10 Camera geometry fundamentals

10.1 Overview

This chapter is concerned with the geometry that describes how objects in the 3D world are captured as 2D pixels. These descriptions are mathematical models that define a mapping between 3D objects (generally points) in the world and 2D objects in a plane that represents an image. The final aim is to be able to obtain 3D information from the 2D position of pixels. The techniques of 3D analysis such as reconstruction or photogrammetry are studied in computer vision [Trucco98, Hartley01]. This chapter does not cover computer vision techniques, but gives an introduction to the fundamental concepts of the geometry of computer vision. It aims to complement the concepts in Chapter 1 by increasing the background knowledge of how camera geometry is mathematically modelled. This chapter explains basic aspects of common camera geometry models.

Although the mapping between 3D points and pixels can be defined using functions in Euclidean co-ordinates, the notation and development is better expressed by using matrix operations in the projective space. As such, we start this chapter by introducing the space defined by *homogenous co-ordinates* and by considering how transformations can be defined using a matrix notation. We will see that there are different models that can be used to represent alternative image capture technologies or that approximate the image capture by using different complexity. A model defines a general type of image formation process, whilst particular models for specific cameras are defined by values given to the camera parameters. Different models can be used according to the capture technology and according to the type of objects in the scene. For example, an affine model can be a good approximation of a projective model when objects are far away or placed in a plane with similar alignment than the camera plane (i.e. images with little or no perspective). The chapter is summarised in Table 10.1.

10.2 Projective space

You should be familiar with the concept of the Euclidean space as the conventional way to represent points, lines, panes and other geometric objects. Euclidean geometry is algebraically represented by the *Cartesian co-ordinate system* in which points are defined by tuples of numbers. Each number is related to one axis, and a set of axes determines the space's dimensions. This representation is a natural way to describe 3D objects in the world, and it is very useful in image processing to describe pixels in 2D images. Cartesian

Feature Extraction and Image Processing for Computer Vision. https://doi.org/10.1016/B978-0-12-814976-8.00010-5

Table 10.1 Overview of this chapter.

Main topic	Subtopics	Main points
Projective space	What is the projective space and what are homogenous co-ordinates.	Transforming *Euclidean co-ordinates* to *homogeneous co-ordinates*. *Homogenous matrix* transformations. *Similarity* and *affine* transformations. *Homography*. How to obtain a homography.
Perspective camera	A general description of how 3D points are mapped into an image plane. How we define a model for the perspective projection.	Image *projection*, *principal point*, *focus length* and *camera parameters*. Parameters of the *perspective camera* model. *Projections*.
Affine camera	Affine as a simplification of the perspective camera.	*Affine projection*. Affine *camera model*, *linear* camera. Affine camera *parameters*.
Weak perspective	Weak perspective camera model.	Relationship between *weak perspective* and *perspective models*. *Weak perspective camera parameters*.

co-ordinates are convenient to describe angles and lengths, and they are simply transformed by matrix algebra to represent translations, rotations and changes of scale. However, the relationship defined by projections cannot be described with the same algebraic simplicity (it can be done but it is cumbersome). For example, it is difficult to express that parallel lines in the 3D world actually have an intersection point when considered in a projection or it is difficult to write that the projection of a point is defined as a line. The best way to express the relationship defined by projections is to use projective geometry. Projective geometry replaces the Cartesian system with homogenous co-ordinates. In a similar way to how Cartesian co-ordinates define the Euclidean space, homogenous co-ordinates define the projective space. This section explains the fundamental concepts of projective geometry.

10.2.1 Homogeneous co-ordinates and projective geometry

Projective geometry is algebraically represented by the homogeneous co-ordinate system. This representation is a natural way to formulate how we relate camera co-ordinates to *real-world* co-ordinates: the relation between image and physical space. Its major advantages are that image transformations like rotations, change of scale and projections become matrix multiplications. Projections provide perspective that corresponds to the distance of objects that affects their size in the image.

It is possible to map points from Cartesian co-ordinates into homogeneous co-ordinates. The 2D point with Cartesian co-ordinates

$$\mathbf{x}_c = [\, x \quad y \,]^{\mathrm{T}} \tag{10.1}$$

is mapped into homogeneous co-ordinates to the point

$$\mathbf{x}_h = [\, wx \quad wy \quad w \,]^{\mathrm{T}} \tag{10.2}$$

where w is an arbitrary scalar. Note that a point in Cartesian co-ordinates is mapped into several points in homogeneous co-ordinates; one point for any value of w. This is why homogeneous co-ordinates are also called redundant co-ordinates. We can use the definition on Eq. (10.2) to obtain a mapping from homogeneous co-ordinates to Cartesian co-ordinates. That is,

$$x_c = \frac{w_x}{w} \text{ and } y_c = \frac{w_y}{w} \tag{10.3}$$

The homogeneous representation can be extended to any dimension. For example, a 3D point in Cartesian co-ordinates

$$\mathbf{x}_c = [x \quad y \quad z]^{\mathrm{T}} \tag{10.4}$$

is mapped into homogeneous form as

$$\mathbf{x}_h = [wx \quad wy \quad wz \quad w]^{\mathrm{T}} \tag{10.5}$$

This point is mapped back to Cartesian co-ordinates by

$$x_c = \frac{w_x}{w}, y_c = \frac{w_y}{w} \text{ and } z_c = \frac{w_z}{w} \tag{10.6}$$

Although it is possible to map points from Cartesian co-ordinates to homogeneous co-ordinates and vice versa, points in both systems define different geometric spaces. Cartesian co-ordinates define the Euclidean space and the points in homogeneous co-ordinates define the projective space. The projective space distinguishes a particular class of points defined when the last co-ordinate is zero. These are known as *ideal points* and to understand them, we need to understand how a line is represented in projective space. This is related to the concept of *duality*.

10.2.2 Representation of a line, duality and ideal points

The homogeneous representation of points has a very interesting idea that relates points and lines. Let us consider the equation of a 2D line in Cartesian co-ordinates,

$$Ax + By + C = 0 \tag{10.7}$$

The same equation in homogeneous co-ordinates becomes

$$Ax + By + Cz = 0 \tag{10.8}$$

It is interesting to notice that in this definition, points and lines are indistinguishable. Both a point $[x \quad y \quad z]^{\mathrm{T}}$ and a line $[A \quad B \quad C]^{\mathrm{T}}$ are represented by triplets, and they can be interchanged in the homogeneous equation of a line. This concept can be generalised and points are indistinguishable to planes for the 3D projective space. This symmetry is known as the *duality of the projective space* that can be combined with the concept of concurrence and incidence to derive the principle of duality [Aguado00]. The principle of duality constitutes an important concept for understanding the geometric relationship in the projective space; the definition of the line can be used to derive the concept of ideal points.

We can use the algebra of homogeneous co-ordinates to find the intersection of parallel lines, planes and hyper-planes. For simplicity, let us consider lines in the 2D plane. In the Cartesian co-ordinates in Eq. (10.7), two lines are parallel when their slopes $y' = -A/B$ are the same. Thus, in order to find the intersection between two parallel lines in the homogeneous form in Eq. (10.8), we need to solve the following system of equations

$$\begin{aligned} A_1 x + B_1 y + C_1 z &= 0 \\ A_2 x + B_2 y + C_2 z &= 0 \end{aligned} \tag{10.9}$$

for $A_1/B_1 = A_2/B_2$. By dividing the first equation by B_1, the second equation by B_2 and by subtracting the second equation to the first, we have that

$$(C_2 - C_1)z = 0 \tag{10.10}$$

Since we are considering different lines, then $C_2 \neq C_1$ and consequently $z = 0$. That is, the intersection of parallel lines is defined by points of the form

$$\mathbf{x}_h = [x \quad y \quad 0]^{\mathrm{T}} \tag{10.11}$$

Similarly, in 3D, the intersection of parallel planes is defined by the points given by

$$\mathbf{x}_h = [x \quad y \quad z \quad 0]^{\mathrm{T}} \tag{10.12}$$

Since parallel lines are assumed to intersect at infinity, then points with the last co-ordinate equal to zero are called points at infinity. They are also called ideal points, and these points plus all the other homogeneous points form the projective space.

