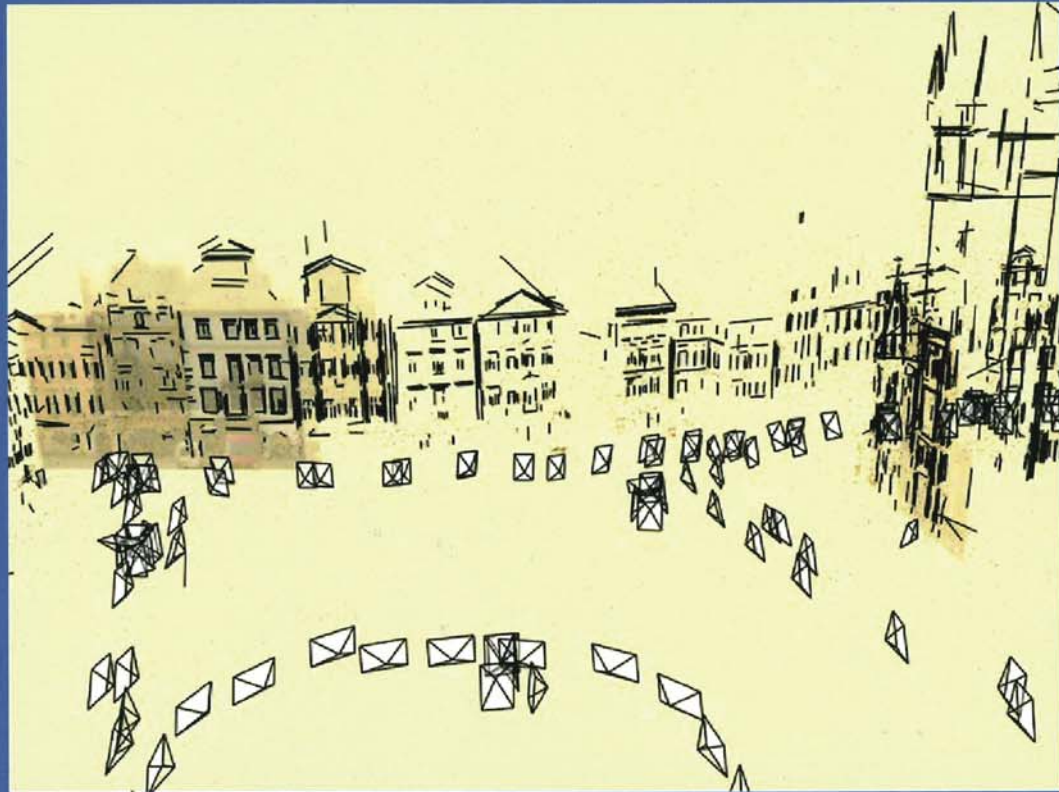


Computer Vision

Algorithms and Applications



Richard Szeliski

Texts in Computer Science

Editors

David Gries

Fred B. Schneider

For further volumes:
www.springer.com/series/3191

Richard Szeliski

Computer Vision

Algorithms and Applications

 Springer

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

Series Editors

David Gries
Department of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853-7501, USA

Fred B. Schneider
Department of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853-7501, USA

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

© Springer-Verlag London Limited 2011

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licenses issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

The use of registered names, trademarks, etc., in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

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

Chapter 2

Image formation

| | | |
|-------|--|----|
| 2.1 | Geometric primitives and transformations | 29 |
| 2.1.1 | Geometric primitives | 29 |
| 2.1.2 | 2D transformations | 33 |
| 2.1.3 | 3D transformations | 36 |
| 2.1.4 | 3D rotations | 37 |
| 2.1.5 | 3D to 2D projections | 42 |
| 2.1.6 | Lens distortions | 52 |
| 2.2 | Photometric image formation | 54 |
| 2.2.1 | Lighting | 54 |
| 2.2.2 | Reflectance and shading | 55 |
| 2.2.3 | Optics | 61 |
| 2.3 | The digital camera | 65 |
| 2.3.1 | Sampling and aliasing | 69 |
| 2.3.2 | Color | 71 |
| 2.3.3 | Compression | 80 |
| 2.4 | Additional reading | 82 |
| 2.5 | Exercises | 82 |

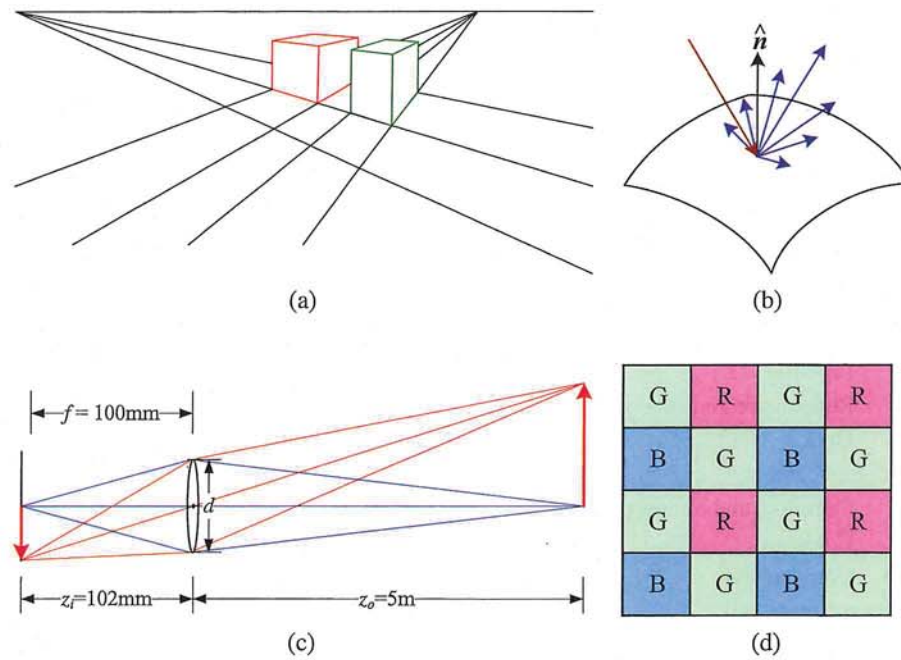


Figure 2.1 A few components of the image formation process: (a) perspective projection; (b) light scattering when hitting a surface; (c) lens optics; (d) Bayer color filter array.

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 Schaffer 1996; Oppenheim, Schaffer, 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

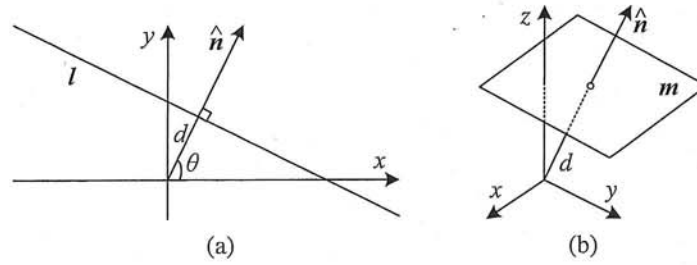


Figure 2.2 (a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal \hat{n} and distance to the origin d .

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, $x = (x, y) \in \mathcal{R}^2$, or alternatively,

$$x = \begin{bmatrix} x \\ y \end{bmatrix}. \quad (2.1)$$

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

2D points can also be represented using *homogeneous coordinates*, $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$, 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{x} can be converted back into an *inhomogeneous* vector x by dividing through by the last element \tilde{w} , i.e.,

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

where $\bar{x} = (x, y, 1)$ 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 $\tilde{l} = (a, b, c)$. The corresponding *line equation* is

$$\bar{x} \cdot \tilde{l} = ax + by + c = 0. \quad (2.3)$$

We can normalize the line equation vector so that $l = (\hat{n}_x, \hat{n}_y, d) = (\hat{n}, d)$ with $\|\hat{n}\| = 1$. In this case, \hat{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* $\tilde{l} = (0, 0, 1)$, which includes all (ideal) points at infinity.)

We can also express \hat{n} as a function of rotation angle θ , $\hat{n} = (\hat{n}_x, \hat{n}_y) = (\cos \theta, \sin \theta)$ (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

$$\tilde{x} = \tilde{l}_1 \times \tilde{l}_2, \quad (2.4)$$

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

$$\tilde{l} = \tilde{x}_1 \times \tilde{x}_2. \quad (2.5)$$

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

$$\tilde{x}^T Q \tilde{x} = 0. \quad (2.6)$$

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 $\mathbf{x} = (x, y, z) \in \mathcal{R}^3$ or homogeneous coordinates $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$. As before, it is sometimes useful to denote a 3D point using the augmented vector $\tilde{\mathbf{x}} = (x, y, z, 1)$ with $\tilde{\mathbf{x}} = \tilde{w}\tilde{\mathbf{x}}$.

