TEXTS IN COMPUTER SCIENCE

Computer Vision

Algorithms and Applications

Richard Szeliski

Springer

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

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

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

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

Library of Congress Control Number: 2010936817

Printed on acid-free paper

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

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

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

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

2.1.5 3D to 2D projections

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

Orthography and para-perspective

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

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

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

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

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

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

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

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

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

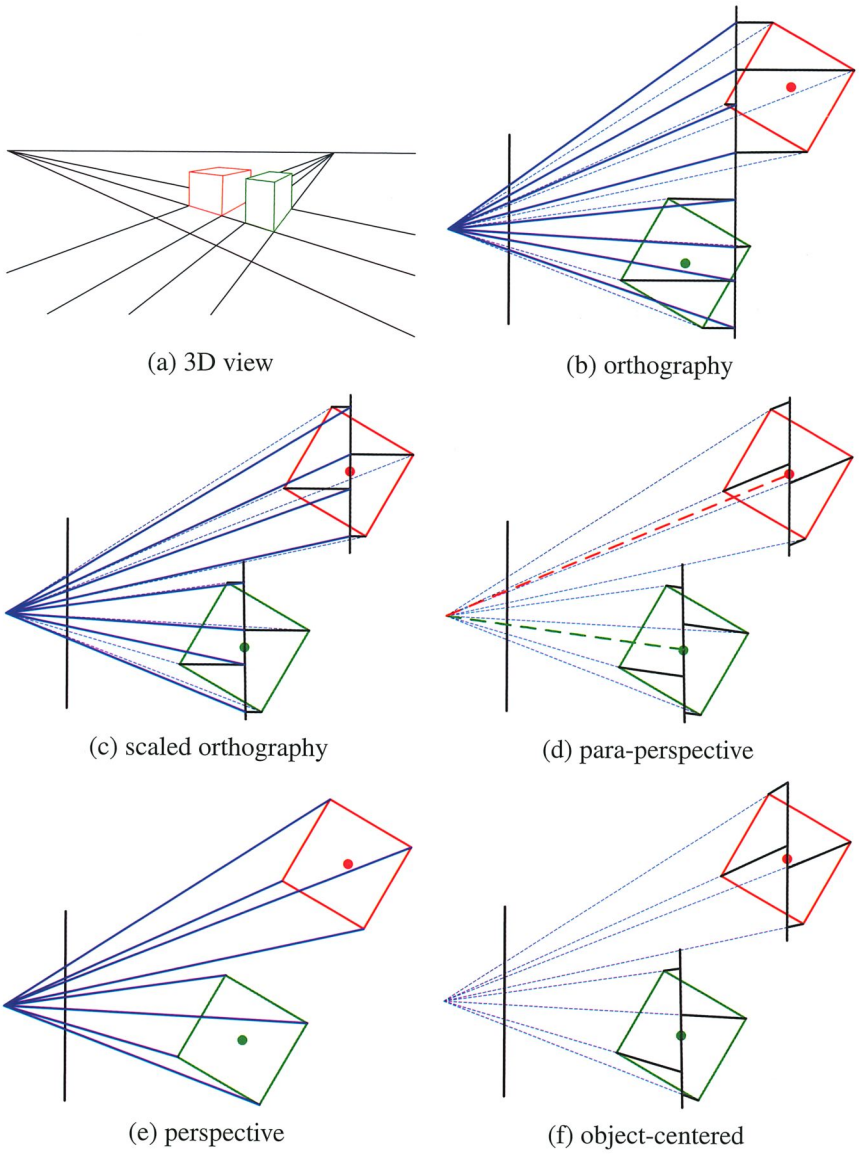

Figure 2.7 Commonly used projection models: (a) 3D view of world, (b) orthography, (c) scaled orthography, (d) para-perspective, (e) perspective, (f) object-centered. Each diagram shows a top-down view of the projection. Note how parallel lines on the ground plane and box sides remain parallel in the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

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

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

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

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

Perspective

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

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

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

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

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

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

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

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

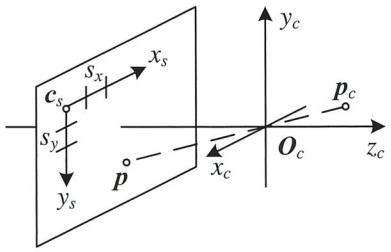

Figure 2.8 Projection of a 3D camera-centered point $\boldsymbol{p}_c$ onto the sensor planes at location $\boldsymbol{p}$. $\boldsymbol{O}_c$ is the camera center (nodal point), $\boldsymbol{c}_s$ is the 3D origin of the sensor plane coordinate system, and s_x and s_y are the pixel spacings.

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

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

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

Camera intrinsics

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

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

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

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

The first two columns of the 3×3 matrix $\boldsymbol{M}_s$ 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 $\boldsymbol{c}_s$.

