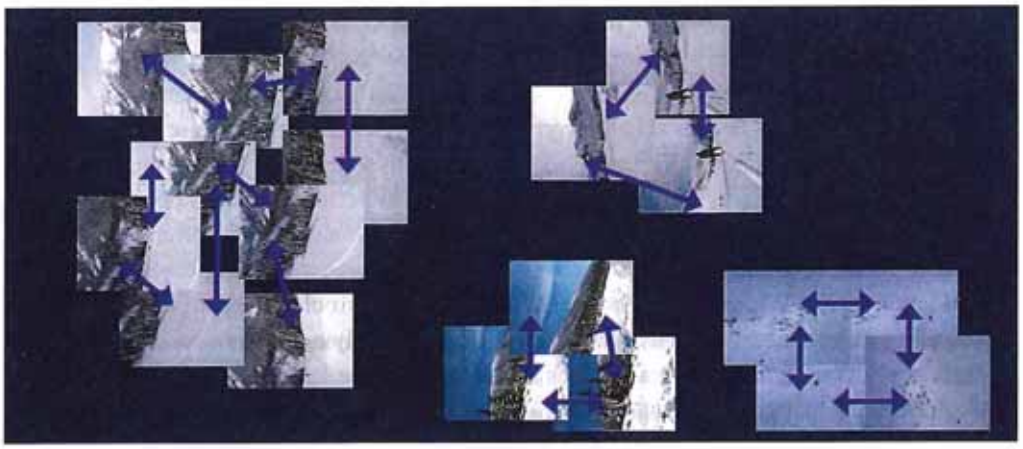
Early feature-based methods would get confused in regions that were either too textured or not textured enough. The features would often be distributed unevenly over the images, thereby failing to match image pairs that should have been aligned. Furthermore, establishing correspondences relied on simple cross-correlation between patches surrounding the feature points, which did not work well when the images were rotated or had foreshortening due to homographies.

Today, feature detection and matching schemes are remarkably robust and can even be used for known object recognition from widely separated views (Lowe 2004). Features not only respond to regions of high "cornerness" (Förstner 1986; Harris and Stephens 1988) but also to "blob-like" regions (Lowe 2004), and uniform areas (Matas, Chum, Urban *et al.* 2004; Tuytelaars and Van Gool 2004). Furthermore, because they operate in scale-space and use a dominant orientation (or orientation invariant descriptors), they can match images that differ in scale, orientation, and even foreshortening. Our own experience in working with feature-based approaches is that if the features are well distributed over the image and the descriptors reasonably designed for repeatability, enough correspondences to permit image stitching can usually be found (Brown, Szeliski, and Winder 2005).

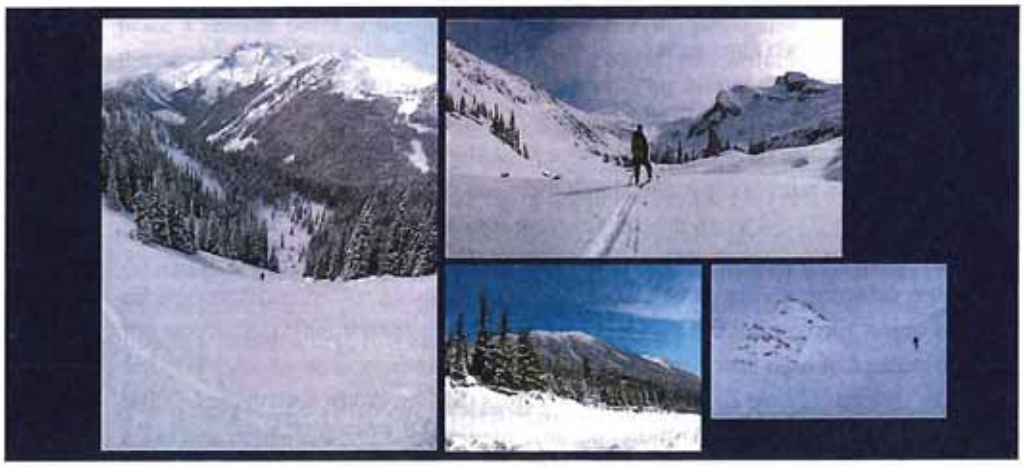
The biggest disadvantage of direct pixel-based alignment techniques is that they have a limited range of convergence. Even though they can be used in a hierarchical (coarse-to-



(a)



(b)



(c)

Figure 9.11 Recognizing panoramas (Brown, Szeliski, and Winder 2005), figures courtesy of Matthew Brown: (a) input images with pairwise matches; (b) images grouped into connected components (panoramas); (c) individual panoramas registered and blended into stitched composites.

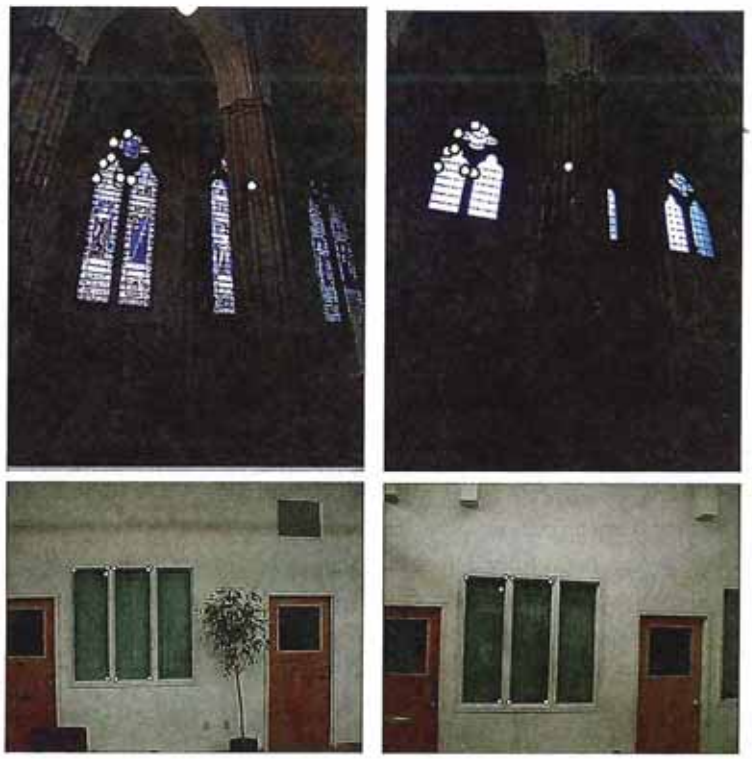


Figure 9.12 Matching errors (Brown, Szeliski, and Winder 2004): accidental matching of several features can lead to matches between pairs of images that do not actually overlap.

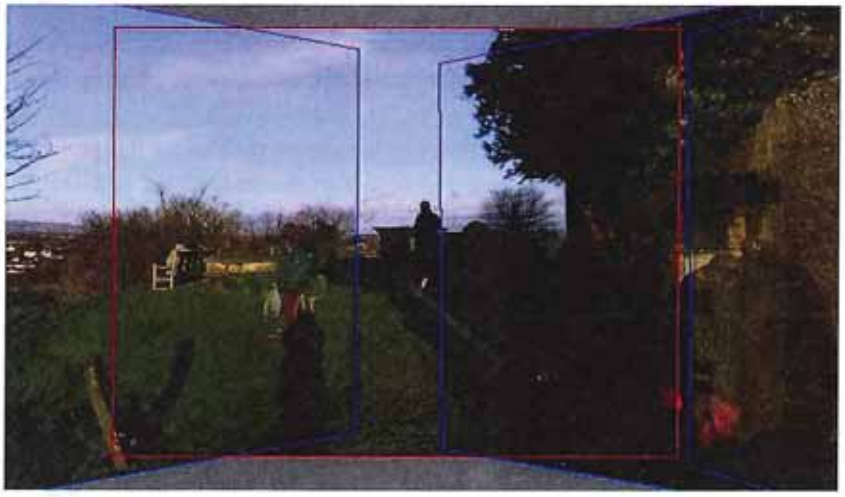


Figure 9.13 Validation of image matches by direct pixel error comparison can fail when the scene contains moving objects (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE.

fine) estimation framework, in practice it is hard to use more than two or three levels of a pyramid before important details start to be blurred away.[11] For matching sequential frames in a video, direct approaches can usually be made to work. However, for matching partially overlapping images in photo-based panoramas or for image collections where the contrast or content varies too much, they fail too often to be useful and feature-based approaches are therefore preferred.

9.3 Compositing

Once we have registered all of the input images with respect to each other, we need to decide how to produce the final stitched mosaic image. This involves selecting a final compositing surface (flat, cylindrical, spherical, etc.) and view (reference image). It also involves selecting which pixels contribute to the final composite and how to optimally blend these pixels to minimize visible seams, blur, and ghosting.

In this section, we review techniques that address these problems, namely compositing surface parameterization, pixel and seam selection, blending, and exposure compensation. My emphasis is on fully automated approaches to the problem. Since the creation of high-quality panoramas and composites is as much an artistic endeavor as a computational one, various interactive tools have been developed to assist this process (Agarwala, Dontcheva, Agrawala *et al.* 2004; Li, Sun, Tang *et al.* 2004; Rother, Kolmogorov, and Blake 2004). Some of these are covered in more detail in Section 10.4.

9.3.1 Choosing a compositing surface

The first choice to be made is how to represent the final image. If only a few images are stitched together, a natural approach is to select one of the images as the *reference* and to then warp all of the other images into its reference coordinate system. The resulting composite is sometimes called a *flat* panorama, since the projection onto the final surface is still a perspective projection, and hence straight lines remain straight (which is often a desirable attribute).[12]

For larger fields of view, however, we cannot maintain a flat representation without excessively stretching pixels near the border of the image. (In practice, flat panoramas start to look severely distorted once the field of view exceeds 90° or so.) The usual choice for compositing larger panoramas is to use a cylindrical (Chen 1995; Szeliski 1996) or spherical (Szeliski and Shum 1997) projection, as described in Section 9.1.6. In fact, any surface used for *environment mapping* in computer graphics can be used, including a *cube map*, which represents the full viewing sphere with the six square faces of a cube (Greene 1986; Szeliski and Shum 1997). Cartographers have also developed a number of alternative methods for representing the globe (Bugayevskiy and Snyder 1995).

The choice of parameterization is somewhat application dependent and involves a trade-off between keeping the local appearance undistorted (e.g., keeping straight lines straight)

[11] Fourier-based correlation (Szeliski 1996; Szeliski and Shum 1997) can extend this range but requires cylindrical images or motion prediction to be useful.

[12] Recently, some techniques have been developed to straighten curved lines in cylindrical and spherical panoramas (Carroll, Agrawala, and Agarwala 2009; Kopf, Lischinski, Deussen *et al.* 2009).

and providing a reasonably uniform sampling of the environment. Automatically making this selection and smoothly transitioning between representations based on the extent of the panorama is an active area of current research (Kopf, Uyttendaele, Deussen *et al.* 2007).

An interesting recent development in panoramic photography has been the use of stereographic projections looking down at the ground (in an outdoor scene) to create "little planet" renderings.[13]

View selection. Once we have chosen the output parameterization, we still need to determine which part of the scene will be *centered* in the final view. As mentioned above, for a flat composite, we can choose one of the images as a reference. Often, a reasonable choice is the one that is geometrically most central. For example, for rotational panoramas represented as a collection of 3D rotation matrices, we can choose the image whose z-axis is closest to the average z-axis (assuming a reasonable field of view). Alternatively, we can use the average z-axis (or quaternion, but this is trickier) to define the reference rotation matrix.

For larger, e.g., cylindrical or spherical, panoramas, we can use the same heuristic if a subset of the viewing sphere has been imaged. In the case of full 360° panoramas, a better choice might be to choose the middle image from the sequence of inputs, or sometimes the first image, assuming this contains the object of greatest interest. In all of these cases, having the user control the final view is often highly desirable. If the "up vector" computation described in Section 9.2.1 is working correctly, this can be as simple as panning over the image or setting a vertical "center line" for the final panorama.

Coordinate transformations. After selecting the parameterization and reference view, we still need to compute the mappings between the input and output pixels coordinates.