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

$$\tilde{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0. \quad (2.7)$$

We can also normalize the plane equation as $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\mathbf{n}}, d)$ with $\|\hat{\mathbf{n}}\| = 1$. In this case, $\hat{\mathbf{n}}$ 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* $\tilde{\mathbf{m}} = (0, 0, 0, 1)$, 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 $\hat{\mathbf{n}}$ as a function of two angles (θ, ϕ) ,

$$\hat{\mathbf{n}} = (\cos \theta \cos \phi, \sin \theta \cos \phi, \sin \phi), \quad (2.8)$$

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, (\mathbf{p}, \mathbf{q}) . Any other point on the line can be expressed as a linear combination of these two points

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

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

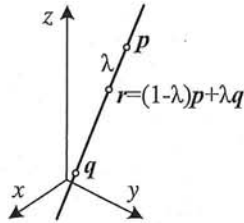


Figure 2.3 3D line equation, $r = (1 - \lambda)p + \lambda q$.

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

$$\tilde{r} = \mu\tilde{p} + \lambda\tilde{q}. \quad (2.10)$$

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

$$r = p + \lambda\hat{d}. \quad (2.11)$$

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

$$L = \tilde{p}\tilde{q}^T - \tilde{q}\tilde{p}^T, \quad (2.12)$$

where \tilde{p} and \tilde{q} are *any* two (non-identical) points on the line. This representation has only four degrees of freedom, since L is homogeneous and also satisfies $\det(L) = 0$, 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.

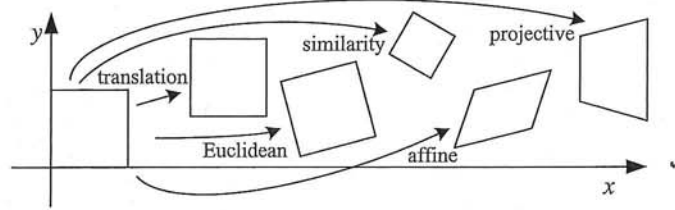


Figure 2.4 Basic set of 2D planar transformations.

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

$$\bar{x}^T Q \bar{x} = 0 \quad (2.13)$$

(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

$$x' = [I \quad t] \bar{x} \quad (2.14)$$

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

$$\bar{x}' = \begin{bmatrix} I & t \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{x} \quad (2.15)$$

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

$$x' = [R \quad t] \bar{x} \quad (2.16)$$

where

$$R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (2.17)$$

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 $x' = sR\bar{x} + t$ where s is an arbitrary scale factor. It can also be written as

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

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.,

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

Parallel lines remain parallel under affine transformations.

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

$$\tilde{x}' = \tilde{H}\tilde{x}, \quad (2.20)$$

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 \tilde{x}' must be normalized in order to obtain an inhomogeneous result x , i.e.,

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

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

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

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 | $[I \mid t]_{2 \times 3}$ | 2 | orientation |  |
| rigid (Euclidean) | $[R \mid t]_{2 \times 3}$ | 3 | lengths |  |
| similarity | $[sR \mid t]_{2 \times 3}$ | 4 | angles |  |
| affine | $[A]_{2 \times 3}$ | 6 | parallelism |  |
| projective | $[\tilde{H}]_{3 \times 3}$ | 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 $[0^T \ 1]$ 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 | $\begin{bmatrix} I & & t \end{bmatrix}_{3 \times 4}$ | 3 | orientation |  |
| rigid (Euclidean) | $\begin{bmatrix} R & & t \end{bmatrix}_{3 \times 4}$ | 6 | lengths |  |
| similarity | $\begin{bmatrix} sR & & t \end{bmatrix}_{3 \times 4}$ | 7 | angles |  |
| affine | $\begin{bmatrix} A \end{bmatrix}_{3 \times 4}$ | 12 | parallelism |  |
| projective | $\begin{bmatrix} \tilde{H} \end{bmatrix}_{4 \times 4}$ | 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 $[0^T \ 1]$ 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 $\mathbf{x}' = \mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} I & | & t \end{bmatrix} \bar{\mathbf{x}} \quad (2.23)$$

where 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 $\mathbf{x}' = R\mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} R & | & t \end{bmatrix} \bar{\mathbf{x}} \quad (2.24)$$

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

$$\mathbf{x}' = R(\mathbf{x} - \mathbf{c}) = R\mathbf{x} - R\mathbf{c}, \quad (2.25)$$

where \mathbf{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

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

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.,

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

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,

$$\tilde{x}' = \tilde{H}\tilde{x}, \quad (2.28)$$

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

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

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

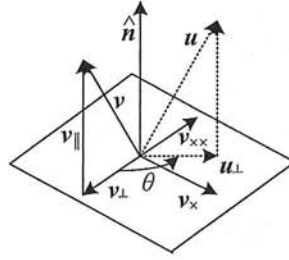


Figure 2.5 Rotation around an axis \hat{n} by an angle θ .

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

$$v_{\perp} = v - v_{\parallel} = (I - \hat{n}\hat{n}^T)v. \quad (2.30)$$

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

$$v_{\times} = \hat{n} \times v = [\hat{n}]_{\times} v, \quad (2.31)$$

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

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

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

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

and hence

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

We can now compute the in-plane component of the rotated vector u as

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

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

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

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

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

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

The product of the axis \hat{n} and angle θ , $\omega = \theta\hat{n} = (\omega_x, \omega_y, \omega_z)$, 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{n}, θ) and $(-\hat{n}, -\theta)$ 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

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

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

$$\frac{\partial \mathbf{R}\mathbf{v}}{\partial \boldsymbol{\omega}^T} = -[\mathbf{v}]_{\times} = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}. \quad (2.36)$$

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 $k \rightarrow \infty$, we obtain

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

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

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

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 $\mathbf{q} = (q_x, q_y, q_z, q_w)$ or $\mathbf{q} = (x, y, z, w)$ for short. Unit quaternions live on the unit sphere $\|\mathbf{q}\| = 1$ and *antipodal* (opposite sign) quaternions, \mathbf{q} and $-\mathbf{q}$, 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" $\mathbf{q}_o = (0, 0, 0, 1)$. For these and other reasons given below, quaternions are a very popular representation for pose and for pose interpolation in computer graphics (Shoemake 1985).

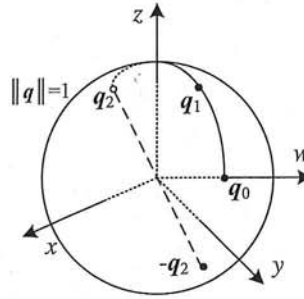


Figure 2.6 Unit quaternions live on the unit sphere $\|q\| = 1$. This figure shows a smooth trajectory through the three quaternions q_0 , q_1 , and q_2 . The *antipodal* point to q_2 , namely $-q_2$, represents the same rotation as q_2 .

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

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

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

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

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

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

We thus obtain

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

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

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

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. $q_r = q_1/q_0 = (v_r, w_r)$
2. if $w_r < 0$ then $q_r \leftarrow -q_r$
3. $\theta_r = 2 \tan^{-1}(\|v_r\|/w_r)$
4. $\hat{n}_r = \mathcal{N}(v_r) = v_r/\|v_r\|$
5. $\theta_\alpha = \alpha \theta_r$
6. $q_\alpha = (\sin \frac{\theta_\alpha}{2} \hat{n}_r, \cos \frac{\theta_\alpha}{2})$
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 v or w (but not both!). (You can verify this has the desired effect of transposing the R matrix in (2.41).) Thus, we can also define *quaternion division* as

$$q_2 = q_0/q_1 = q_0 q_1^{-1} = (v_0 \times v_1 + w_0 v_1 - w_1 v_0, -w_0 w_1 - v_0 \cdot v_1). \quad (2.43)$$