The points in the projective space can be visualised by extending the Euclidean space as shown in Fig. 10.1. This figure illustrates the 2D projective space as a set of points in the 3D Euclidean space. According to Eq. (10.3), points in the homogeneous space are mapped into the Euclidean space when $z = 1$. In the figure, this plane is called the

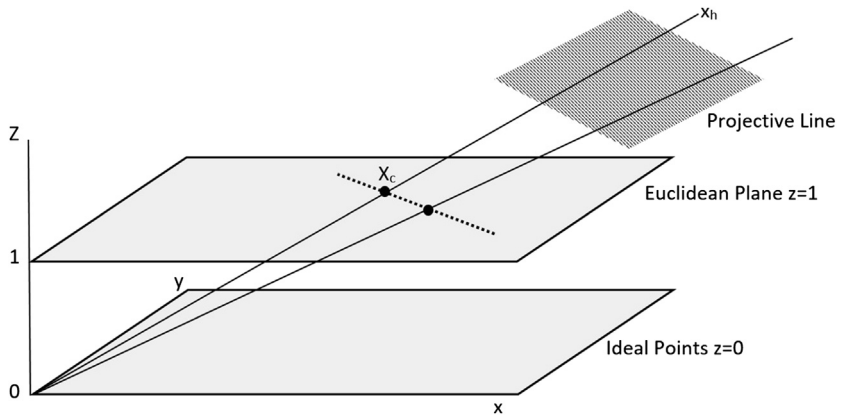


FIGURE 10.1 Model of the 2D projective space (ray space).

Euclidean plane. Fig. 10.1 shows two points in the Euclidean plane. These points define a line that is shown as a dotted line and it extends to infinity in the plane. In homogeneous co-ordinates, points in the Euclidean plane become rays from the origin in the projective space. Each point in the ray is given by a different value of z. The homogeneous co-ordinates of the line in the Euclidean plane define the plane between the two rays in the projective space. When two lines intersect in the Euclidean plane, they define a ray that passes through the intersection point in the Euclidean plane. However, if the lines are parallel, then they define an ideal point. That is a point in the plane $z = 0$.

Note that the origin $[0 \quad 0 \quad 0]^{\mathrm{T}}$ is ambiguous since it can define any point in homogeneous co-ordinates or an ideal point. To avoid this ambiguity, this point is not considered to be part of the projective space. Also remember that the concept of point and line are indistinguishable, so it is possible to draw a dual diagram, where points become lines and vice versa.

10.2.3 Transformations in the projective space

Perhaps the most practical aspect of homogeneous co-ordinates is the way transformations are represented algebraically. This section shows how different types of transformations in Cartesian co-ordinates can be written in a simple form by using homogenous co-ordinates. This is because the Euclidean plane is included in the projective space, so Euclidean transformations are special cases of projective transformations. We start with similarity or rigid transformations. These transformations do not change the angle values in any geometric shape and they define rotations, changes in scale and translations (i.e. position). Similarity transformations are algebraically represented by matrix multiplications and additions. A 2D point $p = (x, y)$ is transformed into a point $p'=(x', y')$ by a similarity transformation as,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} s_x \\ s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{10.13}$$

where θ is a rotation angle, $\mathbf{S} = [s_x, s_y]^T$ defines the scale and $\mathbf{T} = [t_x, t_y]^T$ the translation along each axis. This transformation can be generalised to any dimension and it is written in short form as

$$\boldsymbol{x}' = \mathbf{RS}\boldsymbol{x} + \mathbf{T} \tag{10.14}$$

Notice that $\mathbf{R}$ is an orthogonal matrix. That is, its transpose is equal to its inverse or $\mathbf{R}^T = \mathbf{R}^{-1}$.

There is a more general type of transformations known as *affine transformations* where the matrix $\mathbf{R}$ is replaced by a matrix $\mathbf{A}$ that is not necessarily orthogonal. That is,

$$\boldsymbol{x}' = \mathbf{AS}\boldsymbol{x} + \mathbf{T} \tag{10.15}$$

Affine transformations do not preserve the value of angles, but they preserve parallel lines. The principles and theorems studied under similarities define Euclidean geometry,

and the principles and theorems under affine transformations define the affine geometry.

In the projective space, transformations are called *homographies*. They are more general than similarity and affine transformations; they only preserve collinearities and cross ratios. That is, points forming a straight line are transformed into points forming another straight line and the distance ratio computed from four points is maintained. Image homographies are defined in homogeneous co-ordinates. A 2D point is transformed as,

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{10.16}$$

This transformation can be generalised to other dimensions and it is written in short form as

$$\boldsymbol{x}' = \mathbf{H}\boldsymbol{x} \tag{10.17}$$

A similarity transformation is a special case of an affine transformation, and that an affine transformation is a special case of a homography. That is, rigid and affine transformations can be expressed as homographies. For example, a rigid transformation for a 2D point can be defined as,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cos(\theta) & s_x \sin(\theta) & t_x \\ -s_y \sin(\theta) & s_y \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.18}$$

Or in a more general form as

$$\boldsymbol{x}' = \begin{bmatrix} \mathbf{RS} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \boldsymbol{x} \tag{10.19}$$

An affine transformation is defined as,

$$\boldsymbol{x}' = \begin{bmatrix} \mathbf{A} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \boldsymbol{x} \tag{10.20}$$

The zeros in the last row are actually defining a transformation in a plane; the plane where $z = 1$. According to the discussion in Section 10.2.2, this plane defines the Euclidean plane. Thus these transformations are performed to Euclidean points.

Code 10.1 illustrates the definition of the transformations in Eqs. (10.17), (10.19) and (10.20). This code uses a transformation `T` that is defined according to the transformation type. The main iteration maps the pixels from an input image to an output image by performing the matrix multiplication. Note that the pixel positions are defined such that the centre of the image has the co-ordinates (0,0). Thus, the transformation includes the translations `centreX`, `centreY`. This permits definition of the rotation in the similarity transformation with respect to the centre of the image.

```
if transformationType  == "Similarity":
    # Similarity transformation
    s = [.4, 0.8, 0.8, 100.0, 0.0] # Angle, scaleXY, translationXY
    T = [[ s[1]*cos(s[0]), s[1]*sin(s[0]), s[3]],                          \
         [ -s[2]*sin(s[0]), s[2]*cos(s[0]), s[4]],                         \
         [0 ,0, 1]]
if transformationType  == "Affine":
    # Affine transformation
    T = [[ .8, .1, 100],                                                   \
         [ -.2, 1, 0],                                                     \
         [0 ,0, 1]]
if transformationType  == "Homography":
    # Homography
    T = [[ .8, 0, 100],                                                    \
         [ .2,1, 0],                                                       \
         [.0005 ,-0.0005 , 1.2]]

tImage = createImageRGB(width, height)
for y, x in itertools.product(range(0, height-1), range(0, width-1)):
    # Alpha and colour
    alpha = maskImage[y,x]/256.0
    if alpha == 0:
        continue
    rgb = (inputImage[y,x]/4.0   + inputImage[y+1,x+1]/4.0 +           \
           inputImage[y+1,x]/4.0 + inputImage[y,x+1]/4.0) * alpha

    # Transform
    cx, cy = x - centreX, y - centreY
    p0z = T[2][0] * cx + T[2][1] * cy + T[2][2]
    p1z = T[2][0] * (cx+1) + T[2][1] * cy + T[2][2]
    p2z = T[2][0] * (cx+1) + T[2][1] * (cy+1) + T[2][2]

    if p0z != 0 and p1z != 0 and p2z !=0:
        p0x = int((T[0][0] * cx + T[0][1] * cy + T[0][2]) / p0z + centreX)
        p0y = int((T[1][0] * cx + T[1][1] * cy + T[1][2]) / p0z + centreY)
        p1x = int((T[0][0] * (cx+1) + T[0][1] * cy + T[0][2]) / p1z + centreX)
        p1y = int((T[1][0] * (cx+1) + T[1][1] * cy + T[1][2]) / p1z + centreY)
        p2x = int((T[0][0] * (cx+1) + T[0][1] * (cy+1) + T[0][2]) / p2z + centreX)
        p2y = int((T[1][0] * (cx+1) + T[1][1] * (cy+1) + T[1][2]) / p2z + centreY)

        # Fill output image
        v1,v2 = [p1x - p0x, p1y - p0y], [p2x - p0x, p2y - p0y]

        lv1 = max(.001,sqrt(v1[0]*v1[0] + v1[1]*v1[1]))
        lv2 = max(.001,sqrt(v2[0]*v2[0] + v2[1]*v2[1]))
        v1N = [v1[0]/lv1, v1[1]/lv1]
        v2N = [v2[0]/lv2, v2[1]/lv2]

        for dV1, dV2 in itertools.product(range(0, int(lv1)+1), range(0, int(lv2)+1)):
            a = int(p0x + dV1 * v1N[0] + dV2 * v2N[0])
            b = int(p0y + dV1 * v1N[1] + dV2 * v2N[1])
            if a>0 and a < width and b > 0 and b < height:
                tImage[b,a] = rgb
```