If the final compositing surface is flat (e.g., a single plane or the face of a cube map) and the input images have no radial distortion, the coordinate transformation is the simple homography described by (9.5). This kind of warping can be performed in graphics hardware by appropriately setting texture mapping coordinates and rendering a single quadrilateral.

If the final composite surface has some other analytic form (e.g., cylindrical or spherical), we need to convert every pixel in the final panorama into a viewing ray (3D point) and then map it back into each image according to the projection (and optionally radial distortion) equations. This process can be made more efficient by precomputing some lookup tables, e.g., the partial trigonometric functions needed to map cylindrical or spherical coordinates to 3D coordinates or the radial distortion field at each pixel. It is also possible to accelerate this process by computing exact pixel mappings on a coarser grid and then interpolating these values.

When the final compositing surface is a texture-mapped polyhedron, a slightly more sophisticated algorithm must be used. Not only do the 3D and texture map coordinates have to be properly handled, but a small amount of *overdraw* outside the triangle footprints in the texture map is necessary, to ensure that the texture pixels being interpolated during 3D rendering have valid values (Szeliski and Shum 1997).

[13] These are inspired by *The Little Prince* by Antoine De Saint-Exupery. Go to http://www.flickr.com and search for "little planet projection".

Sampling issues. While the above computations can yield the correct (fractional) pixel addresses in each input image, we still need to pay attention to sampling issues. For example, if the final panorama has a lower resolution than the input images, pre-filtering the input images is necessary to avoid aliasing. These issues have been extensively studied in both the image processing and computer graphics communities. The basic problem is to compute the appropriate pre-filter, which depends on the distance (and arrangement) between neighboring samples in a source image. As discussed in Sections 3.5.2 and 3.6.1, various approximate solutions, such as MIP mapping (Williams 1983) or elliptically weighted Gaussian averaging (Greene and Heckbert 1986) have been developed in the graphics community. For highest visual quality, a higher order (e.g., cubic) interpolator combined with a spatially adaptive pre-filter may be necessary (Wang, Kang, Szeliski *et al.* 2001). Under certain conditions, it may also be possible to produce images with a higher resolution than the input images using the process of *super-resolution* (Section 10.3).

9.3.2 Pixel selection and weighting (de-ghosting)

Once the source pixels have been mapped onto the final composite surface, we must still decide how to blend them in order to create an attractive-looking panorama. If all of the images are in perfect registration and identically exposed, this is an easy problem, i.e., any pixel or combination will do. However, for real images, visible seams (due to exposure differences), blurring (due to mis-registration), or ghosting (due to moving objects) can occur.

Creating clean, pleasing-looking panoramas involves both deciding which pixels to use and how to weight or blend them. The distinction between these two stages is a little fluid, since per-pixel weighting can be thought of as a combination of selection and blending. In this section, we discuss spatially varying weighting, pixel selection (seam placement), and then more sophisticated blending.

Feathering and center-weighting. The simplest way to create a final composite is to simply take an *average* value at each pixel,

$$C(\boldsymbol{x}) = \sum_k w_k(\boldsymbol{x}) \tilde{I}_k(\boldsymbol{x}) \Big/ \sum_k w_k(\boldsymbol{x}) \,, \tag{9.37}$$

where $\tilde{I}_k(\boldsymbol{x})$ are the *warped* (re-sampled) images and $w_k(\boldsymbol{x})$ is 1 at valid pixels and 0 elsewhere. On computer graphics hardware, this kind of summation can be performed in an *accumulation buffer* (using the A channel as the weight).

Simple averaging usually does not work very well, since exposure differences, mis-registrations, and scene movement are all very visible (Figure 9.14a). If rapidly moving objects are the only problem, taking a *median* filter (which is a kind of pixel selection operator) can often be used to remove them (Figure 9.14b) (Irani and Anandan 1998). Conversely, center-weighting (discussed below) and *minimum likelihood* selection (Agarwala, Dontcheva, Agrawala *et al.* 2004) can sometimes be used to retain multiple copies of a moving object (Figure 9.17).

A better approach to averaging is to weight pixels near the center of the image more heavily and to down-weight pixels near the edges. When an image has some cutout regions,

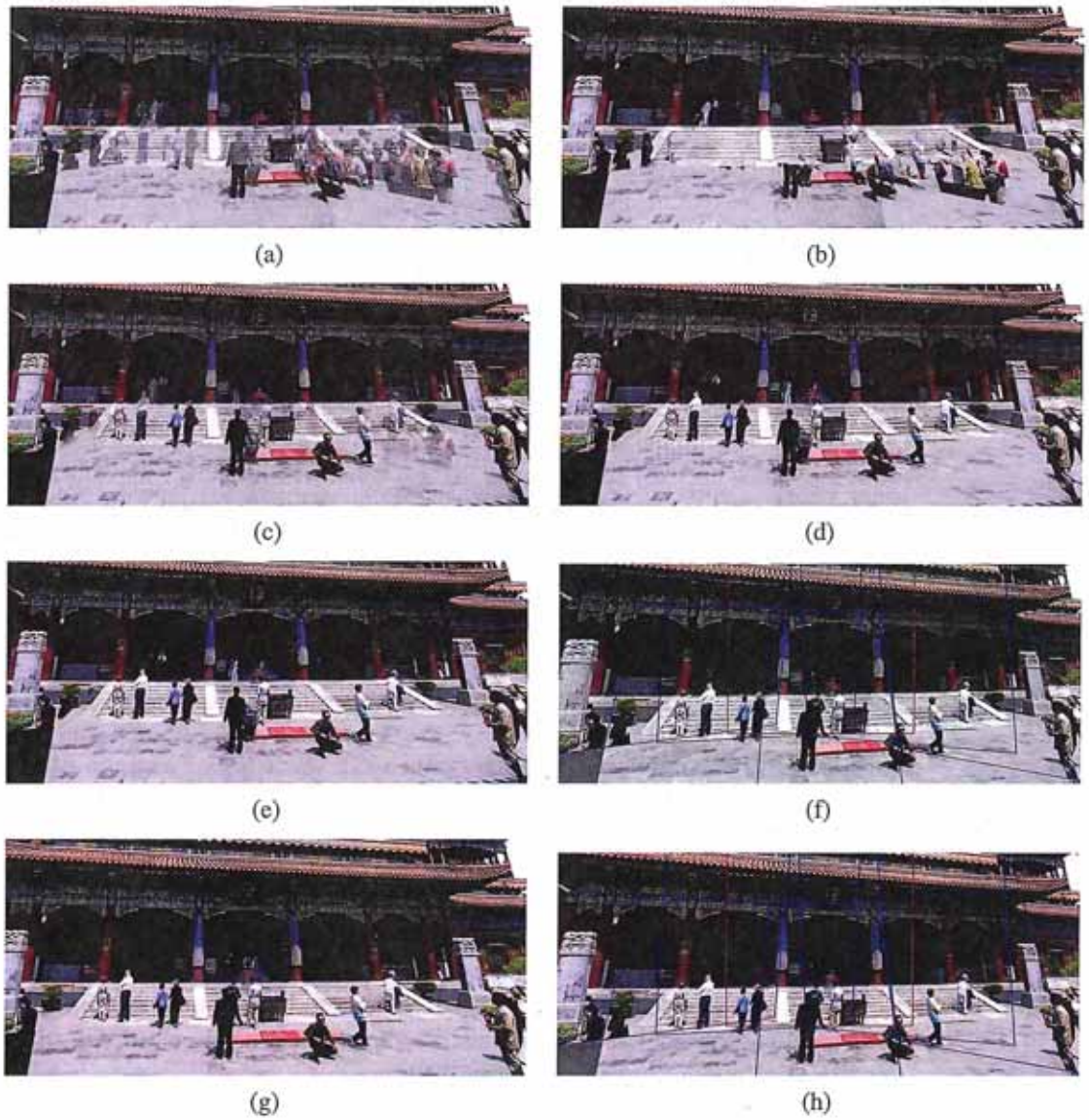


Figure 9.14 Final composites computed by a variety of algorithms (Szeliski 2006a): (a) average, (b) median, (c) feathered average, (d) *p-norm* $p = 10$, (e) Voronoi, (f) weighted ROD vertex cover with feathering, (g) graph cut seams with Poisson blending and (h) with pyramid blending.

down-weighting pixels near the edges of both cutouts and the image is preferable. This can be done by computing a *distance map* or *grassfire transform*,

$$w_k(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} \{\|\boldsymbol{y}\| \mid \tilde{I}_k(\boldsymbol{x}+\boldsymbol{y}) \text{ is invalid } \}, \tag{9.38}$$

where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel (Section 3.3.3). The Euclidean distance map can be efficiently computed using a two-pass raster algorithm (Danielsson 1980; Borgefors 1986).

Weighted averaging with a distance map is often called *feathering* (Szeliski and Shum 1997; Chen and Klette 1999; Uyttendaele, Eden, and Szeliski 2001) and does a reasonable job of blending over exposure differences. However, blurring and ghosting can still be problems (Figure 9.14c). Note that weighted averaging is *not* the same as compositing the individual images with the classic *over* operation (Porter and Duff 1984; Blinn 1994a), even when using the weight values (normalized to sum up to one) as *alpha* (translucency) channels. This is because the over operation attenuates the values from more distant surfaces and, hence, is not equivalent to a direct sum.

One way to improve feathering is to raise the distance map values to some large power, i.e., to use $w_k^p(\boldsymbol{x})$ in Equation (9.37). The weighted averages then become dominated by the larger values, i.e., they act somewhat like a *p-norm*. The resulting composite can often provide a reasonable tradeoff between visible exposure differences and blur (Figure 9.14d).

In the limit as $p \to \infty$, only the pixel with the maximum weight is selected,

$$C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}), \tag{9.39}$$

where

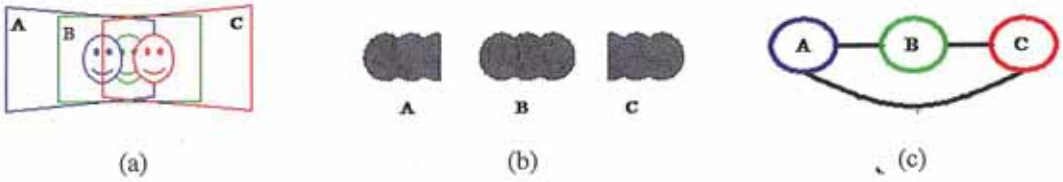
$$l = \arg\max_k w_k(\boldsymbol{x}) \tag{9.40}$$

is the *label assignment* or *pixel selection* function that selects which image to use at each pixel. This hard pixel selection process produces a visibility mask-sensitive variant of the familiar *Voronoi diagram*, which assigns each pixel to the nearest image center in the set (Wood, Finkelstein, Hughes *et al.* 1997; Peleg, Rousso, Rav-Acha *et al.* 2000). The resulting composite, while useful for artistic guidance and in high-overlap panoramas (*manifold mosaics*) tends to have very hard edges with noticeable seams when the exposures vary (Figure 9.14e).

Xiong and Turkowski (1998) use this Voronoi idea (local maximum of the grassfire transform) to select seams for Laplacian pyramid blending (which is discussed below). However, since the seam selection is performed sequentially as new images are added in, some artifacts can occur.

Optimal seam selection. Computing the Voronoi diagram is one way to select the *seams* between regions where different images contribute to the final composite. However, Voronoi images totally ignore the local image structure underlying the seam.

A better approach is to place the seams in regions where the images agree, so that transitions from one source to another are not visible. In this way, the algorithm avoids "cutting through" moving objects where a seam would look unnatural (Davis 1998). For a pair of images, this process can be formulated as a simple dynamic program starting from one edge



(a) (b) (c)

Figure 9.15 Computation of regions of difference (RODs) (Uyttendaele, Eden, and Szeliski 2001) © 2001 IEEE: (a) three overlapping images with a moving face; (b) corresponding RODs; (c) graph of coincident RODs.

of the overlap region and ending at the other (Milgram 1975, 1977; Davis 1998; Efros and Freeman 2001).

When multiple images are being composited, the dynamic program idea does not readily generalize. (For square texture tiles being composited sequentially, Efros and Freeman (2001) run a dynamic program along each of the four tile sides.)