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,

$$q_2 = q_r^\alpha q_0, \quad (2.44)$$

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

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

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 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 p to obtain the 2D point x . (In this section, we use p to denote 3D points and x to denote 2D points.) This can be written as

$$x = [I_{2 \times 2} | \mathbf{0}] p. \quad (2.46)$$

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

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{p}, \quad (2.47)$$

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,

$$x = [sI_{2 \times 2} | \mathbf{0}] p. \quad (2.48)$$

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

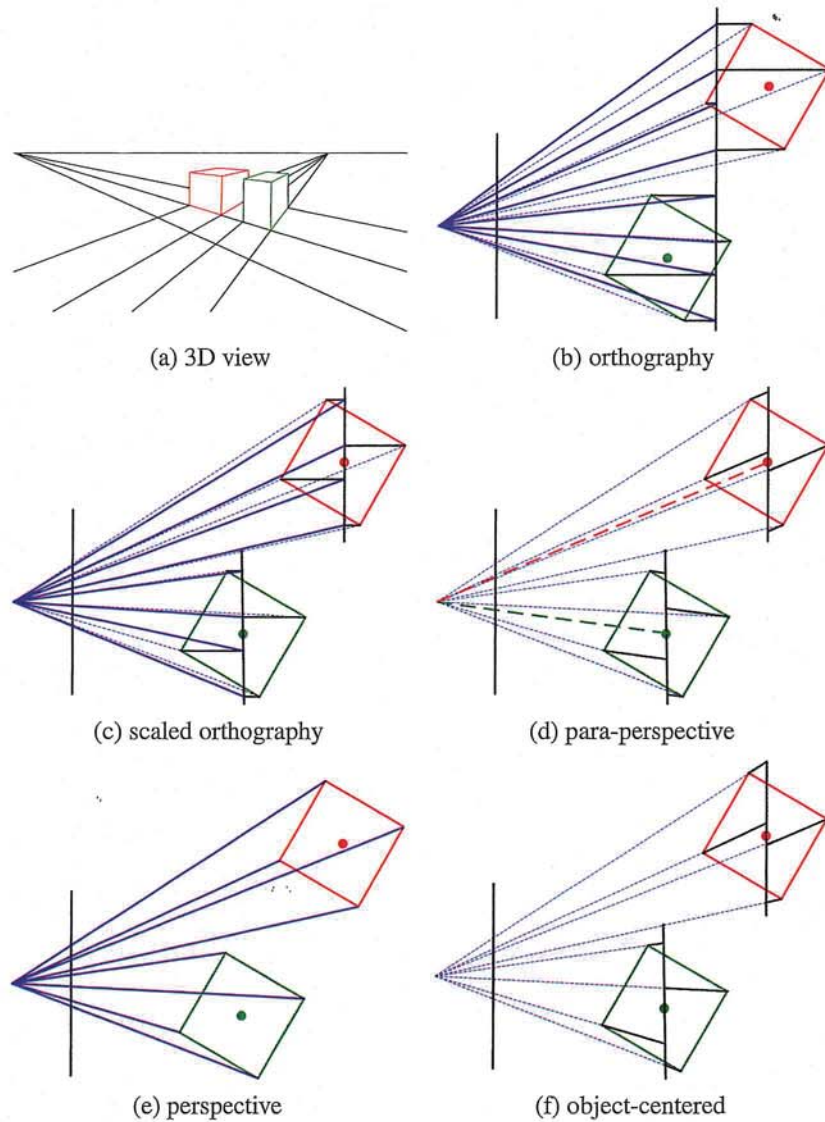


Figure 2.7 Commonly used projection models: (a) 3D view of world, (b) orthography, (c) scaled orthography, (d) para-perspective, (e) perspective, (f) object-centered. Each diagram shows a top-down view of the projection. Note how parallel lines on the ground plane and box sides remain parallel in the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

$$\tilde{\mathbf{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}. \quad (2.49)$$

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

$$\tilde{\mathbf{x}} = \mathcal{P}_z(\mathbf{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}. \quad (2.50)$$

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

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{p}}, \quad (2.51)$$

i.e., we drop the w component of \mathbf{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

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{\text{far}}/z_{\text{range}} & z_{\text{near}}z_{\text{far}}/z_{\text{range}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{p}}, \quad (2.52)$$

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

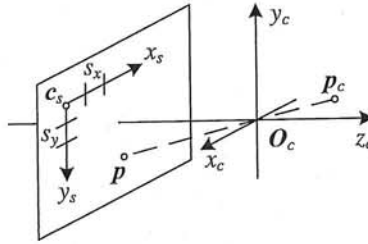


Figure 2.8 Projection of a 3D camera-centered point p_c onto the sensor planes at location p . O_c is the camera center (nodal point), c_s is the 3D origin of the sensor plane coordinate system, and s_x and s_y are the pixel spacings.

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 $z_{\text{near}} = 1$, $z_{\text{far}} \rightarrow \infty$, 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 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 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 K , which is used to map 3D camera-centered points p_c to 2D pixel coordinates \tilde{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 O_c with an origin c_s and a 3D rotation R_s (Figure 2.8).

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

$$\mathbf{p} = [\mathbf{R}_s \mid \mathbf{c}_s] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \tilde{\mathbf{x}}_s. \quad (2.53)$$

The first two columns of the 3×3 matrix \mathbf{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 \mathbf{c}_s .

The matrix \mathbf{M}_s is parameterized by eight unknowns: the three parameters describing the rotation \mathbf{R}_s , the three parameters describing the translation \mathbf{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 \mathbf{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 \mathbf{p} and the 3D camera-centered point \mathbf{p}_c is given by an unknown scaling s , $\mathbf{p} = s\mathbf{p}_c$. We can therefore write the complete projection between \mathbf{p}_c and a homogeneous version of the pixel address $\tilde{\mathbf{x}}_s$ as

$$\tilde{\mathbf{x}}_s = \alpha \mathbf{M}_s^{-1} \mathbf{p}_c = \mathbf{K} \mathbf{p}_c. \quad (2.54)$$

The 3×3 matrix \mathbf{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 \mathbf{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 \mathbf{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 \mathbf{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 (\mathbf{K}) and extrinsic (\mathbf{R}, \mathbf{t}) camera parameters simultaneously using a series of measurements,

$$\tilde{\mathbf{x}}_s = \mathbf{K} [\mathbf{R} \mid \mathbf{t}] \mathbf{p}_w = \mathbf{P} \mathbf{p}_w, \quad (2.55)$$

where \mathbf{p}_w are known 3D world coordinates and

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \quad (2.56)$$

is known as the *camera matrix*. Inspecting this equation, we see that we can post-multiply \mathbf{K} by \mathbf{R}_1 and pre-multiply $[\mathbf{R}|\mathbf{t}]$ by \mathbf{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.

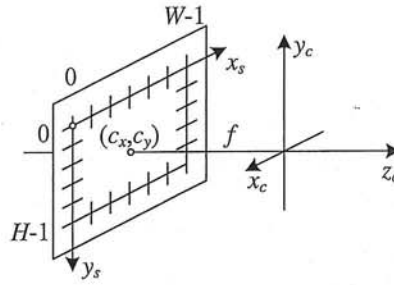


Figure 2.9 Simplified camera intrinsics showing the focal length f and the optical center (c_x, c_y) . The image width and height are W and H .

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

There are several ways to write the upper-triangular form of K . One possibility is

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.57)$$

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

$$K = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.58)$$

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 $s = 0$,

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.59)$$

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.

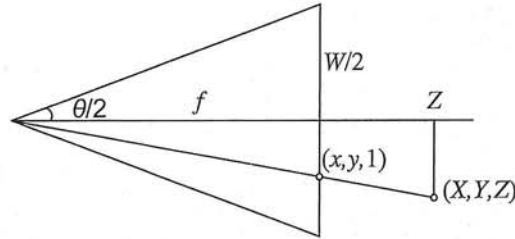


Figure 2.10 Central projection, showing the relationship between the 3D and 2D coordinates, p and x , as well as the relationship between the focal length f , image width W , and the field of view θ .

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

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

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 \mathbf{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*,

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

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

$$f^{-1} = \tan \frac{\theta}{2}. \quad (2.62)$$

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 K , we can put the camera intrinsics and extrinsics together to obtain a single 3×4 camera matrix

$$P = K [R \mid t]. \quad (2.63)$$

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

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

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

$$x_s \sim \tilde{P} \bar{p}_w, \quad (2.65)$$

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

Plane plus parallax (projective depth)

In general, when using the 4×4 matrix \tilde{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{P} as $p_3 = s_3[\hat{n}_0|c_0]$, where $\|\hat{n}_0\| = 1$. We then have the equation

$$d = \frac{s_3}{z} (\hat{n}_0 \cdot p_w + c_0), \quad (2.66)$$

where $z = p_2 \cdot \bar{p}_w = r_z \cdot (p_w - c)$ is the distance of 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 p_w from the *reference plane* $\hat{n}_0 \cdot p_w + c_0 = 0$ (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 $\hat{n}_0 = 0$ 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{P} matrix so that we can map pixels plus disparity directly back to 3D points,

$$\bar{p}_w = \tilde{P}^{-1} x_s. \quad (2.67)$$

In general, we can choose \tilde{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

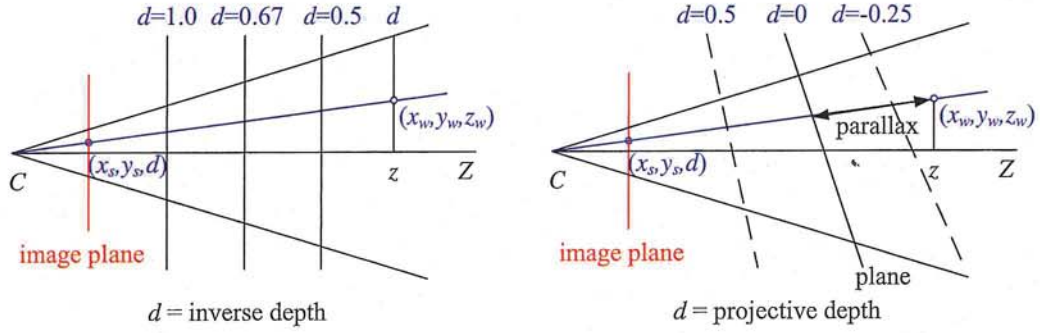


Figure 2.11 Regular disparity (inverse depth) and projective depth (parallax from a reference plane).