CODE 10.1 Apply a geometric transformation to an image.

The implementation in Code 10.1 includes a simple image *warping* process that transforms squares defined by four pixels in the input image into polygons (rhomboids) in the output image. The rhomboids are filled with the average colour of the pixels in the input. In a better implementation, the colours for the rhomboids can be determined by interpolation. Also a more accurate position of the rhomboids' vertices can be obtained

by considering sub-pixel positions. The implementation includes a mask image, so only pixels in the region defined by the Rubik's cube are shown in the output image.

Fig. 10.2 shows an example of the image transformations obtained with Code 10.1. Notice how the similarity transformation preserves angles, the affine transformation preserves parallel lines and the homography preserves straight lines (collinearity). Each transformation can be used to model different changes on the object and the camera. Similarities can be used to find rigid changes in objects or to align images or for simple operations like change aspect ratio or zoom. Affine transformations can be used to correct geometric distortions. Homographies are mainly used for image alignment, and they are useful to obtain relative cameras' positions.

10.2.4 Computing a planar homography

Although applying a transformation to an image is important in some applications, the central problem in computer vision focuses on finding a transformation from image data. This is a parameter estimation problem, and there are many techniques aimed at dealing with problems like noise and occlusions. This section introduces the estimation concepts by considering how four points in corresponding images can be used to solve for $\mathbf{H}$ in the system of equations defined in Eq. (10.17).

According to Eq. (10.3), a point $\mathbf{p}=(x, y, 1)$ is mapped into $\mathbf{p}'=(x', y', 1)$ by

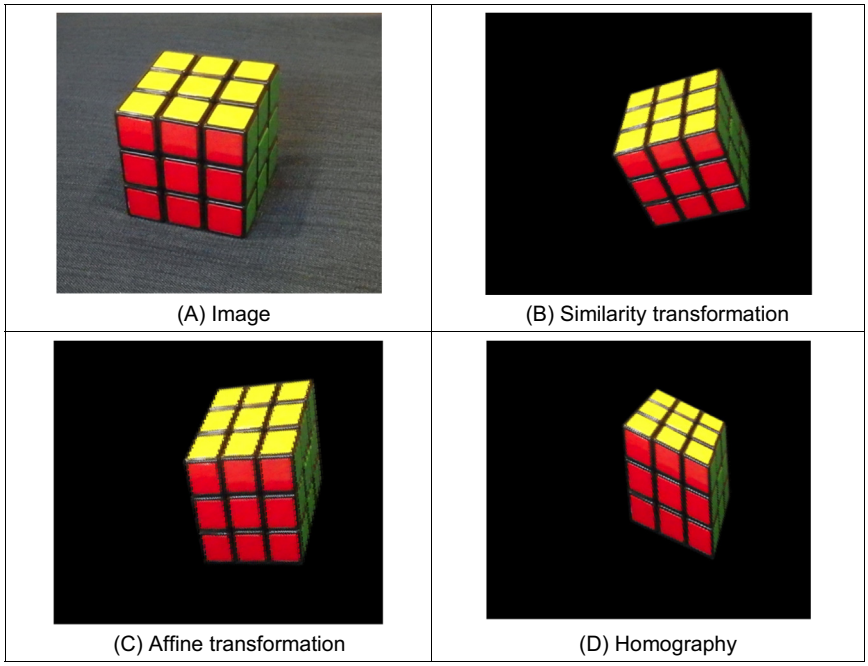


FIGURE 10.2 Image transformations and warping.

$$x' = \frac{xh_{1,1} + yh_{1,2} + zh_{1,3}}{xh_{3,1} + yh_{3,2} + zh_{3,3}} \text{ and } y'_2 = \frac{x_1h_{2,1} + y_1h_{2,2} + z_1h_{2,3}}{x_1h_{3,1} + y_1h_{3,2} + z_1h_{3,3}} \tag{10.21}$$

Since the transform is in the Euclidean plane (i.e. 2D images), then $z = 1$. That is,

$$\begin{aligned} x'(xh_{3,1} + yh_{3,2} + h_{3,3}) &= xh_{1,1} + yh_{1,2} + h_{1,3} \\ y'(xh_{3,1} + yh_{3,2} + h_{3,3}) &= xh_{2,1} + yh_{2,2} + h_{2,3} \end{aligned} \tag{10.22}$$

By rearranging the terms in these equations, we have that

$$\begin{aligned} -xh_{1,1} - yh_{1,2} - h_{1,3} + x'xh_{3,1} + x'yh_{3,2} + x'h_{3,3} = 0 \\ -xh_{2,1} - yh_{2,2} - h_{2,3} + y'xh_{3,1} + y'yh_{3,2} + y'h_{3,3} = 0 \end{aligned} \tag{10.23}$$

This can be written in matrix form as

$$\begin{aligned} [\, -x \quad -y \quad -1 \quad 0 \quad 0 \quad 0 \quad x'x \quad x'y \quad x' \,]\, \mathbf{h}^{\mathbf{T}} = 0 \\ [\, 0 \quad 0 \quad 0 \quad -x \quad -y \quad -1 \quad y'x \quad y'y \quad y' \,]\, \mathbf{h}^{\mathbf{T}} = 0 \end{aligned} \tag{10.24}$$

where $\mathbf{h}$ is the vector containing the coefficients of the homography. That is,

$$\mathbf{h} = [\, h_{1,1} \quad h_{1,2} \quad h_{1,3} \quad h_{2,1} \quad h_{2,2} \quad h_{2,3} \quad h_{3,1} \quad h_{3,2} \quad h_{3,3} \,] \tag{10.25}$$

As such, a pair of corresponding points $\mathbf{p}$=(x, y, 1) and $\mathbf{p}'$=(x', y', 1) define a pair of rows of a linear system of equations of rank nine. In order to make the system determined, we need four corresponding pairs. This defines eight rows in the equation system. To define the last row, we should notice that the system is homogenous. That is, any scalar product of $\mathbf{h}$ is solution of the system. Thus last row of the system can be defined by considering a scale value for the matrix. As such, by considering four corresponding pair of points $\mathbf{p}_i = (x_i, y_i, 1)$ and $\mathbf{p}'_i = (x'_i, y'_i, 1)$, we can form the system of equations