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

$$\tilde{\boldsymbol{x}}_s = \alpha \boldsymbol{M}_s^{-1} \boldsymbol{p}_c = \boldsymbol{K} \boldsymbol{p}_c. \tag{2.54}$$

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

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

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

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

is known as the *camera matrix*. Inspecting this equation, we see that we can post-multiply $\boldsymbol{K}$ by $\boldsymbol{R}_1$ and pre-multiply $[\boldsymbol{R}|\boldsymbol{t}]$ by $\boldsymbol{R}_1^T$, and still end up with a valid calibration. Thus, it is impossible based on image measurements alone to know the true orientation of the sensor and the true camera intrinsics.

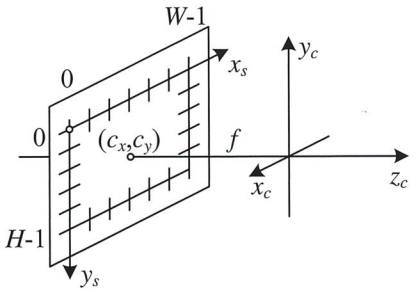

Figure 2.9 Simplified camera intrinsics showing the focal length f and the optical center (c_x, c_y). The image width and height are W and H.

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

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

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

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

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

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

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

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

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

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

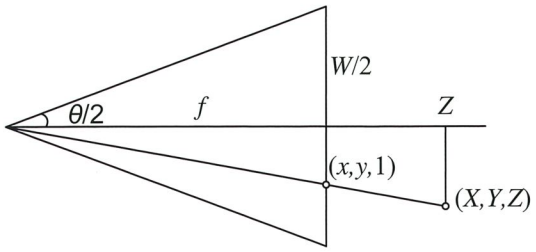

Figure 2.10 Central projection, showing the relationship between the 3D and 2D coordinates, $\boldsymbol{p}$ and $\boldsymbol{x}$, as well as the relationship between the focal length f, image width W, and the field of view θ.

A note on focal lengths

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

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

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

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

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

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

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

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

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

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

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

Camera matrix

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

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

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

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

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

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

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

Plane plus parallax (projective depth)

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

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

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

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

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

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

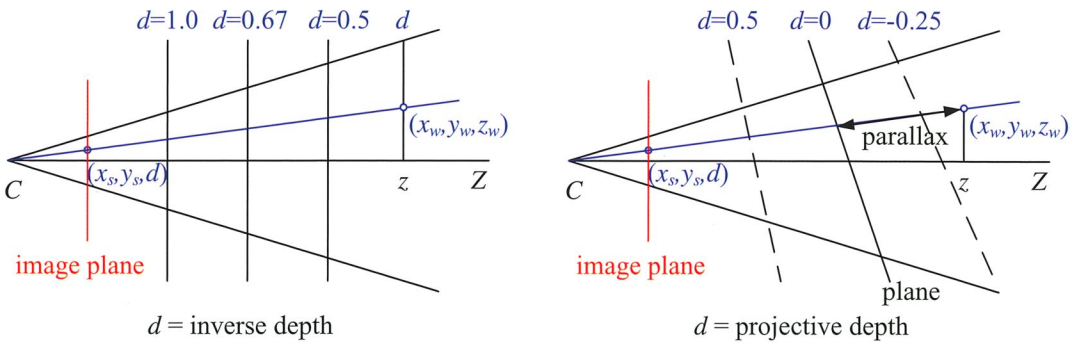

Figure 2.11 Regular disparity (inverse depth) and projective depth (parallax from a reference plane).

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

Mapping from one camera to another

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

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

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

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

and then project it into another image yielding

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

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

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

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

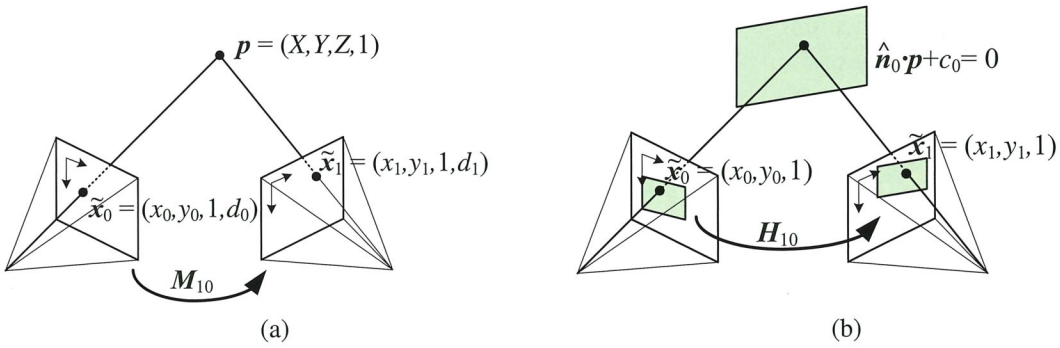

Figure 2.12 A point is projected into two images: (a) relationship between the 3D point coordinate $(X, Y, Z, 1)$ and the 2D projected point $(x, y, 1, d)$; (b) planar homography induced by points all lying on a common plane $\hat{\boldsymbol{n}}_0 \cdot \boldsymbol{p} + c_0 = 0$.

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

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

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

Object-centered projection

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

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

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

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

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

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

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

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

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

2.1.6 Lens distortions

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

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

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

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

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

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

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

springer.com