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 $\tilde{P} = \tilde{K}E$ from (2.64), we can write the projection from world to screen coordinates as

$$\tilde{x}_0 \sim \tilde{K}_0 E_0 p = \tilde{P}_0 p. \quad (2.68)$$

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 p using

$$p \sim E_0^{-1} \tilde{K}_0^{-1} \tilde{x}_0 \quad (2.69)$$

and then project it into another image yielding

$$\tilde{x}_1 \sim \tilde{K}_1 E_1 p = \tilde{K}_1 E_1 E_0^{-1} \tilde{K}_0^{-1} \tilde{x}_0 = \tilde{P}_1 \tilde{P}_0^{-1} \tilde{x}_0 = M_{10} \tilde{x}_0. \quad (2.70)$$

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 P_0 in (2.64) with a general *plane equation*, $\hat{n}_0 \cdot p + c_0$ 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 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

$$\tilde{x}_1 \sim \tilde{H}_{10} \tilde{x}_0, \quad (2.71)$$

where \tilde{H}_{10} is a general 3×3 homography matrix and \tilde{x}_1 and \tilde{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).

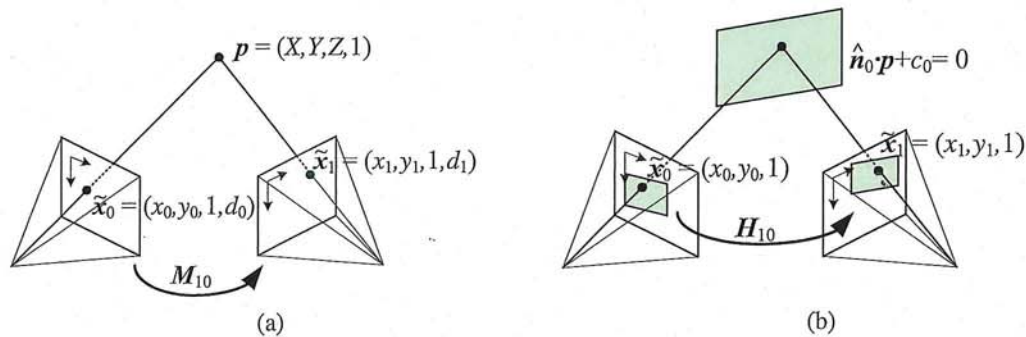


Figure 2.12 A point is projected into two images: (a) relationship between the 3D point coordinate $(X, Y, Z, 1)$ and the 2D projected point $(x, y, 1, d)$; (b) planar homography induced by points all lying on a common plane $\hat{n}_0 \cdot p + c_0 = 0$.

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 $t_0 = t_1$. In this case, we can write

$$\tilde{x}_1 \sim K_1 R_1 R_0^{-1} K_0^{-1} \tilde{x}_0 = K_1 R_{10} K_0^{-1} \tilde{x}_0, \quad (2.72)$$

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 K (2.59),

$$x_s = f \frac{r_x \cdot p + t_x}{r_z \cdot p + t_z} + c_x \quad (2.73)$$

$$y_s = f \frac{r_y \cdot p + t_y}{r_z \cdot p + t_z} + c_y, \quad (2.74)$$

where r_x , r_y , and r_z are the three rows of R . If the distance to the object center $t_z \gg \|p\|$ (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

$$x_s = s \frac{r_x \cdot \mathbf{p} + t_x}{1 + \eta_z r_z \cdot \mathbf{p}} + c_x \quad (2.75)$$

$$y_s = s \frac{r_y \cdot \mathbf{p} + t_y}{1 + \eta_z r_z \cdot \mathbf{p}} + c_y \quad (2.76)$$

(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 \mathbf{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 $\eta_z \rightarrow 0$ (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.,

$$\begin{aligned} x_c &= \frac{r_x \cdot \mathbf{p} + t_x}{r_z \cdot \mathbf{p} + t_z} \\ y_c &= \frac{r_y \cdot \mathbf{p} + t_y}{r_z \cdot \mathbf{p} + t_z} \end{aligned} \quad (2.77)$$

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.,

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

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.

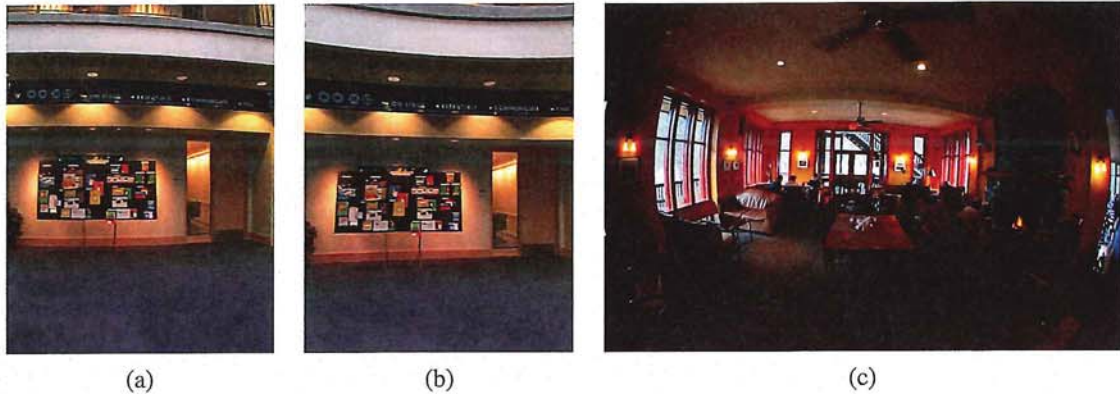


Figure 2.13 Radial lens distortions: (a) barrel, (b) pincushion, and (c) fisheye. The fisheye image spans almost 180° from side-to-side.

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

$$\begin{aligned} x_s &= fx'_c + c_x \\ y_s &= fy'_c + c_y. \end{aligned} \quad (2.79)$$

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.

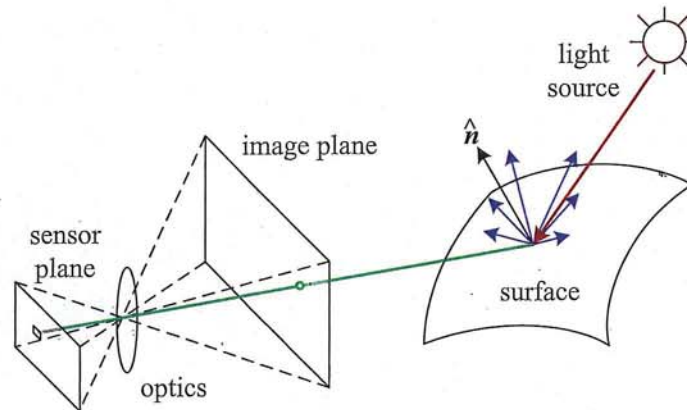


Figure 2.14 A simplified model of photometric image formation. Light is emitted by one or more light sources and is then reflected from an object's surface. A portion of this light is directed towards the camera. This simplified model ignores multiple reflections, which often occur in real-world scenes.

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 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),

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

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).