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x'_1x_1 & x'_1y_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y'_1x_1 & y'_1y_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x'_2x_2 & x'_2y_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y'_2x_2 & y'_2y_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x'_3x_3 & x'_3y_3 & x'_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & y'_3x_3 & y'_3y_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x'_4x_4 & x'_4y_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & y'_4x_4 & y'_4y_4 & y'_4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \mathbf{h}^{\mathbf{T}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{10.26}$$

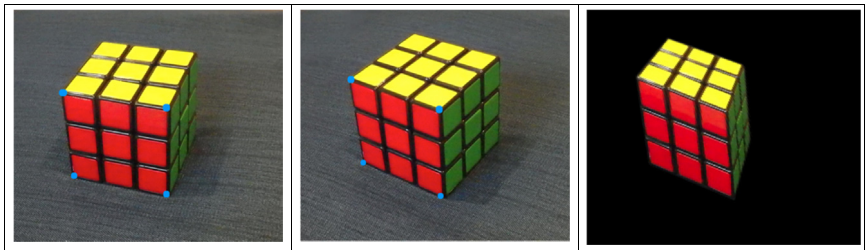
Code 10.2 illustrates the use of this equation to compute a homography. The corresponding points are given in the lists `p` and `q`. The matrices `M` and `b` store the coefficients and constant terms of the system. After the system is solved, the function `imageTransform` is used to obtain an image of the transformation using warping as implemented in Code 10.1 Apply a geometric transformation to an image. Notice that in general we would like to include more points in the solution to increase accuracy and to deal with noise. Thus a better implementation should solve for an overdetermined system.

Fig. 10.3 shows an example of a homography computed with Code 10.2. The homography is computed by considering the corresponding points defined by the four

```
# Corresponding points
p = [[116-centreX,202-centreY],[352-centreX,234-centreY],                    \
     [140-centreX,384-centreY],[344-centreX,422-centreY]]
q = [[118-centreX,168-centreY],[312-centreX,238-centreY],                    \
     [146-centreX,352-centreY],[322-centreX,422-centreY]]

# Find transform
M = [[-p[0][0], -p[0][1], -1, 0, 0, 0,  p[0][0]*q[0][0], p[0][1]*q[0][0], q[0][0]], \
     [ 0, 0, 0, -p[0][0], -p[0][1], -1, p[0][0]*q[0][1], p[0][1]*q[0][1], q[0][1]], \
     [-p[1][0], -p[1][1], -1, 0, 0, 0,  p[1][0]*q[1][0], p[1][1]*q[1][0], q[1][0]], \
     [ 0, 0, 0, -p[1][0], -p[1][1], -1, p[1][0]*q[1][1], p[1][1]*q[1][1], q[1][1]], \
     [-p[2][0], -p[2][1], -1, 0, 0, 0,  p[2][0]*q[2][0], p[2][1]*q[2][0], q[2][0]], \
     [ 0, 0, 0, -p[2][0], -p[2][1], -1, p[2][0]*q[2][1], p[2][1]*q[2][1], q[2][1]], \
     [-p[3][0], -p[3][1], -1, 0, 0, 0,  p[3][0]*q[3][0], p[3][1]*q[3][0], q[3][0]], \
     [ 0, 0, 0, -p[3][0], -p[3][1], -1, p[3][0]*q[3][1], p[3][1]*q[3][1], q[3][1]], \
     [ 1, 1, 1,     1,        1,     1,        1,               1,            1  ]]

# Solves the equation A*x=b
b = [0,0,0,0,0,0,0,0,1]
h = solveSystem(M, b)

H = [[h[0], h[1], h[2]],    \
     [h[3], h[4], h[5]],    \
     [h[6], h[7], h[8]] ]

tImage = imageTransform(inputImage, maskImage, H)
```

CODE 10.2 Compute homography.

corners of the front face of the cube in the images in Figs. 10.3A and B (points shown as dots). Fig. 10.3C shows the result of applying the computed homography to the image in Fig. 10.3A. Notice that the cube's face defined by the corresponding points matches the face of the cube in the target image in Fig. 10.3B. However, other planes of the cube appear distorted. This occurs since a homography is a linear transformation that maps points in a planar surface. This is why this transformation is commonly called planar homography. Notice that we used four pairs of 2D image points to obtain the homography, but also three pairs of 3D world points can be used to solve the system in Eq. (10.3). Both approaches define the same homography. However, it is more evident



(A) Image (B) Target image (C) Homography

FIGURE 10.3 Computing a homography.

that three 3D points define a plane such that the mapping defines a transformation between two 3D planes.

Homographies are useful for several image processing operations like stitching and for creating transformations for far or planar objects. Perhaps the most important homography in computer vision is the one defined by the Epipolar geometry. Epipolar geometry defines a homography known as the fundamental matrix whose transformation aligns the rows of two images. The essential matrix is a particular form of the fundamental matrix, and it is used to find the relative position of two cameras from image correspondences [Hartley01]. The development of these homographies is based on the ideas of projections that describe the perspective camera.

10.3 The perspective camera

As discussed in Chapter 1, an image is formed by a complex process involving optics, electronics and mechanical devices. This process maps information in a 3D scene into pixels in an image. A camera model uses mathematical representations to describe the geometry in this process. Different models include different aspects of the image formation, and they are based on different assumptions or simplifications. The perspective camera is the most common model since it gives a good approximation of the geometrical optics in most cameras, and it can be considered a general model that includes other models as simplified cases.

Fig. 10.4 shows the model of the *perspective* camera. This model is also known as the *pinhole* camera since it describes the image formation process of a simple optical device with a small hole. This device is known as camera obscura, and it was developed in the 16th century as to aid artists to create the correct perspective in paintings. Light going

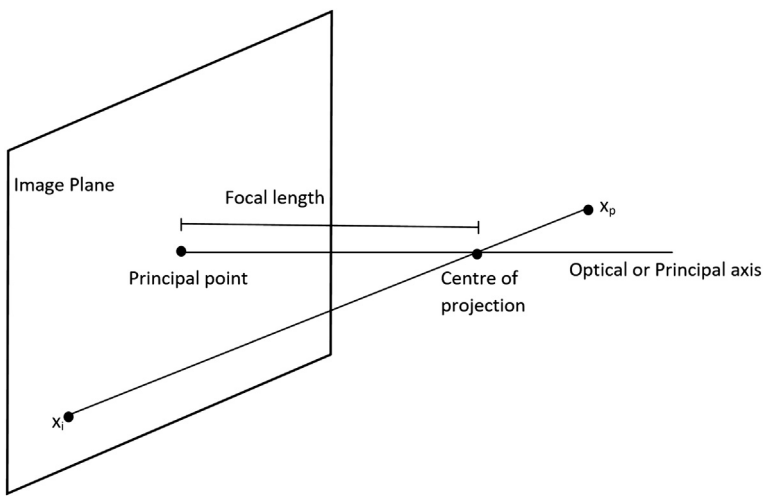


FIGURE 10.4 Pinhole model of perspective camera.

through a pinhole projects an image of a scene onto a back screen. The pinhole is called the centre of projection. Thus, a pixel is obtained by intersecting the image plane with the line between the 3D point and the centre of projection. In the projected image, parallel lines intersect at infinity giving a correct perspective.

Although based on an ancient device, this model represents an accurate description of modern cameras where light is focused in a single point by using lenses. In Fig. 10.1, the centre of projection corresponds to the pinhole. Light passes through the point and it is projected in the image plane. Fig. 10.5 illustrates an alternative configuration where light is focused back to the image plane. The models are equivalent: the image is formed by projecting points through the centre of projection; the point $\mathbf{x}_p$ is mapped into the point $\mathbf{x}_i$ in the image plane and the *focal length* determines the *zoom* distance.

The perspective camera model can be developed using algebraic functions, nevertheless the notation is greatly simplified by using matrix representations. In matrix form, points can be represented in Euclidean co-ordinates, yet a simpler notation is developed using *homogeneous co-ordinates*. Homogeneous co-ordinates represent the projection of points and planes as a simple multiplication.