To overcome this problem, Uyttendaele, Eden, and Szeliski (2001) observed that, for well-registered images, moving objects produce the most visible artifacts, namely translucent looking *ghosts*. Their system therefore decides which objects to keep and which ones to erase. First, the algorithm compares all overlapping input image pairs to determine *regions of difference* (RODs) where the images disagree. Next, a graph is constructed with the RODs as vertices and edges representing ROD pairs that overlap in the final composite (Figure 9.15). Since the presence of an edge indicates an area of disagreement, vertices (regions) must be removed from the final composite until no edge spans a pair of remaining vertices. The smallest such set can be computed using a *vertex cover* algorithm. Since several such covers may exist, a *weighted vertex cover* is used instead, where the vertex weights are computed by summing the feather weights in the ROD (Uyttendaele, Eden, and Szeliski 2001). The algorithm therefore prefers removing regions that are near the edge of the image, which reduces the likelihood that partially visible objects will appear in the final composite. (It is also possible to infer which object in a region of difference is the foreground object by the "edginess" (pixel differences) across the ROD boundary, which should be higher when an object is present (Herley 2005).) Once the desired excess regions of difference have been removed, the final composite can be created by feathering (Figure 9.14f).

A different approach to pixel selection and seam placement is described by Agarwala, Dontcheva, Agrawala *et al.* (2004). Their system computes the label assignment that optimizes the sum of two objective functions. The first is a per-pixel *image objective* that determines which pixels are likely to produce good composites,

$$C_D = \sum_{\boldsymbol{x}} D(\boldsymbol{x}, l(\boldsymbol{x})), \tag{9.41}$$

where $D(\boldsymbol{x}, l)$ is the *data penalty* associated with choosing image l at pixel $\boldsymbol{x}$. In their system, users can select which pixels to use by "painting" over an image with the desired object or appearance, which sets $D(\boldsymbol{x}, l)$ to a large value for all labels l other than the one selected by the user (Figure 9.16). Alternatively, automated selection criteria can be used, such as *maximum likelihood*, which prefers pixels that occur repeatedly in the background (for object removal), or *minimum likelihood* for objects that occur infrequently, i.e., for moving object retention. Using a more traditional center-weighted data term tends to favor objects that are

Figure 9.16 Photomontage (Agarwala, Dontcheva, Agrawala *et al.* 2004) © 2004 ACM. From a set of five source images (of which four are shown on the left), Photomontage quickly creates a composite family portrait in which everyone is smiling and looking at the camera (right). Users simply flip through the stack and coarsely draw strokes using the designated source image objective over the people they wish to add to the composite. The user-applied strokes and computed regions (middle) are color-coded by the borders of the source images on the left.

centered in the input images (Figure 9.17).

The second term is a *seam objective* that penalizes differences in labelings between adjacent images,

$$C_S = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{N}} S(\boldsymbol{x},\boldsymbol{y},l(\boldsymbol{x}),l(\boldsymbol{y})), \tag{9.42}$$

where $S(\boldsymbol{x},\boldsymbol{y},l_x,l_y)$ is the image-dependent *interaction penalty* or *seam cost* of placing a seam between pixels $\boldsymbol{x}$ and $\boldsymbol{y}$, and $\mathcal{N}$ is the set of $\mathcal{N}_4$ neighboring pixels. For example, the simple color-based seam penalty used in (Kwatra, Schödl, Essa *et al.* 2003; Agarwala, Dontcheva, Agrawala *et al.* 2004) can be written as

$$S(\boldsymbol{x},\boldsymbol{y},l_x,l_y) = \|\bar{I}_{l_x}(\boldsymbol{x}) - \bar{I}_{l_y}(\boldsymbol{x})\| + \|\bar{I}_{l_x}(\boldsymbol{y}) - \bar{I}_{l_y}(\boldsymbol{y})\|. \tag{9.43}$$

More sophisticated seam penalties can also look at image gradients or the presence of image edges (Agarwala, Dontcheva, Agrawala *et al.* 2004). Seam penalties are widely used in other computer vision applications such as stereo matching (Boykov, Veksler, and Zabih 2001) to give the labeling function its *coherence* or *smoothness*. An alternative approach, which places seams along strong consistent edges in overlapping images using a watershed computation is described by Soille (2006).

The sum of these two objective functions gives rise to a *Markov random field* (MRF), for which good optimization algorithms are described in Sections 3.7.2 and 5.5 and Appendix B.5. For label computations of this kind, the *α-expansion* algorithm developed by Boykov, Veksler, and Zabih (2001) works particularly well (Szeliski, Zabih, Scharstein *et al.* 2008).

For the result shown in Figure 9.14g, Agarwala, Dontcheva, Agrawala *et al.* (2004) use a large data penalty for invalid pixels and 0 for valid pixels. Notice how the seam placement algorithm avoids regions of difference, including those that border the image and that might result in objects being cut off. Graph cuts (Agarwala, Dontcheva, Agrawala *et al.* 2004) and vertex cover (Uyttendaele, Eden, and Szeliski 2001) often produce similar looking results, although the former is significantly slower since it optimizes over all pixels, while the latter is more sensitive to the thresholds used to determine regions of difference.

Figure 9.17 Set of five photos tracking a snowboarder's jump stitched together into a seamless composite. Because the algorithm prefers pixels near the center of the image, multiple copies of the boarder are retained.

9.3.3 *Application*: Photomontage

While image stitching is normally used to composite partially overlapping photographs, it can also be used to composite repeated shots of a scene taken with the aim of obtaining the best possible composition and appearance of each element.

Figure 9.16 shows the *Photomontage* system developed by Agarwala, Dontcheva, Agrawala *et al.* (2004), where users draw strokes over a set of pre-aligned images to indicate which regions they wish to keep from each image. Once the system solves the resulting multi-label graph cut (9.41–9.42), the various pieces taken from each source photo are blended together using a variant of Poisson image blending (9.44–9.46). Their system can also be used to automatically composite an all-focus image from a series of bracketed focus images (Hasinoff, Kutulakos, Durand *et al.* 2009) or to remove wires and other unwanted elements from sets of photographs. Exercise 9.10 has you implement this system and try out some of its variants.

9.3.4 Blending

Once the seams between images have been determined and unwanted objects removed, we still need to blend the images to compensate for exposure differences and other mis-alignments. The spatially varying weighting (feathering) previously discussed can often be used to accomplish this. However, it is difficult in practice to achieve a pleasing balance between smoothing out low-frequency exposure variations and retaining sharp enough transitions to prevent blurring (although using a high exponent in feathering can help).

Laplacian pyramid blending. An attractive solution to this problem is the Laplacian pyramid blending technique developed by Burt and Adelson (1983b), which we discussed in Section 3.5.5. Instead of using a single transition width, a frequency-adaptive width is used by creating a band-pass (Laplacian) pyramid and making the transition widths within each level

(a) (b) (c)

Figure 9.18 Poisson image editing (Pérez, Gangnet, and Blake 2003) © 2003 ACM: (a) The dog and the two children are chosen as source images to be pasted into the destination swimming pool. (b) Simple pasting fails to match the colors at the boundaries, whereas (c) Poisson image blending masks these differences.

a function of the level, i.e., the same width in pixels. In practice, a small number of levels, i.e., as few as two (Brown and Lowe 2007), may be adequate to compensate for differences in exposure. The result of applying this pyramid blending is shown in Figure 9.14h.

Gradient domain blending. An alternative approach to multi-band image blending is to perform the operations in the *gradient domain*. Reconstructing images from their gradient fields has a long history in computer vision (Horn 1986), starting originally with work in brightness constancy (Horn 1974), shape from shading (Horn and Brooks 1989), and photometric stereo (Woodham 1981). More recently, related ideas have been used for reconstructing images from their edges (Elder and Goldberg 2001), removing shadows from images (Weiss 2001), separating reflections from a single image (Levin, Zomet, and Weiss 2004; Levin and Weiss 2007), and *tone mapping* high dynamic range images by reducing the magnitude of image edges (gradients) (Fattal, Lischinski, and Werman 2002).

Pérez, Gangnet, and Blake (2003) show how gradient domain reconstruction can be used to do seamless object insertion in image editing applications (Figure 9.18). Rather than copying pixels, the *gradients* of the new image fragment are copied instead. The actual pixel values for the copied area are then computed by solving a *Poisson equation* that locally matches the gradients while obeying the fixed *Dirichlet* (exact matching) conditions at the seam boundary. Pérez, Gangnet, and Blake (2003) show that this is equivalent to computing an additive *membrane* interpolant of the mismatch between the source and destination images along the boundary.[14] In earlier work, Peleg (1981) also proposed adding a smooth function to enforce consistency along the seam curve.

Agarwala, Dontcheva, Agrawala *et al.* (2004) extended this idea to a multi-source formulation, where it no longer makes sense to talk of a destination image whose exact pixel values must be matched at the seam. Instead, *each* source image contributes its own gradient field and the Poisson equation is solved using *Neumann* boundary conditions, i.e., dropping any

[14] The membrane interpolant is known to have nicer interpolation properties for arbitrary-shaped constraints than frequency-domain interpolants (Nielson 1993).

equations that involve pixels outside the boundary of the image.

Rather than solving the Poisson partial differential equations, Agarwala, Dontcheva, Agrawala *et al.* (2004) directly minimize a *variational problem*,

$$\min_{C(\boldsymbol{x})} \|\nabla C(\boldsymbol{x}) - \nabla \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x})\|^2. \tag{9.44}$$

The discretized form of this equation is a set of gradient constraint equations

$$C(\boldsymbol{x}+\hat{\boldsymbol{\imath}}) - C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}+\hat{\boldsymbol{\imath}}) - \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}) \text{ and} \tag{9.45}$$

$$C(\boldsymbol{x}+\hat{\boldsymbol{\jmath}}) - C(\boldsymbol{x}) = \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}+\hat{\boldsymbol{\jmath}}) - \tilde{I}_{l(\boldsymbol{x})}(\boldsymbol{x}), \tag{9.46}$$

where $\hat{\boldsymbol{\imath}} = (1, 0)$ and $\hat{\boldsymbol{\jmath}} = (0, 1)$ are unit vectors in the x and y directions.[15] They then solve the associated sparse least squares problem. Since this system of equations is only defined up to an additive constraint, Agarwala, Dontcheva, Agrawala *et al.* (2004) ask the user to select the value of one pixel. In practice, a better choice might be to weakly bias the solution towards reproducing the original color values.

In order to accelerate the solution of this sparse linear system, Fattal, Lischinski, and Werman (2002) use multigrid, whereas Agarwala, Dontcheva, Agrawala *et al.* (2004) use hierarchical basis preconditioned conjugate gradient descent (Szeliski 1990b, 2006b) (Appendix A.5). In subsequent work, Agarwala (2007) shows how using a quadtree representation for the solution can further accelerate the computation with minimal loss in accuracy, while Szeliski, Uyttendaele, and Steedly (2008) show how representing the per-image offset fields using even coarser splines is even faster. This latter work also argues that blending in the log domain, i.e., using multiplicative rather than additive offsets, is preferable, as it more closely matches texture contrasts across seam boundaries. The resulting seam blending works very well in practice (Figure 9.14h), although care must be taken when copying large gradient values near seams so that a "double edge" is not introduced.

Copying gradients directly from the source images after seam placement is just one approach to gradient domain blending. The paper by Levin, Zomet, Peleg *et al.* (2004) examines several different variants of this approach, which they call *Gradient-domain Image STitching* (GIST). The techniques they examine include feathering (blending) the gradients from the source images, as well as using an L1 norm in performing the reconstruction of the image from the gradient field, rather than using an L2 norm as in Equation (9.44). Their preferred technique is the L1 optimization of a feathered (blended) cost function on the original image gradients (which they call GIST1-l_1). Since L1 optimization using linear programming can be slow, they develop a faster iterative median-based algorithm in a multigrid framework. Visual comparisons between their preferred approach and what they call *optimal seam on the gradients* (which is equivalent to the approach of Agarwala, Dontcheva, Agrawala *et al.* (2004)) show similar results, while significantly improving on pyramid blending and feathering algorithms.

Exposure compensation. Pyramid and gradient domain blending can do a good job of compensating for moderate amounts of exposure differences between images. However, when the exposure differences become large, alternative approaches may be necessary.