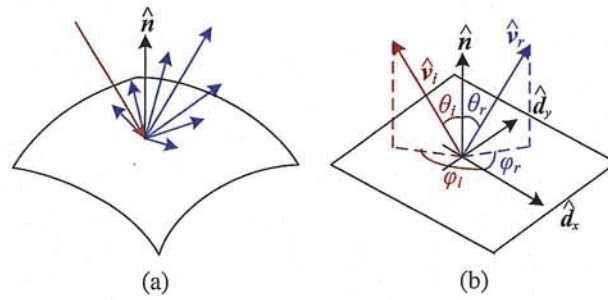


Figure 2.15 (a) Light scatters when it hits a surface. (b) The bidirectional reflectance distribution function (BRDF) $f(\theta_i, \phi_i, \theta_r, \phi_r)$ is parameterized by the angles that the incident, \hat{v}_i , and reflected, \hat{v}_r , light ray directions make with the local surface coordinate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

direction \hat{v}_i is emitted in a *reflected* direction \hat{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

$$f_r(\theta_i, \phi_i, \theta_r, \phi_r; \lambda). \quad (2.81)$$

The BRDF is *reciprocal*, i.e., because of the physics of light transport, you can interchange the roles of \hat{v}_i and \hat{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

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

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

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

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

where

$$\cos^+ \theta_i = \max(0, \cos \theta_i). \quad (2.84)$$

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

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

BRDFs for a given surface can be obtained through physical modeling (Torralba and Sparrow 1967; Cook and Torralba 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,

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

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

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

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

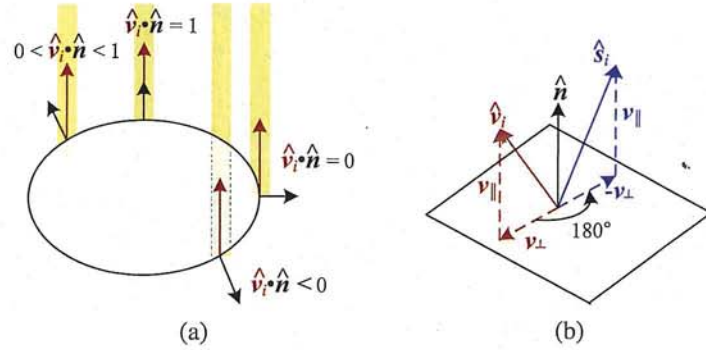


Figure 2.17 (a) The diminution of returned light caused by *foreshortening* depends on $\hat{v}_i \cdot \hat{n}$, the cosine of the angle between the incident light direction \hat{v}_i and the surface normal \hat{n} . (b) Mirror (specular) reflection: The incident light ray direction \hat{v}_i is reflected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

where

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

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{n} . Using the same notation as in Equations (2.29–2.30), we can compute the *specular reflection* direction \hat{s}_i as

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

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

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

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

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

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

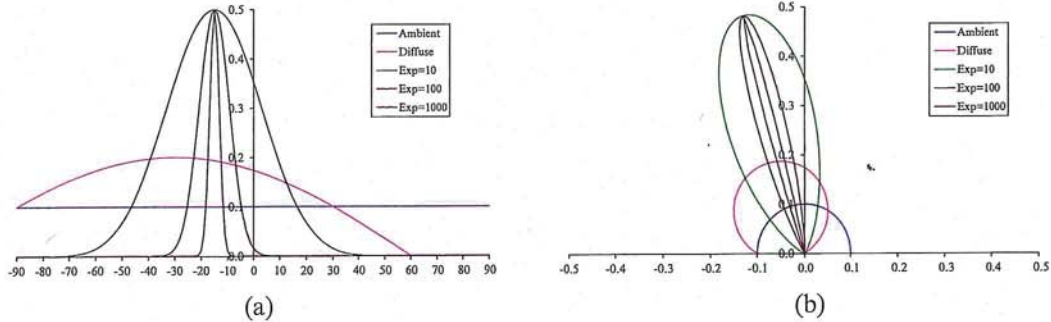


Figure 2.18 Cross-section through a Phong shading model BRDF for a fixed incident illumination direction: (a) component values as a function of angle away from surface normal; (b) polar plot. The value of the Phong exponent k_e is indicated by the “Exp” labels and the light source is at an angle of 30°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

$$f_a(\lambda) = k_a(\lambda)L_a(\lambda). \quad (2.92)$$

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

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

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{v}_i , while the specular component depends on the relative angle between the viewer v_r and the specular reflection direction \hat{s}_i (which itself depends on the incoming light direction \hat{v}_i and the surface normal \hat{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 6

Feature-based alignment

| | | |
|-------|---|-----|
| 6.1 | 2D and 3D feature-based alignment | 275 |
| 6.1.1 | 2D alignment using least squares | 275 |
| 6.1.2 | <i>Application: Panography</i> | 277 |
| 6.1.3 | Iterative algorithms | 278 |
| 6.1.4 | Robust least squares and RANSAC | 281 |
| 6.1.5 | 3D alignment | 283 |
| 6.2 | Pose estimation | 284 |
| 6.2.1 | Linear algorithms | 284 |
| 6.2.2 | Iterative algorithms | 286 |
| 6.2.3 | <i>Application: Augmented reality</i> | 287 |
| 6.3 | Geometric intrinsic calibration | 288 |
| 6.3.1 | Calibration patterns | 289 |
| 6.3.2 | Vanishing points | 290 |
| 6.3.3 | <i>Application: Single view metrology</i> | 292 |
| 6.3.4 | Rotational motion | 293 |
| 6.3.5 | Radial distortion | 295 |
| 6.4 | Additional reading | 296 |
| 6.5 | Exercises | 296 |

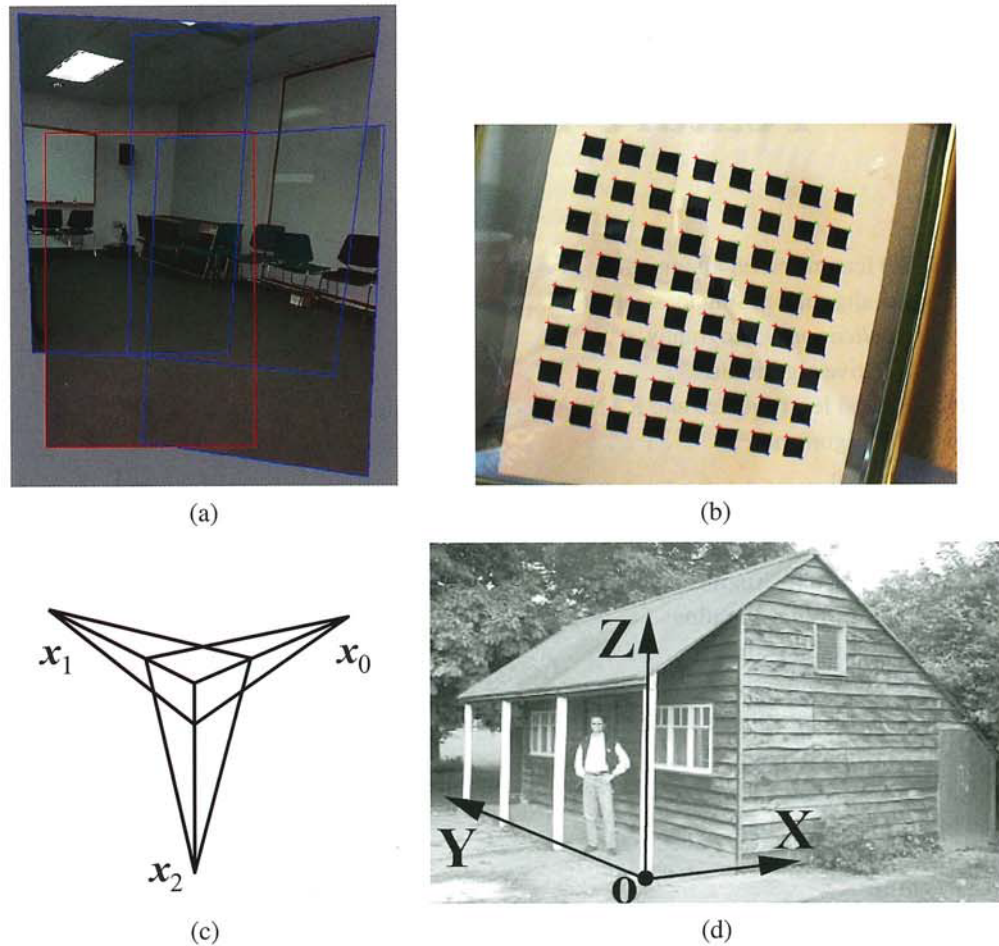


Figure 6.1 Geometric alignment and calibration: (a) geometric alignment of 2D images for stitching (Szeliski and Shum 1997) © 1997 ACM; (b) a two-dimensional calibration target (Zhang 2000) © 2000 IEEE; (c) calibration from vanishing points; (d) scene with easy-to-find lines and vanishing directions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

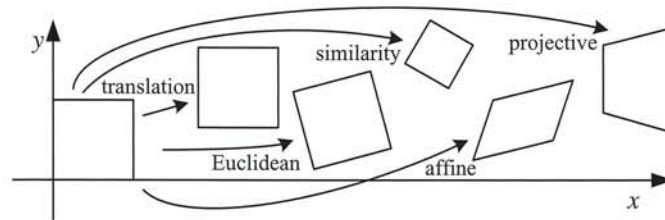


Figure 6.2 Basic set of 2D planar transformations

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 $\{(x_i, x'_i)\}$ and a planar parametric transformation¹ of the form

$$x' = f(x; p), \quad (6.1)$$

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 | $\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$ | $(h_{00}, h_{01}, \dots, h_{21})$ | (see Section 6.1.3) |