10.3.1 Perspective camera model

The *perspective camera model* uses the algebra of the projective space to describe the way in which 3D space points are mapped into an image plane. By using homogeneous co-ordinates the geometry of image formation is simply defined by the projection of a 3D point into the plane by one special type of homography known as a projection. In a projection, the matrix $\mathbf{H}$ is not square, so a point in a higher dimension is mapped into a lower dimension. The perspective camera model is defined by the projection transformation,

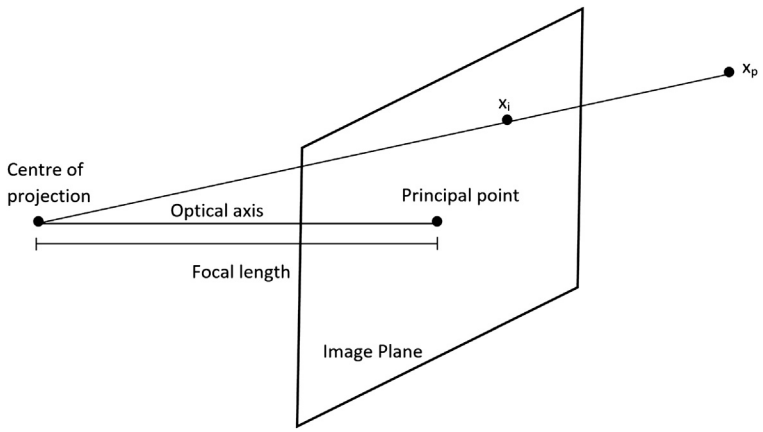


FIGURE 10.5 Perspective camera.

$$\begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.27}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}\mathbf{x}_p \tag{10.28}$$

Here, we have changed the elements from h to p to emphasise that we are using a projection. Also, we use $\mathbf{x}_i$ and $\mathbf{x}_p$ to denote the space and image points as introduced in Fig. 10.4. Notice that the point in the image is in homogeneous form, so the co-ordinates in the image are given by Eq. (10.3).

The matrix $\mathbf{P}$ is generally described by three geometric transformations as

$$\mathbf{P} = \mathbf{V}\,\mathbf{Q}\,\mathbf{M} \tag{10.29}$$

The matrix $\mathbf{M}$ transforms the 3D co-ordinates of $\mathbf{x}_p$ to make them relative to the camera system. That is, it transforms world co-ordinates into camera co-ordinates. That is, it transforms the co-ordinates of the point as if the camera were the origin of the co-ordinate system.

If the camera is posed in the world by a rotation and a translation, then the transformation between world and camera co-ordinates is given by the inverse of rotation and translation. We define this matrix as

$$\mathbf{M} = [\mathbf{R} \quad \mathbf{T}] \tag{10.30}$$

or more explicitly as

$$\mathbf{M} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.31}$$

The matrix $\mathbf{R}$ defines a rotation matrix and $\mathbf{T}$ a translation vector. The rotation matrix is composed by rotations along each axis. If α, β and γ are the rotation angles, then

$$\mathbf{R} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \tag{10.32}$$

Once the points are made relative to the camera frame, the transformation $\mathbf{Q}$ obtains the co-ordinates of the point projected into the image. As illustrated in Fig. 10.5, the focal length of a camera defines the distance between the centre of projection and the image plane. If f denotes the focal length of a camera, then

$$\mathbf{Q} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.33}$$

To understand this projection, let us consider the way a point is mapped into the camera frame as shown in Fig. 10.6. This figure illustrates the side view of the camera; to the right is the depth z axis and to the top down is the y axis. The image plane is shown as a dotted line. The point $\mathbf{x}_p$ is projected into $\mathbf{x}_i$ in the image plane. The tangent of the angle between the line from the centre of projection to $\mathbf{x}_p$ and the principal axis is given by

$$\frac{y_i}{f} = \frac{y_p}{z_p} \tag{10.34}$$

That is

$$y_i = \frac{y_p}{z_p} f \tag{10.35}$$

Using a similar rationale, we can obtain the value

$$x_i = \frac{x_p}{z_p} f \tag{10.36}$$

That is, the projection is obtained by multiplying by the focal length and by dividing by the depth of the point. The transformation matrix in Eq. (10.33) multiplies each co-ordinate by the focal length and copies the depth value into the last co-ordinate of the point. However, since Eq. (10.28) is in homogeneous co-ordinates, the depth value is actually used as divisor when obtaining co-ordinates of the point according to Eq. (10.3). Thus projection can be simply defined by a matrix multiplication as defined in Eq. (10.33).

The factors **M** and **Q** define the co-ordinates of a point in the image plane. However, the co-ordinates in an image are given in pixels. Thus, the last factor **V** is used to change from image co-ordinates to pixels. This transformation also includes a skew deformation to account for misalignments that may occur in the camera system. The transformation **V** is defined as

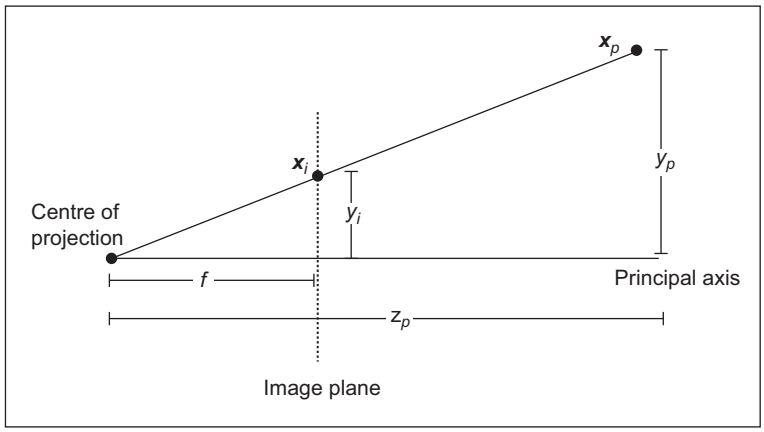


FIGURE 10.6 Projection of a point.

$$\mathbf{V} = \begin{bmatrix} k_u & k_u \cot(\varphi) & u_0 \\ 0 & k_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.37}$$

The constants k_u and k_v define the number of pixels in a world unit, the angle φ defines the skew angle and (u_0, v_0) is the position of the principal point in the image.

Fig. 10.7 illustrates the transformation in Eq. (10.37). The image plane is shown as a dotted rectangle, but it actually extends to infinity. The image is delineated by the axes u and v. A point (x_1, y_1) in the image plane has co-ordinates (u_1, v_1) in the image frame. As previously discussed in Fig. 10.1, the co-ordinates of (x_1, y_1) are relative to the principal point (u_0, v_0). As shown in Fig. 10.7, the skew displaces the point (u_0, v_0) by an amount given by,

$$a_1 = y_1 \cot(\varphi) \text{ and } c_1 = y_1/\sin(\varphi) \tag{10.38}$$

Thus, the new co-ordinates of the point after skew are

$$(x_1 + y_1 \cot(\varphi) \quad y_1/\sin(\varphi)) \tag{10.39}$$

To convert these co-ordinates to pixels, we need to multiply by the number of pixels that define a unit in the image plane and we also need to add the displacement (u_0, v_0) in pixels. That is,

$$u_1 = k_u x_1 + k_u y_1 \cot(\varphi) + u_0 \text{ and } v_1 = k_v y_1/\sin(\varphi) + v_0 \tag{10.40}$$

These algebraic equations are expressed in matrix form by Eq. (10.37).