[15] At seam locations, the right hand side is replaced by the average of the gradients in the two source images.

Uyttendaele, Eden, and Szeliski (2001) iteratively estimate a local correction between each source image and a blended composite. First, a block-based quadratic transfer function is fit between each source image and an initial feathered composite. Next, transfer functions are averaged with their neighbors to get a smoother mapping and per-pixel transfer functions are computed by *splining* (interpolating) between neighboring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed and the process is repeated (typically three times). The results shown by Uyttendaele, Eden, and Szeliski (2001) demonstrate that this does a better job of exposure compensation than simple feathering and can handle local variations in exposure due to effects such as lens vignetting.

Ultimately, however, the most principled way to deal with exposure differences is to stitch images in the radiance domain, i.e., to convert each image into a radiance image using its exposure value and then create a stitched, high dynamic range image, as discussed in Section 10.2 (Eden, Uyttendaele, and Szeliski 2006).

9.4 Additional reading

The literature on image stitching dates back to work in the photogrammetry community in the 1970s (Milgram 1975, 1977; Slama 1980). In computer vision, papers started appearing in the early 1980s (Peleg 1981), while the development of fully automated techniques came about a decade later (Mann and Picard 1994; Chen 1995; Szeliski 1996; Szeliski and Shum 1997; Sawhney and Kumar 1999; Shum and Szeliski 2000). Those techniques used direct pixel-based alignment but feature-based approaches are now the norm (Zoghlami, Faugeras, and Deriche 1997; Capel and Zisserman 1998; Cham and Cipolla 1998; Badra, Qumsieh, and Dudek 1998; McLauchlan and Jaenicke 2002; Brown and Lowe 2007). A collection of some of these papers can be found in the book by Benosman and Kang (2001). Szeliski (2006a) provides a comprehensive survey of image stitching, on which the material in this chapter is based.

High-quality techniques for optimal seam finding and blending are another important component of image stitching systems. Important developments in this field include work by Milgram (1977), Burt and Adelson (1983b), Davis (1998), Uyttendaele, Eden, and Szeliski (2001),Pérez, Gangnet, and Blake (2003), Levin, Zomet, Peleg *et al.* (2004), Agarwala, Dontcheva, Agrawala *et al.* (2004), Eden, Uyttendaele, and Szeliski (2006), and Kopf, Uyttendaele, Deussen *et al.* (2007).

In addition to the merging of multiple overlapping photographs taken for aerial or terrestrial panoramic image creation, stitching techniques can be used for automated whiteboard scanning (He and Zhang 2005; Zhang and He 2007), scanning with a mouse (Nakao, Kashitani, and Kaneyoshi 1998), and retinal image mosaics (Can, Stewart, Roysam *et al.* 2002). They can also be applied to video sequences (Teodosio and Bender 1993; Irani, Hsu, and Anandan 1995; Kumar, Anandan, Irani *et al.* 1995; Sawhney and Ayer 1996; Massey and Bender 1996; Irani and Anandan 1998; Sawhney, Arpa, Kumar *et al.* 2002; Agarwala, Zheng, Pal *et al.* 2005; Rav-Acha, Pritch, Lischinski *et al.* 2005; Steedly, Pal, and Szeliski 2005; Baudisch, Tan, Steedly *et al.* 2006) and can even be used for video compression (Lee, ge Chen, lung Bruce Lin *et al.* 1997).

9.5 Exercises

Ex 9.1: Direct pixel-based alignment Take a pair of images, compute a coarse-to-fine affine alignment (Exercise 8.2) and then blend them using either averaging (Exercise 6.2) or a Laplacian pyramid (Exercise 3.20). Extend your motion model from affine to perspective (homography) to better deal with rotational mosaics and planar surfaces seen under arbitrary motion.

Ex 9.2: Featured-based stitching Extend your feature-based alignment technique from Exercise 6.2 to use a full perspective model and then blend the resulting mosaic using either averaging or more sophisticated distance-based feathering (Exercise 9.9).

Ex 9.3: Cylindrical strip panoramas To generate cylindrical or spherical panoramas from a horizontally panning (rotating) camera, it is best to use a tripod. Set your camera up to take a series of 50% overlapped photos and then use the following steps to create your panorama:

1. Estimate the amount of radial distortion by taking some pictures with lots of long straight lines near the edges of the image and then using the plumb-line method from Exercise 6.10.
2. Compute the focal length either by using a ruler and paper, as in Figure 6.7 (Debevec, Wenger, Tchou *et al.* 2002) or by rotating your camera on the tripod, overlapping the images by exactly 0% and counting the number of images it takes to make a 360° panorama.
3. Convert each of your images to cylindrical coordinates using (9.12–9.16).
4. Line up the images with a translational motion model using either a direct pixel-based technique, such as coarse-to-fine incremental or an FFT, or a feature-based technique.
5. (Optional) If doing a complete 360° panorama, align the first and last images. Compute the amount of accumulated vertical mis-registration and re-distribute this among the images.
6. Blend the resulting images using feathering or some other technique.

Ex 9.4: Coarse alignment Use FFT or phase correlation (Section 8.1.2) to estimate the initial alignment between successive images. How well does this work? Over what range of overlaps? If it does not work, does aligning sub-sections (e.g., quarters) do better?

Ex 9.5: Automated mosaicing Use feature-based alignment with four-point RANSAC for homographies (Section 6.1.3, Equations (6.19–6.23)) or three-point RANSAC for rotational motions (Brown, Hartley, and Nistér 2007) to match up all pairs of overlapping images.

Merge these pairwise estimates together by finding a spanning tree of pairwise relations. Visualize the resulting global alignment, e.g., by displaying a blend of each image with all other images that overlap it.

For greater robustness, try multiple spanning trees (perhaps randomly sampled based on the confidence in pairwise alignments) to see if you can recover from bad pairwise matches (Zach, Klopschitz, and Pollefeys 2010). As a measure of fitness, count how many pairwise estimates are consistent with the global alignment.

Ex 9.6: Global optimization Use the initialization from the previous algorithm to perform a full bundle adjustment over all of the camera rotations and focal lengths, as described in Section 7.4 and by Shum and Szeliski (2000). Optionally, estimate radial distortion parameters as well or support fisheye lenses (Section 2.1.6).

As in the previous exercise, visualize the quality of your registration by creating composites of each input image with its neighbors, optionally blinking between the original image and the composite to better see mis-alignment artifacts.

Ex 9.7: De-ghosting Use the results of the previous bundle adjustment to predict the location of each feature in a consensus geometry. Use the difference between the predicted and actual feature locations to correct for small mis-registrations, as described in Section 9.2.2 (Shum and Szeliski 2000).

Ex 9.8: Compositing surface Choose a compositing surface (Section 9.3.1), e.g., a single reference image extended to a larger plane, a sphere represented using cylindrical or spherical coordinates, a stereographic "little planet" projection, or a cube map.

Project all of your images onto this surface and blend them with equal weighting, for now (just to see where the original image seams are).

Ex 9.9: Feathering and blending Compute a feather (distance) map for each warped source image and use these maps to blend the warped images.

Alternatively, use Laplacian pyramid blending (Exercise 3.20) or gradient domain blending.

Ex 9.10: Photomontage and object removal Implement a "PhotoMontage" system in which users can indicate desired or unwanted regions in pre-registered images using strokes or other primitives (such as bounding boxes).

(Optional) Devise an automatic moving objects remover (or "keeper") by analyzing which inconsistent regions are more or less typical given some consensus (e.g., median filtering) of the aligned images. Figure 9.17 shows an example where the moving object was kept. Try to make this work for sequences with large amounts of overlaps and consider averaging the images to make the moving object look more ghosted.

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

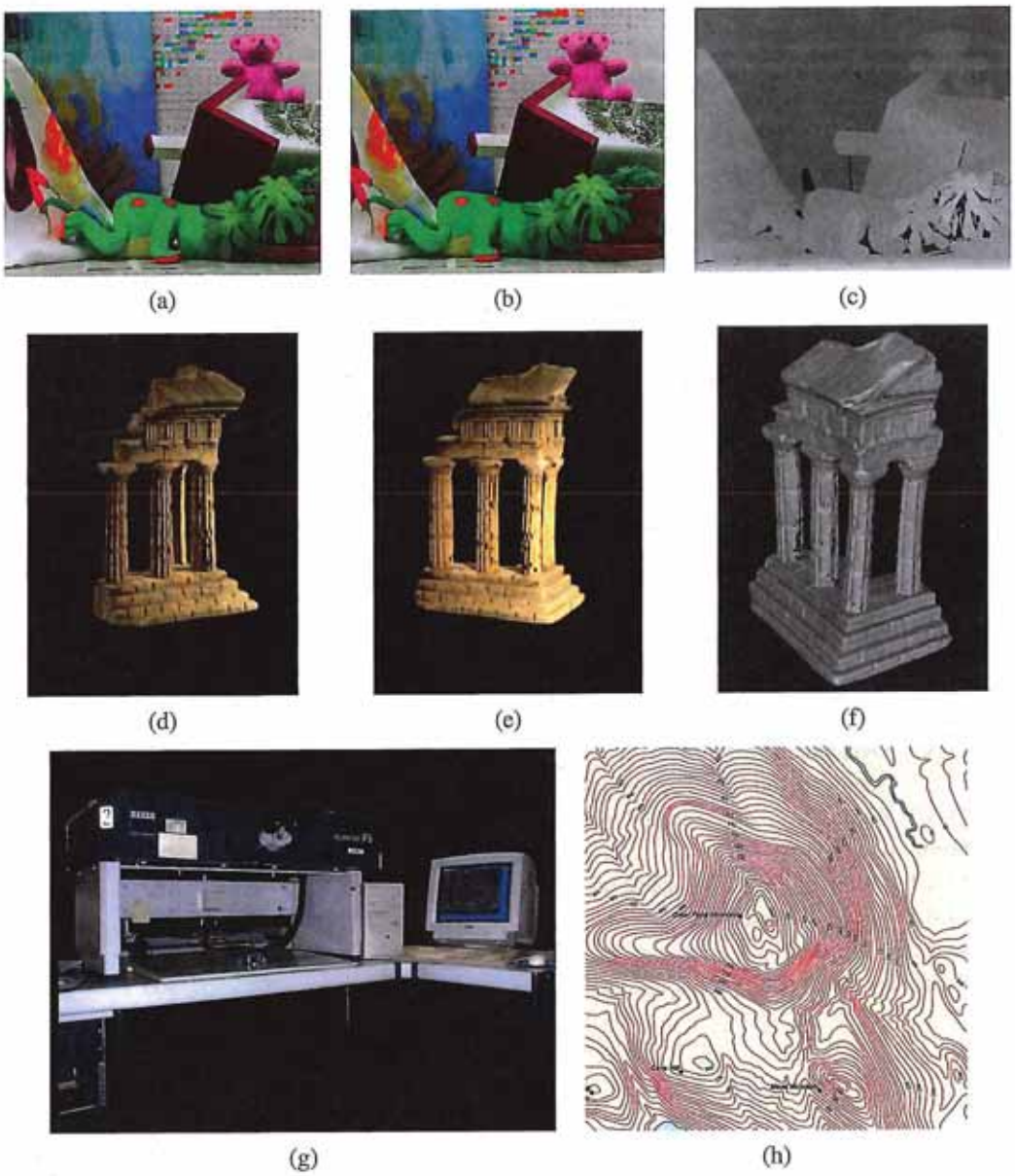


Figure 11.1 Stereo reconstruction techniques can convert (a–b) a pair of images into (c) a depth map (http://vision.middlebury.edu/stereo/data/scenes2003/) or (d–e) a sequence of images into (f) a 3D model (http://vision.middlebury.edu/mview/data/). (g) An analytical stereo plotter, courtesy of Kenney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