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

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

$$E_{LS} = \sum_i \|r_i\|^2 = \sum_i \|f(x_i; p) - x'_i\|^2, \quad (6.2)$$

where

$$r_i = f(x_i; p) - x'_i = \hat{x}'_i - \tilde{x}'_i \quad (6.3)$$

is the *residual* between the measured location \hat{x}'_i and its corresponding current *predicted* location $\tilde{x}'_i = f(x_i; p)$. (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 $\Delta x = x' - x$ and the unknown parameters p ,

$$\Delta x = x' - x = J(x)p, \quad (6.4)$$

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

$$E_{LLS} = \sum_i \|J(x_i)p - \Delta x_i\|^2 \quad (6.5)$$

$$= p^T \left[\sum_i J^T(x_i)J(x_i) \right] p - 2p^T \left[\sum_i J^T(x_i)\Delta x_i \right] + \sum_i \|\Delta x_i\|^2 \quad (6.6)$$

$$= p^T A p - 2p^T b + c. \quad (6.7)$$

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

$$A\mathbf{p} = \mathbf{b}, \quad (6.8)$$

where

$$A = \sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i) \quad (6.9)$$

is called the *Hessian* and $\mathbf{b} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i$. 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

$$E_{\text{WLS}} = \sum_i \sigma_i^{-2} \|\mathbf{r}_i\|^2. \quad (6.10)$$

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* A_i (8.55) with the per-pixel noise covariance σ_n^2 (8.44). Weighting each squared residual by its inverse covariance $\Sigma_i^{-1} = \sigma_n^{-2} A_i$ (which is called the *information matrix*), we obtain

$$E_{\text{CWLS}} = \sum_i \|\mathbf{r}_i\|_{\Sigma_i^{-1}}^2 = \sum_i \mathbf{r}_i^T \Sigma_i^{-1} \mathbf{r}_i = \sum_i \sigma_n^{-2} \mathbf{r}_i^T A_i \mathbf{r}_i. \quad (6.11)$$

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 \mathbf{t}_j be the location of the j th image coordinate frame in the global composite frame and \mathbf{x}_{ij} be the location of the i th matched feature in the j th image. In order to align the images, we wish to minimize the least squares error

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

² For poorly conditioned problems, it is better to use QR decomposition on the set of linear equations $\mathbf{J}(\mathbf{x}_i)\mathbf{p} = \Delta\mathbf{x}_i$ 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 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 t_j and 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 $\mathbf{p} = 0$, which makes it easier to initialize the motion parameters.

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

$$E_{\text{NLS}}(\Delta\mathbf{p}) = \sum_i \|f(\mathbf{x}_i; \mathbf{p} + \Delta\mathbf{p}) - \mathbf{x}'_i\|^2 \quad (6.13)$$

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

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

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

where the “Hessian”⁵ \mathbf{A} is the same as Equation (6.9) and the right hand side vector

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

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

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

and update the parameter vector $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$ 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

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

The Jacobian is therefore

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

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 $\{h_{00}, h_{01}, \dots, h_{21}\}$ 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” \mathbf{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 $\mathbf{f}(\mathbf{x}_i; \mathbf{p} + \Delta \mathbf{p})$.

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 ,

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

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 \mathbf{p} , as shown in (6.14), which yields the set of equations

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

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 \mathbf{p} . Second, the quantities inside \mathbf{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 $\mathbf{p} = 0$) and the above formula simplifies to

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

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

$$E_{\text{RLS}}(\Delta \mathbf{p}) = \sum_i \rho(\|\mathbf{r}_i\|) \quad (6.25)$$

instead of squaring them.

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

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

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

$$E_{\text{IRLS}} = \sum_i w(\|\mathbf{r}_i\|) \|\mathbf{r}_i\|^2, \quad (6.27)$$

where the $w(\|\mathbf{r}_i\|)$ play the same local weighting role as σ_i^{-2} in (6.10). The IRLS algorithm alternates between computing the influence functions $w(\|\mathbf{r}_i\|)$ 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 \mathbf{p} . The *residuals* of the full set of correspondences are then computed as

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

where $\tilde{\mathbf{x}}'_i$ are the *estimated* (mapped) locations and $\hat{\mathbf{x}}'_i$ 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 $\|\mathbf{r}_i\| \leq \epsilon$. (The ϵ value is application dependent but is often around 1–3 pixels.) Least median of squares finds the median value of the $\|\mathbf{r}_i\|^2$ 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 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

$$1 - P = (1 - p^k)^S \quad (6.29)$$

and the required minimum number of trials is

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

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,

$$E_{R3D} = \sum_i \|\mathbf{x}'_i - \mathbf{R}\mathbf{x}_i - \mathbf{t}\|^2, \quad (6.31)$$

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 \mathbf{c} and \mathbf{c}' can be used to estimate the translation $\mathbf{t} = \mathbf{c}' - \mathbf{R}\mathbf{c}$.⁸ We are then left with the problem of estimating the rotation between two sets of points $\{\hat{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{c}\}$ and $\{\hat{\mathbf{x}}'_i = \mathbf{x}'_i - \mathbf{c}'\}$ 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

$$\mathbf{C} = \sum_i \hat{\mathbf{x}}'_i \hat{\mathbf{x}}_i^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \quad (6.32)$$

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

Another technique is the absolute orientation algorithm (Horn 1987) for estimating the unit quaternion corresponding to the rotation matrix \mathbf{R} , which involves forming a 4×4 matrix from the entries in \mathbf{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),

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

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

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

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \quad (6.35)$$

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

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

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

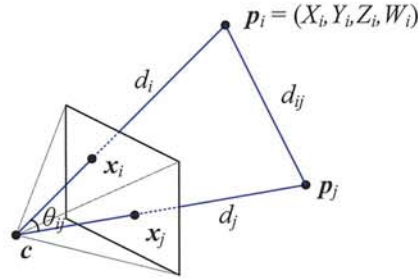


Figure 6.4 Pose estimation by the direct linear transform and by measuring visual angles and distances between pairs of points.

In most applications, however, we have some prior knowledge about the intrinsic calibration matrix \mathbf{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 \mathbf{K} and (\mathbf{R}, t) , as described in Section 6.2.2.

In the case where the camera is already calibrated, i.e., the matrix \mathbf{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 $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ must be the same as the angle between their corresponding 3D points \mathbf{p}_i and \mathbf{p}_j (Figure 6.4).

Given a set of corresponding 2D and 3D points $\{(\hat{\mathbf{x}}_i, \mathbf{p}_i)\}$, where the $\hat{\mathbf{x}}_i$ are unit directions obtained by transforming 2D pixel measurements \mathbf{x}_i to unit norm 3D directions $\hat{\mathbf{x}}_i$ through the inverse calibration matrix \mathbf{K} ,

$$\hat{\mathbf{x}}_i = \mathcal{N}(\mathbf{K}^{-1}\mathbf{x}_i) = \mathbf{K}^{-1}\mathbf{x}_i / \|\mathbf{K}^{-1}\mathbf{x}_i\|, \quad (6.36)$$

the unknowns are the distances d_i from the camera origin c to the 3D points \mathbf{p}_i , where

$$\mathbf{p}_i = d_i \hat{\mathbf{x}}_i + c \quad (6.37)$$

(Figure 6.4). The cosine law for triangle $\Delta(c, \mathbf{p}_i, \mathbf{p}_j)$ gives us

$$f_{ij}(d_i, d_j) = d_i^2 + d_j^2 - 2d_i d_j c_{ij} - d_{ij}^2 = 0, \quad (6.38)$$

where

$$c_{ij} = \cos \theta_{ij} = \hat{\mathbf{x}}_i \cdot \hat{\mathbf{x}}_j \quad (6.39)$$

and

$$d_{ij}^2 = \|\mathbf{p}_i - \mathbf{p}_j\|^2. \quad (6.40)$$

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 ,

$$g_{ijk}(d_i^2) = a_4 d_i^8 + a_3 d_i^6 + a_2 d_i^4 + a_1 d_i^2 + a_0 = 0. \quad (6.41)$$