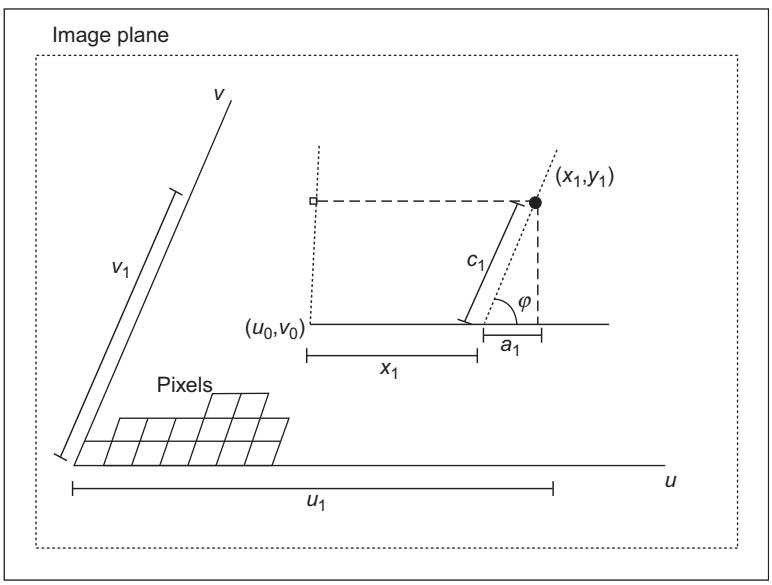


FIGURE 10.7 Image plane to pixels transformation.

10.3.2 Parameters of the perspective camera model

The perspective camera model in Eq. (10.27) has 12 elements. Thus, a particular camera model is completely defined by giving values to 12 unknowns. These unknowns are determined by the parameters of the transformations **M**, **Q** and **V**. The transformation **M** has three rotation angles (α, β, γ) and three translation parameters $(t_x,\ t_y,\ t_z)$. The transformation **V** has a single parameter f, whilst the transformation **Q** has the two translation parameters $(u_0,\ v_0)$, two scale parameters $(k_u,\ k_v)$ and one skew parameter φ. Thus, 12 parameters of the transformation matrices define 12 elements of the projection model. However, one parameter can be eliminated by combining the matrices **V** and **Q**. That is, the projection matrix in Eq. (10.29) can be written as,

$$\mathbf{P} = \begin{bmatrix} k_u & k_u \cot(\varphi) & u_0 \\ 0 & k_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.41}$$

or

$$\mathbf{P} = \begin{bmatrix} s_u & s_u \cot(\varphi) & u_0 \\ 0 & s_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.42}$$

for

$$s_u = f k_u \ \text{ and } \ s_v = f k_v \tag{10.43}$$

Thus, the camera model is actually defined by the eleven camera parameters $(\alpha, \beta, \gamma, t_x, t_y, t_z, u_0, v_0, s_u, s_v, \varphi)$.

The camera parameters are divided into two groups to indicate the parameters that are internal or external to the camera. The intrinsic parameters are $(u_0, v_0, s_u, s_v, \varphi)$ and the extrinsic are $(\alpha, \beta, \gamma, t_x, t_y, t_z)$. Generally, the intrinsic parameters do not change when capturing different images of a scene, so they are inherent to the system; they depend on the camera characteristics. The extrinsic parameters change by moving the camera in the world.

10.3.3 Computing a projection from an image

The process of computing the camera parameters is called *camera calibration.* Generally, the camera calibration process uses images of a 3D object with a geometrical pattern (e.g. checker board). The pattern is called the *calibration grid*. The 3D co-ordinates of the pattern are matched to 2D image points. The correspondences are used to solve the equation system in Eq. (10.28). Once the matrix **P** is known, the parameters in Eq. (10.41) can be obtained by observing the relationships between the projection matrix and the camera parameters [Truco98]. Implicit calibration is the process of finding the projection matrix without explicitly computing its physical parameters.

The straightforward solution of the linear system of equations was originally presented in [Abdel-Aziz71], and it is similar to the development presented in Section 10.2.4.

The solution starts by observing that according to Eq. (10.28), a point $\mathbf{x}_p = (x_p, y_p, z_p)$ is mapped into $\mathbf{x}_i = (x_i, y_i)$ by

$$x_i = \frac{x_p p_{1,1} + y_p p_{1,2} + z_p p_{1,3} + p_{1,4}}{x_p p_{3,1} + y_p p_{3,2} + z_p p_{3,3} + p_{3,4}} \quad \text{and} \quad y_i = \frac{x_p p_{2,1} + y_p p_{2,2} + z_p p_{2,3} + p_{2,4}}{x_p p_{3,1} + y_p p_{3,2} + z_p p_{3,3} + p_{3,4}} \tag{10.44}$$

That is,

$$\begin{aligned} x_p p_{1,1} + y_p p_{1,2} + z_p p_{1,3} + p_{1,4} - x_i\left(x_p p_{3,1} + y_p p_{3,2} + z_p p_{3,3} + p_{3,4}\right) = 0 \\ x_p p_{2,1} + y_p p_{2,2} + z_p p_{2,3} + p_{2,4} - y_i\left(x_p p_{3,1} + y_p p_{3,2} + z_p p_{3,3} + p_{3,4}\right) = 0 \end{aligned} \tag{10.45}$$

This can be written in matrix form as

$$\begin{aligned} [\ x_p \quad y_p \quad z_p \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad -x_i x_p \quad -x_i y_p \quad -x_i z_p \quad -x_i\ ]\ \mathbf{p}^{\mathrm{T}} = 0 \\ [\ 0 \quad 0 \quad 0 \quad 0 \quad x_p \quad y_p \quad z_p \quad 1 \quad -y_i x_p \quad -y_i y_p \quad -y_i z_p \quad -x_i\ ]\ \mathbf{p}^{\mathrm{T}} = 0 \end{aligned} \tag{10.46}$$

where $\mathbf{p}$ is the vector containing the coefficients of the projection. That is,

$$\mathbf{p} = [\ p_{1,1} \quad p_{1,2} \quad p_{1,3} \quad p_{1,4} \quad p_{2,1} \quad p_{2,2} \quad p_{2,3} \quad p_{2,4} \quad p_{3,1} \quad p_{3,2} \quad p_{3,3} \quad p_{3,4}\ ] \tag{10.47}$$

Thus, a pair of corresponding points defines two rows of a system of equations with 12 unknowns. Six points will provide 12 rows so that the system is determined. More than 12 points define an overdetermined system that can be solved using least squares.

Code 10.3 illustrates the system of equations defined by Eq. (10.46) for six corresponding points. The lists `pts` and `q` store the corresponding image and 3D points, respectively. In this example, we define the unseen vertex of the cube as origin of the 3D world, and the axes are aligned with the cube edges. The x-axis is towards the right, the y-axis upwards and the z-axis is towards the image plane. The points are defined by the corners of the cube. The matrix in Eq. (10.46) is stored in the variable `M`, and it is filled by using an iteration that takes a pair of corresponding points and evaluates the relation in Eq. (10.46). The solution of the system is stored in the variable `p`. Notice that Eq. (10.46) defines a homogenous system, thus to avoid the trivial solution, we need to include a normalisation. In this case, we set the last value of the coefficient to one. In general, a better normalisation consists on including the constraint $p_{3,1}^2 + p_{3,2}^2 + p_{3,3}^2 = 1$ [Faugeras87].

In order to illustrate the projection, Code 10.3 uses the computed projection to map 100×100 points from the cube's faces into the image. The points in the image are stored in the list `xy`. Since four neighbour points in the face of the cube are mapped as a polygon in the image, then the warping process defined in Code 10.1 can be used to fill the image with a colour. Fig. 10.8 shows the result of this process. Fig. 10.8A shows the original image. Fig. 10.8B shows the filled polygons formed by the projected points. This example uses a different colour for each cube's face. We can see that the computed projection matrix provides a description of how 3D points are mapped into the image.