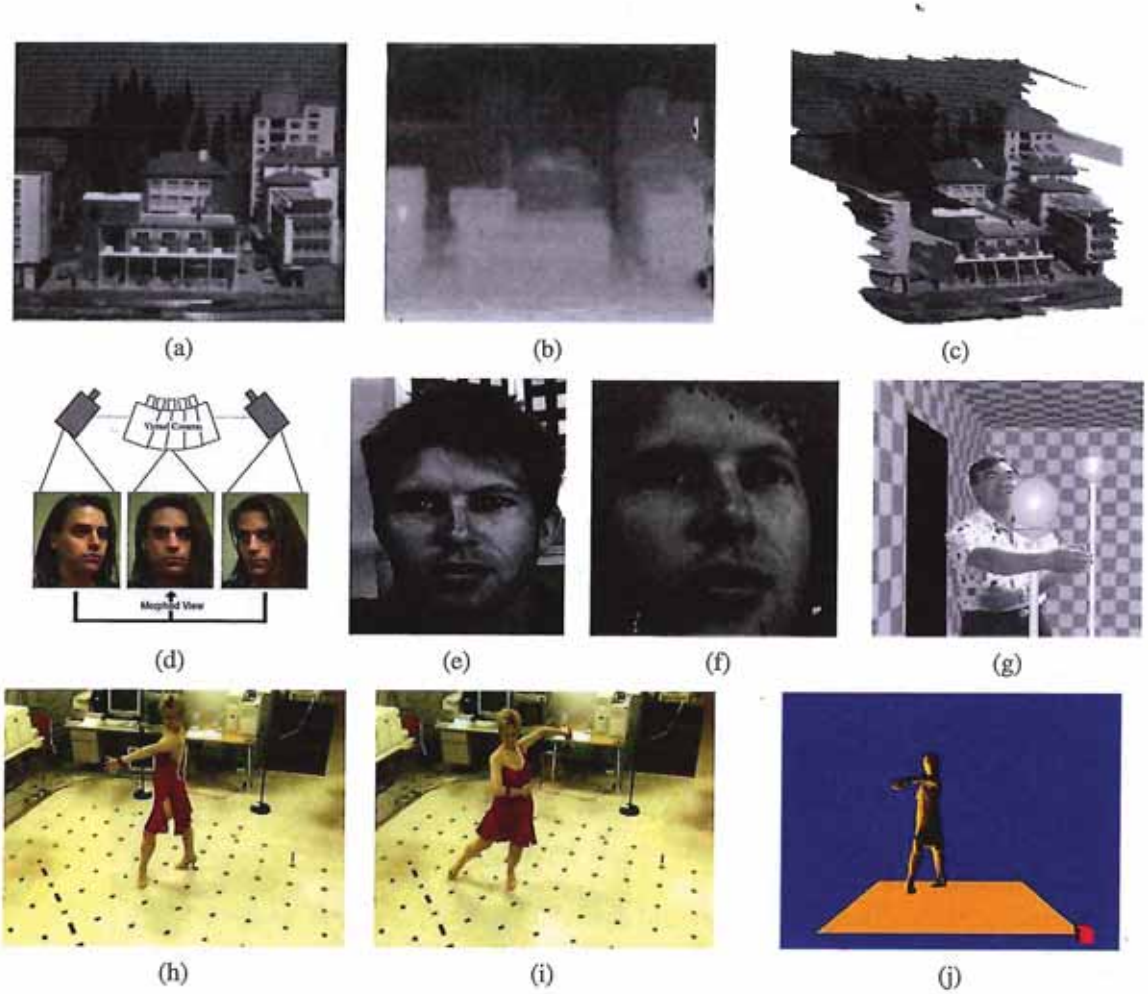


Figure 11.2 Applications of stereo vision: (a) input image, (b) computed depth map, and (c) new view generation from multi-view stereo (Matthies, Kanade, and Szeliski 1989) © 1989 Springer; (d) view morphing between two images (Seitz and Dyer 1996) © 1996 ACM; (e–f) 3D face modeling (images courtesy 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).

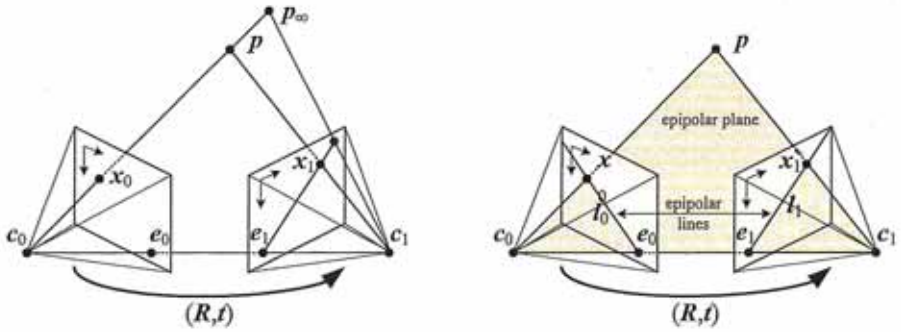


Figure 11.3 Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.

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).

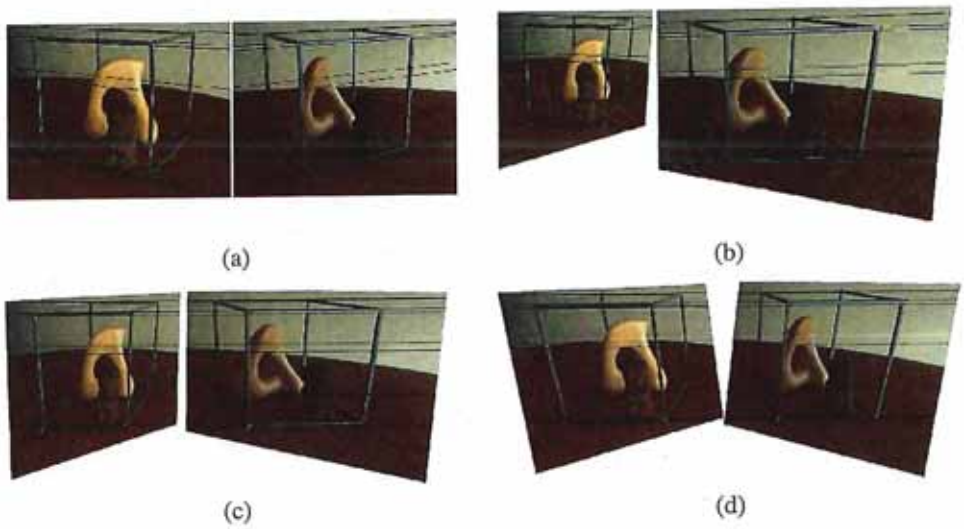


Figure 11.4 The multi-stage stereo rectification algorithm of Loop and Zhang (1999) © 1999 IEEE. (a) Original image pair overlaid with several epipolar lines; (b) images transformed so that epipolar lines are parallel; (c) images rectified so that epipolar lines are horizontal and in vertial correspondence;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

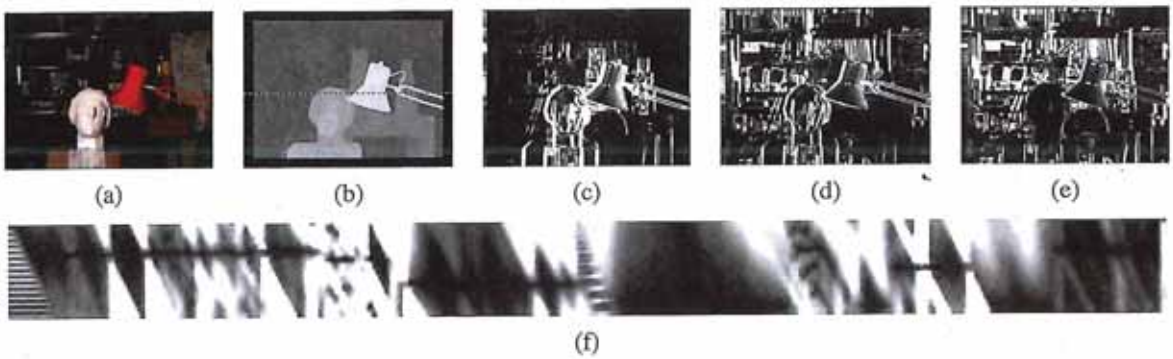


Figure 11.5 Slices through a typical disparity space image (DSI) (Scharstein and Szeliski 2002) © 2002 Springer: (a) original color image; (b) ground truth disparities; (c–e) three (x, y) slices for $d = 10, 16, 21$; (f) an (x, d) slice for $y = 151$ (the dashed line in (b)). Various dark (matching) 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.

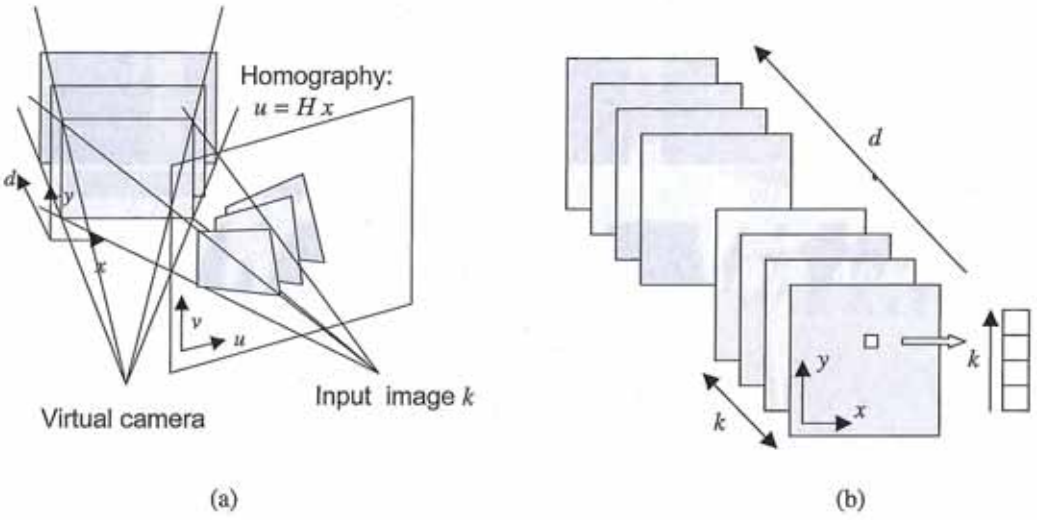


Figure 11.6 Sweeping a set of planes through a scene (Szeliski and Golland 1999) © 1999 Springer: (a) The set of planes seen from a virtual camera induces a set of homographies in any other source (input) camera image. (b) The warped images from all the other cameras can be stacked into a generalized 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

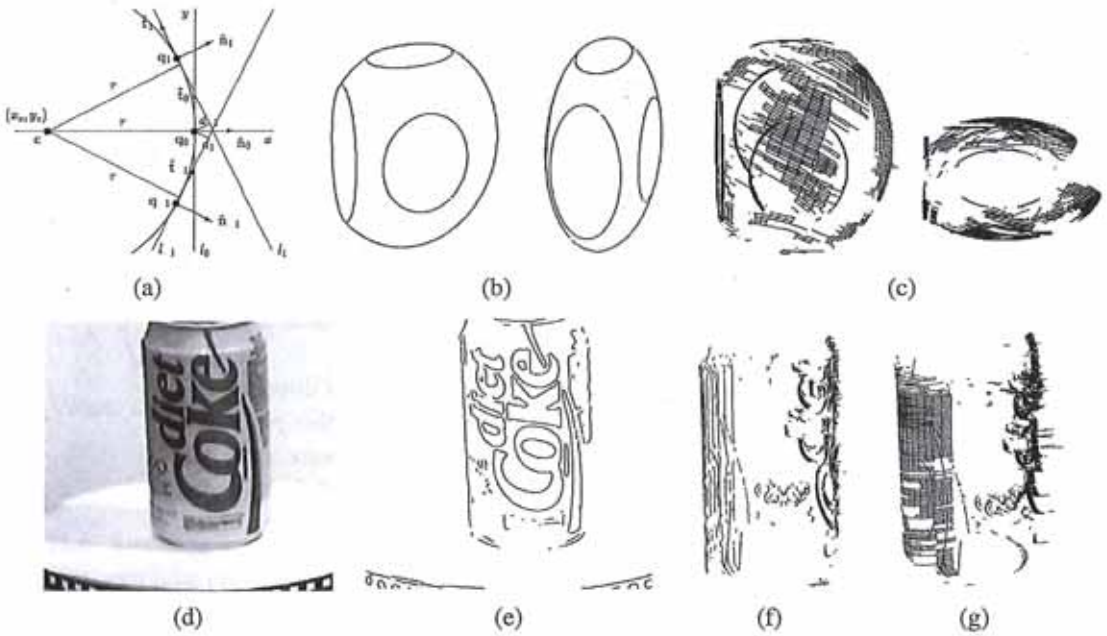


Figure 11.7 Surface reconstruction from occluding contours (Szeliski and Weiss 1998) © 2002 Springer: (a) circular arc fitting in the epipolar plane; (b) synthetic example of an ellipsoid with a truncated side and elliptic surface markings; (c) partially reconstructed surface mesh seen from an oblique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

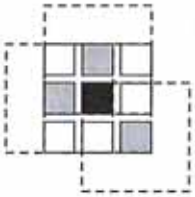


Figure 11.8 Shiftable window (Scharstein and Szeliski 2002) © 2002 Springer. The effect of trying all 3×3 shifted windows around the black pixel is the same as taking the minimum matching score across all *centered* (non-shifted) windows in the same neighborhood. (For clarity, only three of the neighboring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

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).