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 be 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 $d_i \hat{x}_i$, which can then be aligned with the 3D point cloud $\{p_i\}$ using absolute orientation (Section 6.1.5) to obtain 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 (R, t) and optionally K using non-linear least squares (Tsai 1987; Bogart 1991; Gleicher and Witkin 1992). We can write the projection equations as

$$x_i = f(p_i; R, t, K) \quad (6.42)$$

and iteratively minimize the robustified linearized reprojection errors

$$E_{\text{NLP}} = \sum_i \rho \left(\frac{\partial f}{\partial R} \Delta R + \frac{\partial f}{\partial t} \Delta t + \frac{\partial f}{\partial K} \Delta K - r_i \right), \quad (6.43)$$

where $r_i = \tilde{x}_i - \hat{x}_i$ 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 p_i by a simple transformation such as translation, rotation, or perspective division (Figure 6.5). The resulting projection equations can be written as

$$y^{(1)} = f_T(p_i; c_j) = p_i - c_j, \quad (6.44)$$

$$y^{(2)} = f_R(y^{(1)}; q_j) = R(q_j) y^{(1)}, \quad (6.45)$$

$$y^{(3)} = f_P(y^{(2)}) = \frac{y^{(2)}}{z^{(2)}}, \quad (6.46)$$

$$x_i = f_C(y^{(3)}; k) = K(k) y^{(3)}. \quad (6.47)$$

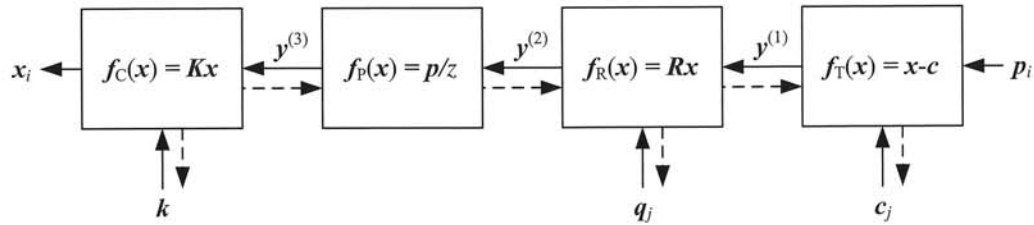


Figure 6.5 A set of chained transforms for projecting a 3D point p_i to a 2D measurement x_i through a series of transformations $f^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

Note that in these equations, we have indexed the camera centers c_j and camera rotation quaternions 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 c_j instead of the world translation 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 \tilde{x}_i has been computed based on the 3D point location p_i and the current values of the pose parameters (c_j, q_j, k) , we can obtain all of the required partial derivatives using the chain rule

$$\frac{\partial r_i}{\partial p^{(k)}} = \frac{\partial r_i}{\partial y^{(k)}} \frac{\partial y^{(k)}}{\partial p^{(k)}}, \quad (6.48)$$

where $p^{(k)}$ 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 $R(q)$ as a function of the unit quaternion entries, you can prepend the incremental rotation matrix $\Delta R(\omega)$ 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

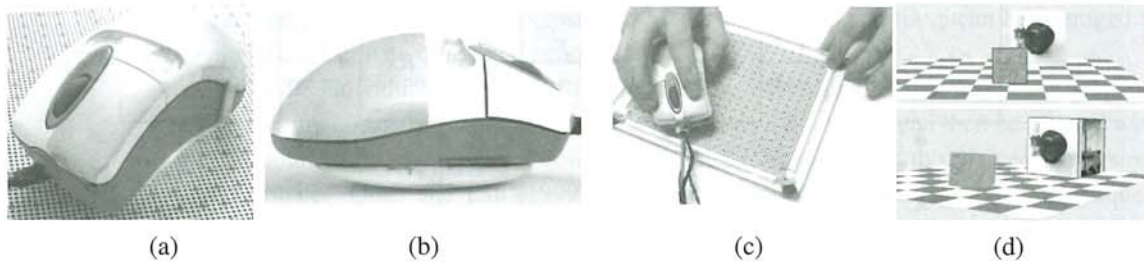


Figure 6.6 The VideoMouse can sense six degrees of freedom relative to a specially printed mouse pad using its embedded camera (Hinckley, Sinclair, Hanson *et al.* 1999) © 1999 ACM: (a) top view of the mouse; (b) view of the mouse showing the curved base for rocking; (c) moving the mouse pad with the 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.

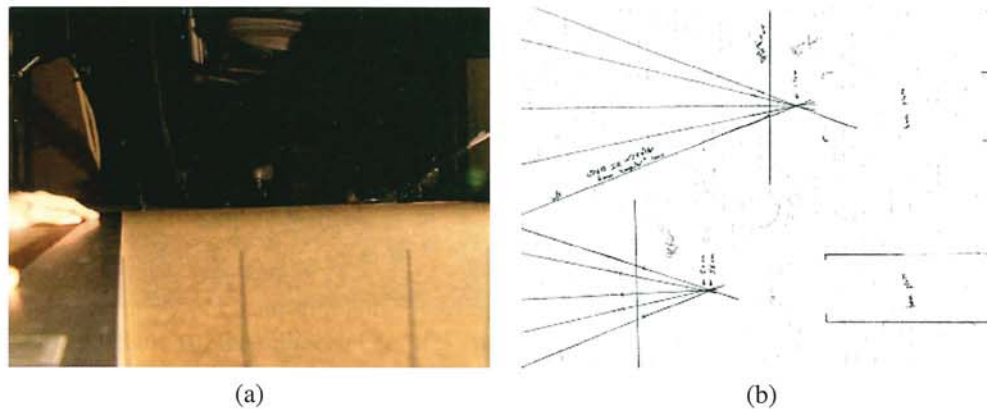


Figure 6.7 Calibrating a lens by drawing straight lines on cardboard (Debevec, Wenger, Tchou *et al.* 2002) © 2002 ACM: (a) an image taken by the video camera showing a hand holding a metal ruler whose right edge appears vertical in the image; (b) the set of lines drawn on the cardboard converging on 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.

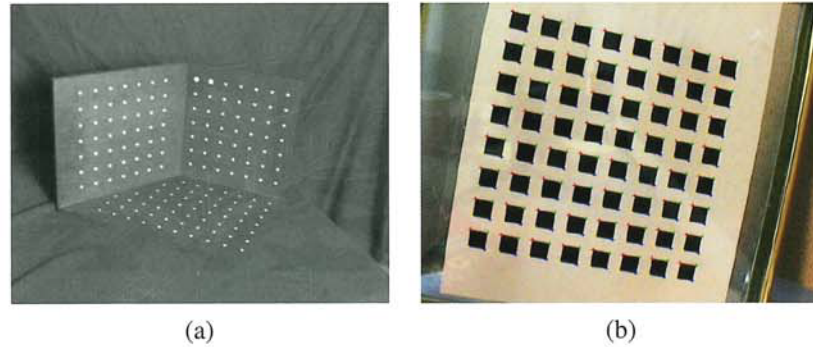


Figure 6.8 Calibration patterns: (a) a three-dimensional target (Quan and Lan 1999) © 1999 IEEE; (b) a two-dimensional target (Zhang 2000) © 2000 IEEE. Note that radial distortion needs to be removed from such images before the feature points can be used for calibration.

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 \mathbf{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) $\tilde{\mathbf{H}}$ mapping the plane's calibration points $(X_i, Y_i, 0)$ into image coordinates (x_i, y_i) ,

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \mathbf{K} \begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\mathbf{H}} \mathbf{p}_i, \quad (6.49)$$

where the \mathbf{r}_i are the first two columns of \mathbf{R} and \sim indicates equality up to scale. From these, Zhang (2000) shows how to form linear constraints on the nine entries in the $\mathbf{B} = \mathbf{K}^{-T} \mathbf{K}^{-1}$ matrix, from which the calibration matrix \mathbf{K} can be recovered using a matrix square root and inversion. (The matrix \mathbf{B} is known as the *image of the absolute conic* (IAC) in projective geometry and is commonly used for camera calibration (Hartley and Zisserman 2004, Section 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 rectangular objects such as boxes or room walls. In this case, we can intersect the 2D lines corresponding to 3D parallel lines to compute their

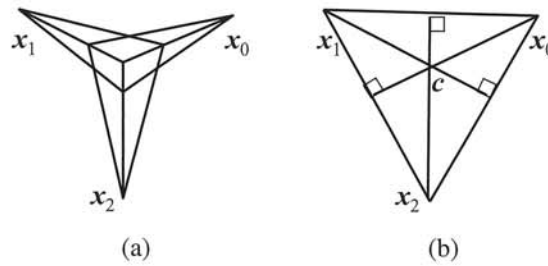


Figure 6.9 Calibration from vanishing points: (a) any pair of finite vanishing points (\hat{x}_i, \hat{x}_j) can be used to estimate the focal length; (b) the orthocenter of the vanishing point triangle gives the optical center of the image c .