Code 10.4 illustrates the use of the projection to get a transformation of an image. Similar to the previous example, the implementation computes the projection by using the corners of the cube. Also, the projection is used to map 3D points from the cube's faces into the image. The function `projectionCubePoints` obtains the projection points in a similar fashion to the implementation in Code 10.3. The function `getPointColours`

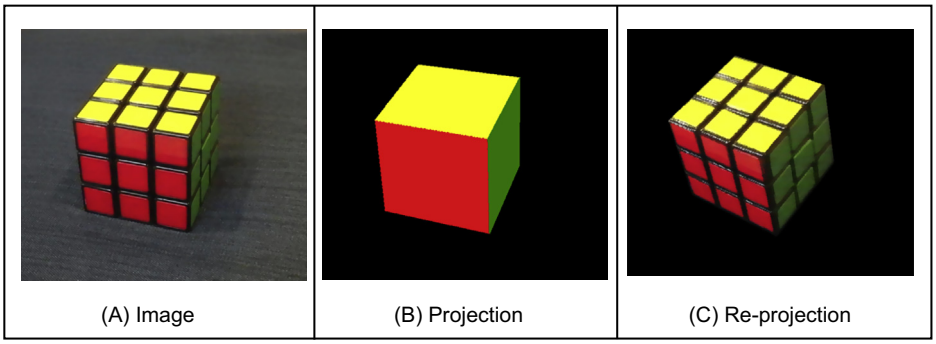


FIGURE 10.8 Image projections.

```
# Points in the cube image
pts = [[131-centreX,378-centreY],[110-centreX,188-centreY],
       [200-centreX,73-centreY],[412-centreX,100-centreY],
       [410-centreX,285-centreY],[349-centreX,418-centreY],
       [345-centreX,220-centreY]]

# Points in the 3D space
q = [[0,0,1],[0,1,1], [0,1,0],[1,1,0], [1,0,0],[1,0,1], [1,1,1]]

# Fill matrix
M = [ ]
for row in range(0,6):
    r1 = [ q[row][0],q[row][1], q[row][2],1,0,0,0,0,-pts[row][0]*q[row][0],  \
          -pts[row][0]*q[row][1],-pts[row][0]*q[row][2],-pts[row][0] ]
    r2 = [ 0,0,0,0,q[row][0],q[row][1], q[row][2],1,-pts[row][1]*q[row][0],  \
          -pts[row][1]*q[row][1],-pts[row][1]*q[row][2], -pts[row][1] ]
    M.append(r1)
    M.append(r2)

# Solves the equation M*p=r
r = [0,0,0,0,0,0,0,0,0,0,0,1]
p = solveSystem(M, r)
P = [[p[0], p[1], p[2], p[3]],    \
     [p[4], p[5], p[6], p[7]],    \
     [p[8], p[9], p[10], p[11]] ]

# Project world points into the image using the computed projection
npts = 100
origin, v1, v2 = [0,0,1], [1,0,0], [0,1,0]
xy = [ ]
for a in range(0, npts):
    rowxy = [ ]
    for b in range(0, npts):

        v1D = [a*v1[0]/float(npts-1), a*v1[1]/float(npts-1), a*v1[2]/float(npts-1)]
        v2D = [b*v2[0]/float(npts-1), b*v2[1]/float(npts-1), b*v2[2]/float(npts-1)]
        s = [origin[0]+v1D[0]+v2D[0], origin[1]+v1D[1]+v2D[1], origin[2]+v1D[2]+v2D[2]]

        sx = p[0]*s[0] + p[1]*s[1] + p[2]*s[2]  + p[3]
        sy = p[4]*s[0] + p[5]*s[1] + p[6]*s[2]  + p[7]
        sz = p[8]*s[0] + p[9]*s[1] + p[10]*s[2] + p[11]

        rowxy.append([int(sx/sz) + centreX, int(sy/sz) + centreY])

    xy.append(rowxy)
```

CODE 10.3 Compute projection.

```
# Obtain the projection
p = computeProjection(pts,q)

npts = 100
xy = projectionCubePoints(npts, p, centreX, centreY)
colours = getPointColours(xy, maskImage, inputImage)

# Transform points
qT = [ ]
angY = 0.3
angX = -0.2
for pointNum in range(0,len(q)):
    s = [q[pointNum][0]-.5, q[pointNum][1]-.5, q[pointNum][2]-.5]
    rx = .5 + cos(angY)*s[0] + sin(angY)*s[2]
    ry = .5 + sin(angX)*sin(angY)*s[0] + cos(angX)*s[1] - sin(angX)*cos(angY)*s[2]
    rz = .5 - cos(angX)*sin(angY)*s[0] + sin(angX)*s[1] + cos(angX)*cos(angY)*s[2]

    qT.append([rx,ry,rz])

p = computeProjection(pts,qT)
xy = projectionCubePoints(npts, p, centreX, centreY)

# Output image
tImage = createImageRGB(width, height)
fillImageColours(colours, xy, tImage)
showImageRGB(tImage)
```

CODE 10.4 Compute re-projection

obtains the colours of the projected points from the original image. That is, for each point in the cubes' face, it obtains the colour in the original image. A re-projection is obtained by projecting the 3D coloured points into another image to generate a new view. To explain this process, we should remember that according to Eq. (10.29) the projection is defined as

$$\mathbf{x}_i = \mathbf{VQM}\mathbf{x}_p \tag{10.48}$$

Since $\mathbf{M}$ defines the rotation and translation, then we can define a new view by changing the projection as

$$\mathbf{x}_i = \mathbf{VQM}'\mathbf{x}_p \tag{10.49}$$

If we define $\mathbf{M}' = \mathbf{ML}$,

$$\mathbf{x}_i = \mathbf{VQML}\mathbf{x}_p \text{ or } \boldsymbol{x_i} = \mathbf{PL}\mathbf{x}_p \tag{10.50}$$

That is, a new view can be obtained by transforming the points $\mathbf{x}_p$ by a rigid transformation $\mathbf{L}$ and then using the matrix projection. Accordingly, the implementation in Code 10.4 first transforms the original points in the list `q` into the points in the list `qT` by a transformation $\mathbf{L}$ that, in this example, only rotates along the x and y axis. Afterwards, the transformed points are used to compute a new projection that maps the transformed points into the image projection. The projection is used to map the faces of the cube into the new image, and the result is used in the warping process with the colours obtained from the original projection. Fig. 10.8C shows the result of the whole process. Other

views can be generated by changing the transformation **L**. The obtained view contains the geometry of the model and the textures of the original image.

10.4 Affine camera

Although the perspective camera model is probably the most common model used in computer vision, there are alternative models that are useful in particular situations. One alternative model of reduced complexity and that is useful in many applications is the *affine camera model*. This model is also called the *paraperspective* or *linear* model and it reduces the perspective model by setting the focal length f to infinity. Fig. 10.9 compares how the perspective and affine camera models map points into the image plane. The figure illustrates the projection of points from a side view, and it projects the corner points of a pair of objects represented by two rectangles. In the projective model, the projection produces changes of size in the objects according to their distance to the image plane; the far object is projected into a smaller area than the close object. The size