11.4.1 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, Hanna *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.

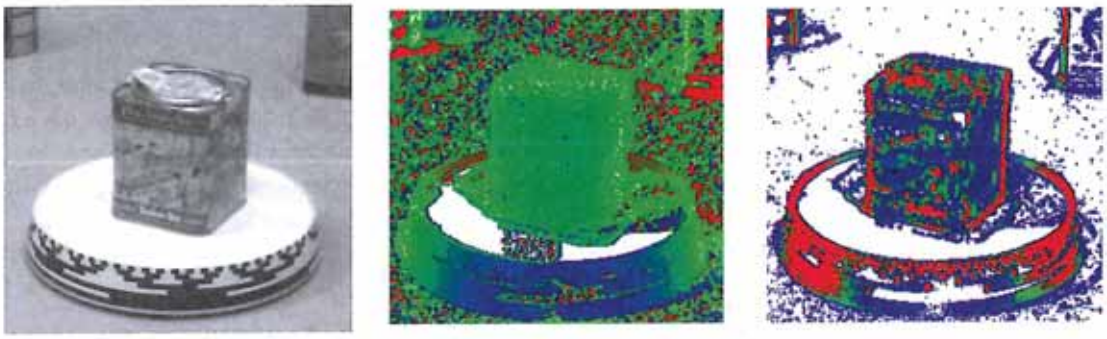
Recently, some questions have been raised about the advisability of fitting correlation curves to integer-sampled matching costs (Shimizu and Okutomi 2001). This situation may even be worse when sampling-insensitive dissimilarity measures are used (Birchfield and Tomasi 1998). These issues are explored in more depth by Szeliski and Scharstein (2004).

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 11.10), 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

$$Var(d) = \frac{\sigma_I^2}{a}, \tag{11.7}$$

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 6.1.4, (8.44), and Appendix B.6.)



(a) (b) (c)

Figure 11.10 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.

11.4.2 *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, Harville *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 11.5.3.

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*, since the 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, Crow *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. Since 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 13.1) is used to synthesize the novel in-between view (Criminisi, Shotton, Blake *et al.* 2003).

11.5 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 Sections 3.7 (3.100–3.102) and 8.4, the objective is to find a solution d that minimizes a global energy,

$$E(d) = E_d(d) + \lambda E_s(d). \tag{11.8}$$

The data term, $E_d(d)$, measures how well the disparity function d agrees with the input image pair. Using our previously defined disparity space image, we define this energy as

$$E_d(d) = \sum_{(x,y)} C(x, y, d(x, y)), \tag{11.9}$$

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

The smoothness term $E_s(d)$ encodes the smoothness assumptions made by the algorithm. To make the optimization computationally tractable, the smoothness term is often restricted to measuring only the differences between neighboring pixels' disparities,

$$E_s(d) = \sum_{(x,y)} \rho(d(x, y) - d(x + 1, y)) + \rho(d(x, y) - d(x, y + 1)), \tag{11.10}$$

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, Torr *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 3.7.1), 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.,

$$\rho_d(d(x, y) - d(x + 1, y)) \cdot \rho_I(\|I(x, y) - I(x + 1, y)\|), \tag{11.11}$$

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, Scharstein and Pal (2007) show how the free parameters in such *conditional random fields* (Section 3.7.2, (3.118)) 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).

More recently, *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 a recent survey paper on MRF inference (Szeliski, Zabih, Scharstein *et al.* 2008) discuss and compare such techniques in more detail.

While global optimization techniques currently produce the best stereo matching results, there are some alternative approaches worth studying.

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 that 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).

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, Hanna *et al.* 1992; Barron, Fleet, and Beauchemin 1994; Szeliski and Coughlan 1997).

11.5.1 Dynamic programming

A different class of global optimization algorithm is based on *dynamic programming*. While the 2D optimization of Equation (11.8) 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, Rao *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

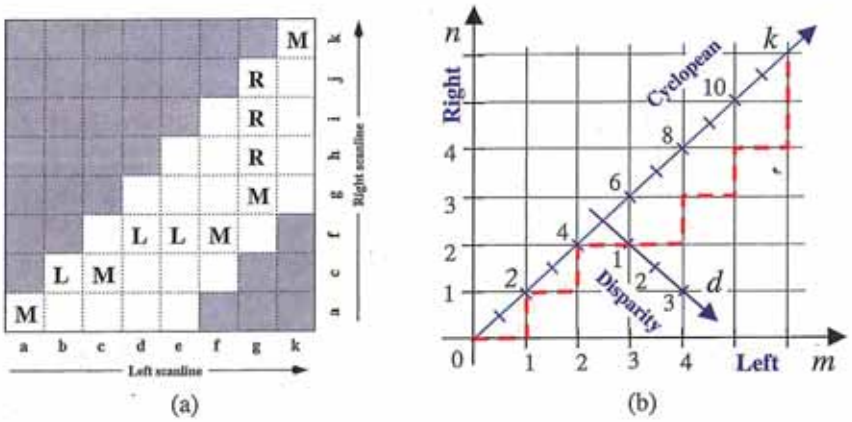


Figure 11.11 Stereo matching using dynamic programming, as illustrated by (a) Scharstein and Szeliski (2002) © 2002 Springer and (b) Kolmogorov, Criminisi, Blake *et al.* (2006). © 2006 IEEE. For each pair of corresponding scanlines, a minimizing path through the matrix of all pairwise matching costs (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.

handled explicitly by assigning a group of pixels in one image to a single pixel in the other image. Figure 11.11 schematically shows how DP works, while Figure 11.5f 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 value

$$C'(m,n) = C(m+n, m-n, y) \tag{11.12}$$

with one of its predecessor cost values. Using the representation shown in Figure 11.11a, which allows for "diagonal" moves, the aggregated match costs can be recursively computed as

$$\begin{aligned} D(m,n,M) &= \min(D(m-1,n-1,M), D(m-1,n,L), D(m-1,n-1,R)) \\ &\quad + C'(m,n) \\ D(m,n,L) &= \min(D(m-1,n-1,M), D(m-1,n,L)) + O \\ D(m,n,R) &= \min(D(m,n-1,M), D(m,n-1,R)) + O, \end{aligned} \tag{11.13}$$

where O is a per-pixel occlusion cost. The aggregation rules corresponding to Figure 11.11b are given by Kolmogorov, Criminisi, Blake *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

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

Figure 11.12 Segmentation-based stereo matching (Zitnick, Kang, Uyttendaele *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.

methods propose ways of addressing the latter (Ohta and Kanade 1985; Belhumeur 1996; Cox, Hingorani, Rao *et al.* 1996; Bobick and Intille 1999; Birchfield and Tomasi 1999; Kolmogorov, Criminisi, Blake *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 (11.10) and simply optimize independent scanlines in the global energy function (11.8), which can easily be done using a recursive algorithm,

$$D(x, y, d) = C(x, y, d) + \min_{d'} \{D(x-1, y, d') + \rho_d(d - d')\}. \tag{11.14}$$

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 (11.8). Unfortunately, it still suffers from the same streaking artifacts as dynamic programming.

A much better approach is to evaluate the cumulative cost function (11.14) from multiple directions, e.g, from the eight cardinal directions, N, E, W, S, NE, SE, SW, NW (Hirschmüller 2008). The resulting *semi-global* optimization performs quite well and is extremely efficient to implement.

Even though dynamic programming and scanline optimization algorithms do not generally produce *the* most accurate stereo reconstructions, when combined with sophisticated aggregation strategies, they can produce very fast and high-quality results.

11.5.2 Segmentation-based techniques

While most stereo matching algorithms perform their computations on a per-pixel basis, some of the more recent techniques first segment the images into regions and then try to label each region with a disparity.

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

(a) (b)

Figure 11.13 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).

before applying smoothness constraints between neighboring segments. Zitnick, Kang, Uyttendaele *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 11.12. 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 11.13.

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. When the algorithm was first introduced in 2006, it was the top ranked algorithm on the evaluation site at http://vision.middlebury.edu/stereo; in early 2010, it still had the top rank on the new evaluation datasets.

The highest ranked 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. Another highly ranked algorithm, by Yang, Wang, Yang *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. After left–right consistency checking to detect occluded pixels, the data terms for low-confidence and occluded pixels are recomputed using segmentation-based plane fits and one or more rounds of hierarchical belief propagation are used to obtain the final disparity estimates.

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, Rother *et al.* 2009). This ability is crucial when attempting to create virtual view interpolation without clinging boundary or tearing artifacts (Zitnick, Kang, Uyttendaele *et al.* 2004) and also to seamlessly insert virtual objects (Taguchi, Wilburn, and Zitnick 2008), as shown in Figure 11.13b.

Figure 11.14 Background replacement using z-keying with a bi-layer segmentation algorithm (Kolmogorov, Criminisi, Blake *et al.* 2006) © 2006 IEEE.

Since new stereo matching algorithms continue to be introduced every year, it is a good idea to periodically check the Middlebury evaluation site at http://vision.middlebury.edu/stereo for a listing of the most recent algorithms to be evaluated.

11.5.3 *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 11.2g.

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, Oda *et al.* 1996; Iddan and Yahav 2001). Off-line systems were also developed for estimating 3D multi-viewpoint geometry from video streams (Section 13.5.4) (Kanade, Rander, and Narayanan 1997; Carranza, Theobalt, Magnor *et al.* 2003; Zitnick, Kang, Uyttendaele *et al.* 2004; Vedula, Baker, and Kanade 2005). Recent advances in highly accurate real-time stereo matching, however, now make it possible to perform z-keying on regular PCs, enabling desktop videoconferencing applications such as those shown in Figure 11.14 (Kolmogorov, Criminisi, Blake *et al.* 2006).

11.6 Multi-view stereo

While matching pairs of images is a useful way of obtaining depth information, matching more images can lead to even better 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.