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 \mathbf{K} where only the focal length is unknown (2.59). (It is often safe for rough 3D modeling to assume that the optical center is at the center of the image, that the aspect ratio is 1, and that there is no skew.) In this case, the projection equation for the vanishing points can be written as

$$\hat{\mathbf{x}}_i = \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ f \end{bmatrix} \sim \mathbf{R}\mathbf{p}_i = \mathbf{r}_i, \quad (6.50)$$

where \mathbf{p}_i corresponds to one of the cardinal directions $(1, 0, 0)$, $(0, 1, 0)$, or $(0, 0, 1)$, and \mathbf{r}_i is the i th column of the rotation matrix \mathbf{R} .

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

$$\mathbf{r}_i \cdot \mathbf{r}_j \sim (x_i - c_x)(x_j - c_x) + (y_i - c_y)(y_j - c_y) + f^2 = 0 \quad (6.51)$$

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 \mathbf{R} can be estimated by normalizing the left hand side of (6.50) and taking cross products. Alternatively, an SVD of the initial \mathbf{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).

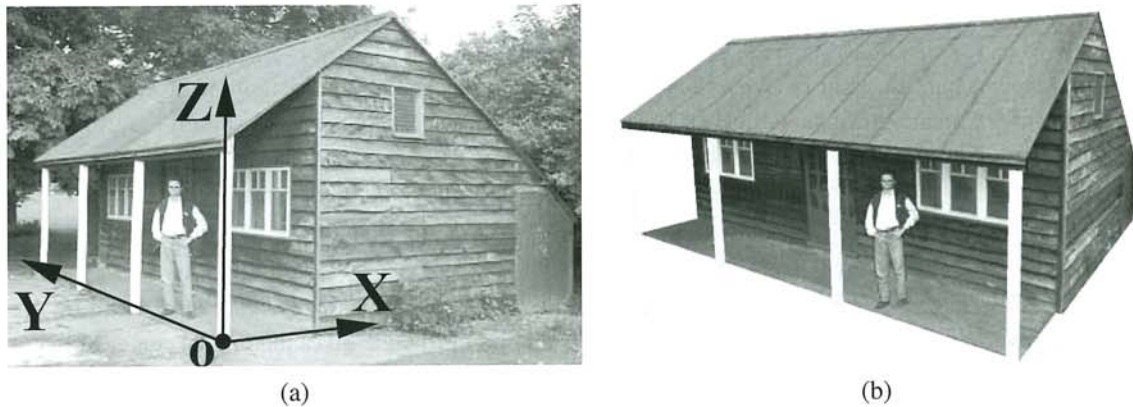


Figure 6.10 Single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer: (a) input image showing the three coordinate axes computed from the two horizontal vanishing points (which can be determined from the sidings on the shed); (b) a new view of the 3D reconstruction.

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).

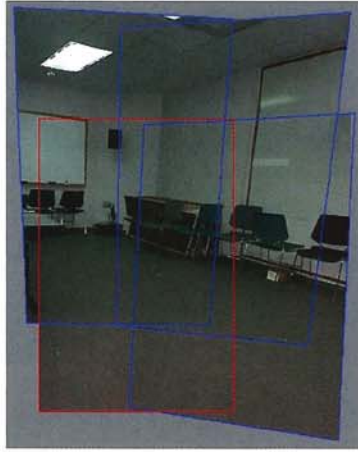


Figure 6.11 Four images taken with a hand-held camera registered using a 3D rotation motion model, which can be used to estimate the focal length of the camera (Szeliski and Shum 1997) © 2000 ACM.

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{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 R_{ij} through the (unknown) calibration matrices K_i and K_j ,

$$\tilde{H}_{ij} = K_i R_i R_j^{-1} K_j^{-1} = K_i R_{ij} K_j^{-1}. \quad (6.52)$$

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., $K_k = \text{diag}(f_k, f_k, 1)$. (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

$$R_{10} \sim K_1^{-1} \tilde{H}_{10} K_0 \sim \begin{bmatrix} h_{00} & h_{01} & f_0^{-1} h_{02} \\ h_{10} & h_{11} & f_0^{-1} h_{12} \\ f_1 h_{20} & f_1 h_{21} & f_0^{-1} f_1 h_{22} \end{bmatrix}, \quad (6.53)$$

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

Using the orthonormality properties of the rotation matrix \mathbf{R}_{10} and the fact that the right hand side of (6.53) is known only up to a scale, we obtain

$$h_{00}^2 + h_{01}^2 + f_0^{-2}h_{02}^2 = h_{10}^2 + h_{11}^2 + f_0^{-2}h_{12}^2 \quad (6.54)$$

and

$$h_{00}h_{10} + h_{01}h_{11} + f_0^{-2}h_{02}h_{12} = 0. \quad (6.55)$$

From this, we can compute estimates for f_0 of

$$f_0^2 = \frac{h_{12}^2 - h_{02}^2}{h_{00}^2 + h_{01}^2 - h_{10}^2 - h_{11}^2} \quad \text{if } h_{00}^2 + h_{01}^2 \neq h_{10}^2 + h_{11}^2 \quad (6.56)$$

or

$$f_0^2 = -\frac{h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \quad \text{if } h_{00}h_{10} \neq -h_{01}h_{11}. \quad (6.57)$$

(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 $\tilde{\mathbf{H}}_{10}$. If the focal length is the same for both images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, e.g., from different homographies, the median value can be used as the final estimate.

A more general (upper-triangular) estimate of \mathbf{K} can be obtained in the case of a fixed-parameter camera $\mathbf{K}_i = \mathbf{K}$ using the technique of Hartley (1997b). Observe from (6.52) that $\mathbf{R}_{ij} \sim \mathbf{K}^{-1} \tilde{\mathbf{H}}_{ij} \mathbf{K}$ and $\mathbf{R}_{ij}^{-T} \sim \mathbf{K}^T \tilde{\mathbf{H}}_{ij}^{-T} \mathbf{K}^{-T}$. Equating $\mathbf{R}_{ij} = \mathbf{R}_{ij}^{-T}$ we obtain $\mathbf{K}^{-1} \tilde{\mathbf{H}}_{ij} \mathbf{K} \sim \mathbf{K}^T \tilde{\mathbf{H}}_{ij}^{-T} \mathbf{K}^{-T}$, from which we get

$$\tilde{\mathbf{H}}_{ij}(\mathbf{K}\mathbf{K}^T) \sim (\mathbf{K}\mathbf{K}^T)\tilde{\mathbf{H}}_{ij}^{-T}. \quad (6.58)$$

This provides us with some homogeneous linear constraints on the entries in $\mathbf{A} = \mathbf{K}\mathbf{K}^T$, 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 $\tilde{\mathbf{H}}_{ij}$, we can recover \mathbf{A} (up to a scale) using either SVD or eigenvalue analysis and then recover \mathbf{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 f_C and the perspective division box f_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.

Humans perceive the three-dimensional structure of the world with apparent ease. However, despite all of the recent advances in computer vision research, the dream of having a computer interpret an image at the same level as a two-year old remains elusive. Why is computer vision such a challenging problem and what is the current state of the art?

Computer Vision: Algorithms and Applications explores the variety of techniques commonly used to analyze and interpret images. It also describes challenging real-world applications where vision is being successfully used, both for specialized applications such as medical imaging, and for fun, consumer-level tasks such as image editing and stitching, which students can apply to their own personal photos and videos.

More than just a source of “recipes,” this exceptionally authoritative and comprehensive textbook/reference also takes a scientific approach to basic vision problems, formulating physical models of the imaging process before inverting them to produce descriptions of a scene. These problems are also analyzed using statistical models and solved using rigorous engineering techniques.

Topics and Features:

- Structured to support active curricula and project-oriented courses, with tips in the Introduction for using the book in a variety of customized courses
- Presents exercises at the end of each chapter with a heavy emphasis on testing algorithms and containing numerous suggestions for small mid-term projects
- Provides additional material and more detailed mathematical topics in the Appendices, which cover linear algebra, numerical techniques, and Bayesian estimation theory
- Suggests additional reading at the end of each chapter, including the latest research in each sub-field, in addition to a full Bibliography at the end of the book
- Supplies supplementary course material for students at the associated website, <http://szeliski.org/Book/>

Suitable for an upper-level undergraduate or graduate-level course in computer science or engineering, this textbook focuses on basic techniques that work under real-world conditions and encourages students to push their creative boundaries. Its design and exposition also make it eminently suitable as a unique reference to the fundamental techniques and current research literature in computer vision.

Dr. Richard Szeliski has more than 25 years' experience in computer vision research, most notably at Digital Equipment Corporation and Microsoft Research. This text draws on that experience, as well as on computer vision courses he has taught at the University of Washington and Stanford.

ISBN 978-1-84882-934-3



9 781848 829343