As we saw in our discussion of plane sweep (Section 11.1.2), it is possible to resample all neighboring k images at each disparity hypothesis d into a generalized disparity space

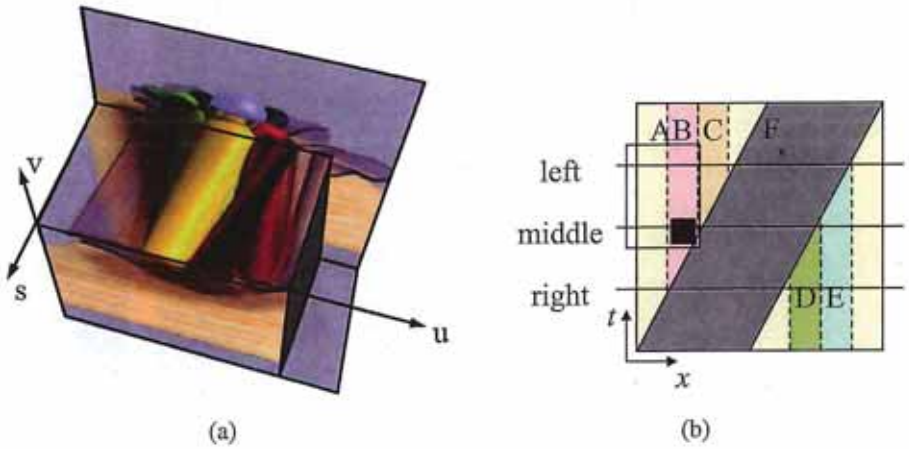


Figure 11.15 Epipolar plane image (EPI) (Gortler, Grzeszczuk, Szeliski *et al.* 1996) © 1996 ACM and a schematic EPI (Kang, Szeliski, and Chai 2001) © 2001 IEEE. (a) The Lumigraph (light field) (Section 13.3) is the 4D space of all light rays passing through a volume of space. Taking a 2D slice results 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 11.16 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 the right image is not being used in matching.

volume $\bar{I}(x, y, d, k)$. The simplest way to take advantage of these additional images is to sum up their differences from the reference image I_r as in (11.4),

$$C(x, y, d) = \sum_k \rho(\bar{I}(x, y, d, k) - I_r(x, y)). \tag{11.15}$$

This is the basis of the well-known sum of summed-squared-difference (SSSD) and SSAD approaches (Okutomi and Kanade 1993; Kang, Webb, Zitnick *et al.* 1995), which can be extended to reason about likely patterns of occlusion (Nakamura, Matsuura, Satoh *et al.* 1996). More recent work by Gallup, Frahm, Mordohai *et al.* (2008) show how to adapt the baselines used to the expected depth in order 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, Zitnick *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 8.13c and 11.15 (Bolles, Baker, and Marimont 1987; Baker and Bolles 1989; Baker 1989). As you can see in Figure 11.15, as a camera translates horizontally (in a standard horizontally rectified geometry), objects at different depths move sideways at a rate inversely proportional to their depth (11.1).[6] Foreground objects occlude background objects,

[6] The four-dimensional generalization of the EPI is the *light field*, which we study in Section 13.3. In principle, there is enough information in a light field to recover both the shape and the BRDF of objects (Soatto, Yezzi, and Jin 2003), although relatively little progress has been made to date on this topic.

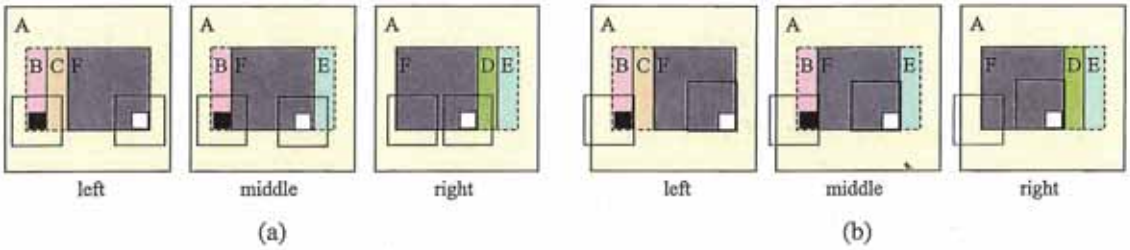


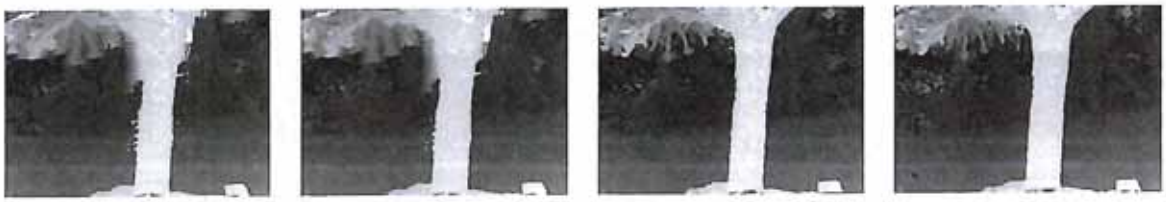
Figure 11.16 Spatio-temporally shiftable windows (Kang, Szeliski, and Chai 2001) © 2001 IEEE: A simple three-image sequence (the middle image is the reference image), which has a moving frontal gray square (marked F) and a stationary background. Regions B, C, D, and E are partially occluded. (a) A regular 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 11.15b shows an EPI corresponding to this sequence and describes in more detail how temporal selection works.

which can be seen as *EPI-strips* (Criminisi, Kang, Swaminathan *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 in order to both reconstruct the 3D scene and make inferences about translucent objects (Tsin, Kang, and Szeliski 2006) and specular reflections (Swaminathan, Kang, Szeliski *et al.* 2002; Criminisi, Kang, Swaminathan *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 11.16a, but also to optionally select a subset of neighboring frames in order to discount those images where the region of interest is occluded, as shown in Figure 11.16b (Kang, Szeliski, and Chai 2001). Figure11.15b 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 11.17, 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



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

Figure 11.17 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.

simultaneously (Szeliski 1999; Kang and Szeliski 2004). The existence of multiple depth maps enables more accurate reasoning about occlusions, as regions which 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 8.13c) while maximizing not only photoconsistency and piecewise disparity smoothness but also the consistency between disparity estimates at different frames. While Szeliski (1999) 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. Newer algorithms that simultaneously estimate multiple disparity maps include papers by Maitre, Shinagawa, and Do (2008) and Zhang, Jia, Wong *et al.* (2008).

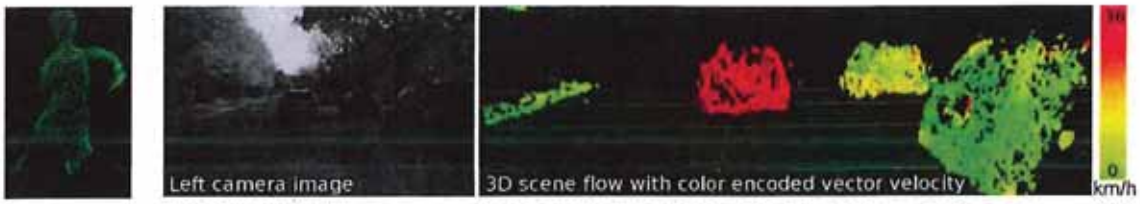
A closely related topic to multi-frame stereo estimation is *scene flow*, in which multiple cameras are used to capture a dynamic scene. The task is then to simultaneously recover the 3D shape of the object at every instant in time and to estimate the full 3D motion of every surface point between frames. Representative papers in this area include those by Vedula, Baker, Rander *et al.* (2005), Zhang and Kambhamettu (2003), Pons, Keriven, and Faugeras (2007), Huguet and Devernay (2007), and Wedel, Rabe, Vaudrey *et al.* (2008). Figure 11.18a shows an image of the 3D scene flow for the tango dancer shown in Figure 11.2h–j, while Figure 11.18b 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 13.5.4), as demonstrated by Vedula, Baker, and Kanade (2005).

11.6.1 Volumetric and 3D surface reconstruction

According to Seitz, Curless, Diebel *et al.* (2006):

> *The goal of multi-view stereo is to reconstruct a complete 3D object model from a collection of images taken from known camera viewpoints.*

The most challenging but potentially most useful variant of multi-view stereo reconstruction is to create globally consistent 3D models. This topic has a long history in computer vision, starting with surface mesh reconstruction techniques such as the one developed by



(a) (b)

Figure 11.18 Three-dimensional scene flow: (a) computed from a multi-camera dome surrounding the dancer shown in Figure 11.2h–j (Vedula, Baker, Rander *et al.* 2005) © 2005 IEEE; (b) computed from stereo cameras mounted on a moving vehicle (Wedel, Rabe, Vaudrey *et al.* 2008) © 2008 Springer.

Fua and Leclerc (1995) (Figure 11.19a). A variety of approaches and representations have been used to solve this problem, including 3D voxel representations (Seitz and Dyer 1999; Szeliski and Golland 1999; De Bonet and Viola 1999; Kutulakos and Seitz 2000; Eisert, Steinbach, and Girod 2000; Slabaugh, Culbertson, Slabaugh *et al.* 2004; Sinha and Pollefeys 2005; Vogiatzis, Hernandez, Torr *et al.* 2007; Hiep, Keriven, Pons *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; Hernandez and Schmitt 2004; Furukawa and Ponce 2009), and multiple depth maps (Kolmogorov and Zabih 2002). Figure 11.19 shows representative examples of 3D object models reconstructed using some of these techniques.

In order to organize and compare all these techniques, Seitz, Curless, Diebel *et al.* (2006) developed a six-point taxonomy that can help classify algorithms according to the *scene representation*, *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, Diebel *et al.* 2006) and the evaluation Web site, http://vision.middlebury.edu/mview/, which contains pointers to even more recent papers and results.

Scene representation. One of the more popular 3D representations is a uniform grid of 3D voxels,[7] which can be reconstructed using a variety of carving (Seitz and Dyer 1999; Kutulakos and Seitz 2000) or optimization (Sinha and Pollefeys 2005; Vogiatzis, Hernandez, Torr *et al.* 2007; Hiep, Keriven, Pons *et al.* 2009) techniques. Level set techniques (Section 5.1.4) 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. Polygonal meshes are another popular representation (Fua and Leclerc 1995; Narayanan, Rander, and Kanade 1998; Isidoro and Sclaroff 2003; Hernandez and Schmitt 2004; Furukawa and Ponce 2009; Hiep, Keriven, Pons *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 1999; 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

[7] For outdoor scenes that go to infinity, a non-uniform gridding of space may be preferable (Slabaugh, Culbertson, Slabaugh *et al.* 2004).

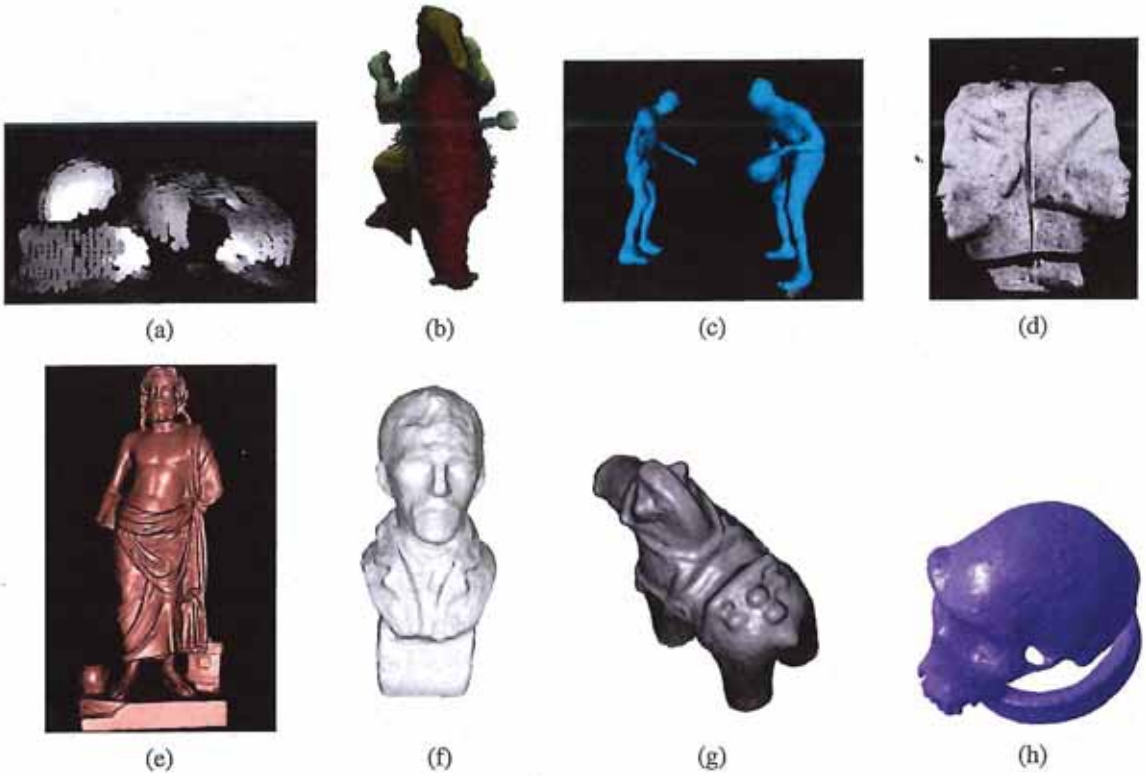


Figure 11.19 Multi-view stereo algorithms: (a) surface-based stereo (Fua and Leclerc 1995); (b) voxel coloring (Seitz and Dyer 1999) © 1999 Springer; (c) depth map merging (Narayanan, Rander, and Kanade 1998); (d) level set evolution (Faugeras and Keriven 1998) © 1998 IEEE; (e) silhouette and stereo fusion (Hernandez 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.

them into a 3D object model (Narayanan, Rander, and Kanade 1998; Furukawa and Ponce 2009; Goesele, Curless, and Seitz 2006; Goesele, Snavely, Curless *et al.* 2007; Furukawa, Curless, Seitz *et al.* 2010).

Photoconsistency measure. As we discussed in (Section 11.3.1), 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, since 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, Curless *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, Hernandez, Torr *et al.* 2007; Hiep, Keriven, Pons *et al.* 2009) are classified as using *geometric visibility models* in the taxonomy of Seitz, Curless, Diebel *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; Hernandez 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, Hernandez, Torr *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.

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, Hernandez, Torr *et al.* 2007; Hiep, Keriven, Pons *et al.* 2009). 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, and Faugeras 2007). In order 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", since 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

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

Figure 11.20 The multi-view stereo data sets captured by Seitz, Curless, Diebel *et al.* (2006) © 2006 Springer. Only (a) and (b) are currently used for evaluation.

both a photoconsistent 3D model and to enclose whatever true 3D object model generated the images.

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.

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 12.2.1, 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, Curless *et al.* (2007). It is also possible to reconstruct sparser representations, such as 3D points and lines, and to interpolate them to full 3D surfaces (Section 12.3.1) (Taylor 2003).

Initialization requirements. One final element discussed by Seitz, Curless, Diebel *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 11.6.2. However, if the algorithm performs a global optimization (Kolev, Klodt, Brox *et al.* 2009; Kolev and Cremers 2009), this dependence on initialization is not an issue.

Empirical evaluation. In order to evaluate the large number of design alternatives in multi-view stereo, Seitz, Curless, Diebel *et al.* (2006) collected a dataset of calibrated images using a spherical gantry. A representative image from each of the six datasets is shown in Figure 11.20, although only the first two datasets have as yet been fully processed and used for evaluation. Figure 11.21 shows the results of running seven different algorithms on the *temple* dataset. As you can see, most of the techniques do an impressive job of capturing the fine details in the columns, although it is also clear that the techniques employ differing amounts of smoothing to achieve these results.

Since the publication of the survey by Seitz, Curless, Diebel *et al.* (2006), the field of

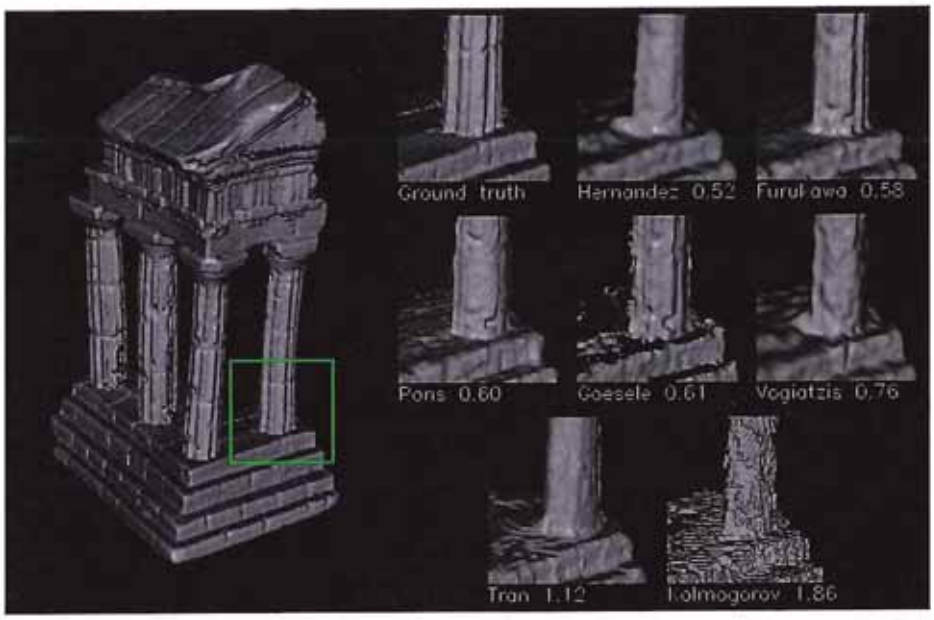


Figure 11.21 Reconstruction results (details) for seven algorithms (Hernandez and Schmitt 2004; Furukawa and Ponce 2009; Pons, Keriven, and Faugeras 2005; Goesele, Curless, and Seitz 2006; Vogiatzis, Torr, and Cipolla 2005; Tran and Davis 2002; Kolmogorov and Zabih 2002) evaluated by Seitz, Curless, Diebel *et al.* (2006) on the 47-image Temple Ring dataset. The numbers underneath each detail image are the accuracy of each of these techniques measured in millimeters.

multi-view stereo has continued to advance at a rapid pace (Strecha, Fransens, and Van Gool 2006; Hernandez, Vogiatzis, and Cipolla 2007; Habbecke and Kobbelt 2007; Furukawa and Ponce 2007; Vogiatzis, Hernandez, Torr *et al.* 2007; Goesele, Snavely, Curless *et al.* 2007; Sinha, Mordohai, and Pollefeys 2007; Gargallo, Prados, and Sturm 2007; Merrell, Akbarzadeh, Wang *et al.* 2007; Zach, Pock, and Bischof 2007b; Furukawa and Ponce 2008; Hornung, Zeng, and Kobbelt 2008; Bradley, Boubekeur, and Heidrich 2008; Zach 2008; Campbell, Vogiatzis, Hernández *et al.* 2008; Kolev, Klodt, Brox *et al.* 2009; Hiep, Keriven, Pons *et al.* 2009; Furukawa, Curless, Seitz *et al.* 2010). The multi-view stereo evaluation site, http://vision.middlebury.edu/mview/, provides quantitative results for these algorithms along with pointers to where to find these papers.

11.6.2 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, Matusik *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, since the volume is maximal with respect

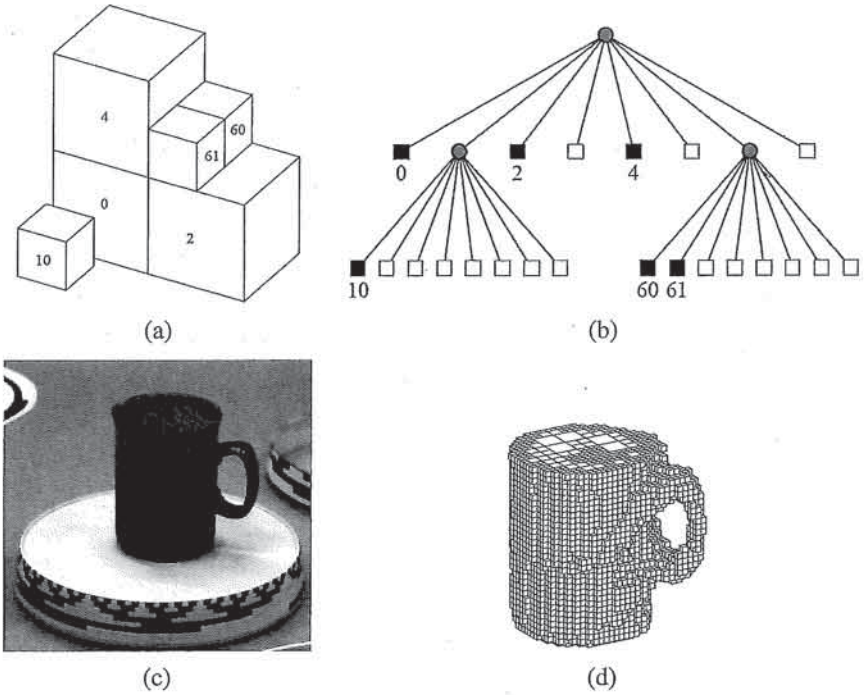


Figure 11.22 Volumetric octree reconstruction from binary silhouettes (Szeliski 1993) © 1993 Elsevier: (a) octree representation and its corresponding (b) tree structure; (c) input image of an object on a turntable; (d) computed 3D volumetric octree model.

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, Rushmeier *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 11.22a–b show an example of a 3D octree 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 those by 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, 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 11.22d 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, Raskar *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 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, Matusik *et al.* 2008).

11.7 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, Diebel *et al.* 2006) 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; 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).

The topic of computing epipolar geometry and pre-rectifying images is covered in Sections 7.2 and 11.1 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, Tang *et al.* 2004; Zheng, Kang, Cohen *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 11.3.1. 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 11.4 and surveyed by Gong, Yang, Wang *et al.* (2007) and Tombari,

Mattoccia, Di Stefano *et al.* (2008). More commonly, global optimization frameworks are used to compute the best disparity field, as described in Section 11.5. These techniques include dynamic programming and truly global optimization algorithms, such as graph cuts and loopy belief propagation. Because the literature on this is so extensive, it is described in more detail in Section 11.5. A good place to find pointers to the latest results in this field is the Middlebury Stereo Vision Page at http://vision.middlebury.edu/stereo.

Algorithms for multi-view stereo typically fall into two categories. The first include algorithms that compute traditional depth maps using several images for computing photoconsistency measures (Okutomi and Kanade 1993; Kang, Webb, Zitnick *et al.* 1995; Nakamura, Matsuura, Satoh *et al.* 1996; Szeliski and Golland 1999; Kang, Szeliski, and Chai 2001; Vaish, Szeliski, Zitnick *et al.* 2006; Gallup, Frahm, Mordohai *et al.* 2008). Optionally, some of these techniques compute multiple depth maps and use additional constraints to encourage the different depth maps to be consistent (Szeliski 1999; Kolmogorov and Zabih 2002; Kang and Szeliski 2004; Maitre, Shinagawa, and Do 2008; Zhang, Jia, Wong *et al.* 2008).

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 11.6.1, the survey by Seitz, Curless, Diebel *et al.* (2006), and the on-line evaluation Web site at http://vision.middlebury.edu/mview/.

11.8 Exercises

Ex 11.1: Stereo pair rectification Implement the following simple algorithm (Section 11.1.1):

1. Rotate both cameras so that they are looking perpendicular to the line joining the two camera centers c_0 and 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 11.2: Rigid direct alignment Modify your spline-based or optical flow motion estimator from Exercise 8.4 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 11.3: Shape from profiles Reconstruct a surface model from a series of edge images (Section 11.2.1).

1. Extract edges and link them (Exercises 4.7–4.8).
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 (11.5).
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 11.4: Plane sweep Implement a plane sweep algorithm (Section 11.1.2).

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 11.3.1) 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 pre-computed 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 11.3.1.

Ex 11.5: Aggregation and window-based stereo Implement one or more of the matching cost aggregation strategies described in Section 11.4:

- 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 11.6: Optimization-based stereo Compute the disparity space image (DSI) volume using one of the techniques you implemented in Exercise 11.4 and then implement one (or more) of the global optimization techniques described in Section 11.5 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 (http://vision.middlebury.edu/stereo);
- loopy belief propagation (Appendix B.5.3).

Evaluate your algorithm by running it on the Middlebury stereo data sets.

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 11.7: 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 data set.

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 13.1 and Exercises 13.1–13.4.

Ex 11.8: Multi-frame stereo Extend one of your previous techniques to use multiple input frames (Section 11.6) 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 data sets.

You can also try to simultaneously estimate multiple depth maps and make them consistent (Kolmogorov and Zabih 2002; Kang and Szeliski 2004).

Test your algorithms out on some standard multi-view data sets.

Ex 11.9: Volumetric stereo Implement voxel coloring (Seitz and Dyer 1999) as a simple extension to the plane sweep algorithm you implemented in Exercise 11.4.

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, Slabaugh *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 11.10: Depth map merging Use the technique you developed for multi-frame stereo in Exercise 11.8 or a different technique, such as the one described by Goesele, Snavely, Curless *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 11.11: Shape from silhouettes Build a silhouette-based volume reconstruction algorithm (Section 11.6.2). Use an octree or some other representation of your choosing.

Index