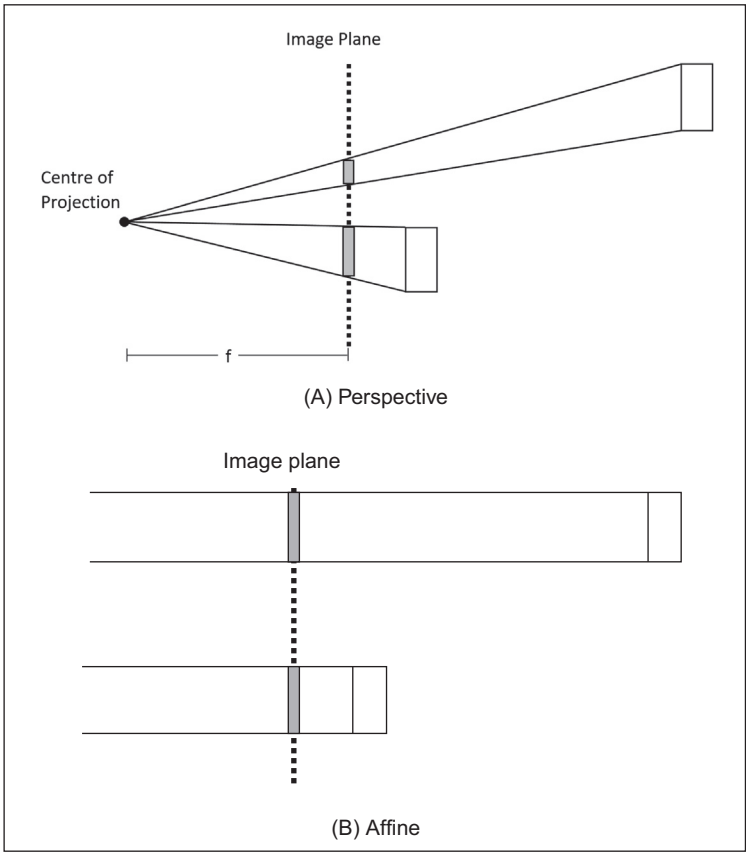


FIGURE 10.9 Perspective and affine camera models.

and distance relationship is determined by the focal length f. As we increase the focal length, projection lines decrease their slope and become horizontal. As illustrated in the figure, in the limit when the centre of projection is infinitely far away from the image plane, the lines do not intersect and the objects have the same projected area.

In spite of not accounting for changes in size due to distances, the affine camera provides a useful model when the depth position of objects in the scene with respect to the camera frame does not change significantly. This is the case in many indoor scenes and in many industrial applications where objects are aligned to a working plane. It is very useful to represent scenes on layers, that is, planes of objects with similar depth. Also affine models are simple and thus algorithms are more stable. Additionally, an affine camera is linear since it does not include the projection division.

10.4.1 Affine camera model

In the affine camera model, Eq. (10.27) is defined as

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.51}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}_\mathbf{A}\mathbf{x}_p \tag{10.52}$$

Here, we use the sub-index $\mathbf{A}$ to indicate that the affine camera transformation is given by a special form of the projection $\mathbf{P}$. The last row in Eq. (10.39) can be omitted. It is only shown to emphasise that it is a special case of the perspective model. However, at difference of the perspective camera, points in the image plane are actually in Euclidean co-ordinates. That is, the affine camera maps points from the projective space to the Euclidean plane.

Similar to the projection transformation, the transformation $\mathbf{A}$ can be factorised in three factors that account for the camera's rigid transformation, the projection of points from space into the image plane and for the mapping of points on the image plane into image pixels.

$$\mathbf{A} = \mathbf{V}\,\mathbf{Q}_\mathbf{A}\,\mathbf{M}_\mathbf{A} \tag{10.53}$$

Here, the sub-index $\mathbf{A}$ indicates that these matrices are the affine versions of the transformations defined in Eq. (10.29). We start by a rigid transformation as defined in Eq. (10.31). As in the case of the perspective model, this transformation is defined by the position of the camera and makes the co-ordinates of a point in 3D space relative to the camera frame.

$$\mathbf{M}_\mathbf{A} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.54}$$

That is,

$$\mathbf{M_A} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \tag{10.55}$$

The last row is added so the transformation $\mathbf{Q_A}$ can have four rows. We need four rows in $\mathbf{Q_A}$ in order to define a parallel projection into the image plane. Similar to the transformation $\mathbf{Q}$, the transformation $\mathbf{Q_A}$ projects a point in the camera frame into the image plane. The difference is that in the affine model, points in space are orthographically projected into the image plane. This can be defined by

$$\mathbf{Q_A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.56}$$

This defines a projection when the focal length is set to infinity. Intuitively, you can see that when transforming a point $\mathbf{x}_p^T = [\, x_p \quad y_p \quad z_p \quad 1 \,]$ by Eq. (10.56), the x and y co-ordinates are copied and the depth z_p value does not change the projection. Thus, Eqs. (10.34) and (10.35) for the affine camera become

$$x_i = x_p \text{ and } y_i = y_p \tag{10.57}$$

That is, the points in the camera frame are projected along the line $z_p = 0$. This is a line parallel to the image plane. The transformation $\mathbf{V}$ in Eq. (10.53) provides the pixel co-ordinates of points in the image plane. This process is exactly the same in the perspective and affine models and it is defined by Eq. (10.37).

10.4.2 Affine camera model and the perspective projection

It is possible to show that the affine model is a particular case of the perspective model by considering the alternative camera representation illustrated in Fig. 10.10. This figure is similar to Fig. 10.6. The difference is that in the previous model, the centre of the camera frame is in the centre of projection and in Fig. 10.10 it is considered to be the principal point (i.e. on the image plane). In general, the camera frame does not need to be located at a particular position in the camera, but it can be arbitrarily set. When set in the image plane, as illustrated in Fig. 10.10, the z camera co-ordinate of a point defines its depth in the image plane. Thus Eq. (10.34) is replaced by

$$\frac{y_i}{f} = \frac{h}{z_p} \tag{10.58}$$

From Fig. 10.10, we can see that $y_p = y_i + h$. Thus,

$$y_p = y_i + z_p \frac{y_i}{f} \tag{10.59}$$

Solving for y_i we have that

$$y_i = \frac{f\, y_p}{f + z_p} \tag{10.60}$$

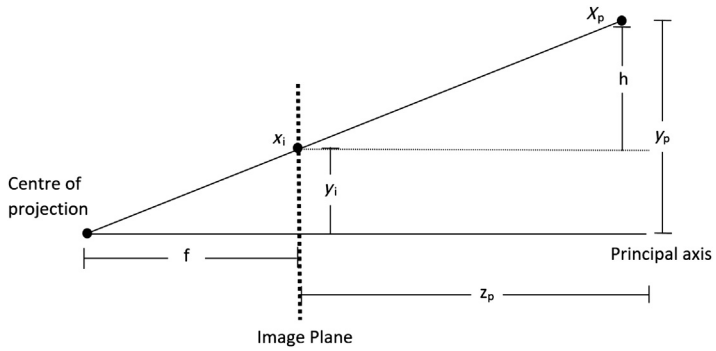


FIGURE 10.10 Projection of a point.

We can use a similar development to find the x_i co-ordinate. That is,

$$x_i = \frac{f\, x_p}{f + z_p} \tag{10.61}$$

Using homogeneous co-ordinates, Eqs. (10.60) and (10.61) can be written in matrix form as

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & f \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.62}$$

This equation is an alternative to Eq. (10.33); it represents a perspective projection. The difference is that Eq. (10.62) assumes that the camera axis is the located at the principal point of a camera. Using Eq. (10.62), it is easy to see the projection in the affine camera model as a special case of projection in the perspective camera model. To show that Eq. (10.33) becomes an affine model when f is set to be infinite, we define $B = 1/f$. Thus, Eq. (10.60) can be rewritten as,

$$y_i = \frac{y_p}{1 + B\, z_p} \quad \text{and} \quad x_i = \frac{x_p}{1 + B z_p} \tag{10.63}$$

or

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & B & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.64}$$

When f tends to infinity B tends to zero. Thus, the projection in Eq. (10.62) for an affine camera becomes

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.65}$$

The transformation in this equation is defined in Eq. (10.56). Thus, the projection in the affine model is a special case of the projection in the perspective model obtained by setting the focal length to infinity.

10.4.3 Parameters of the affine camera model

The affine camera model in Eq. (10.52) is composed of eight elements. Thus, a particular camera model is completely defined by giving values to eight unknowns. These unknowns are determined by the 11 parameters $(\alpha, \beta, \gamma, t_x, t_y, t_z, u_0, v_0, k_u, k_v, \varphi)$ defined in the matrices in Eq. (10.53). However, since we are projecting points orthographically into the image plane, the translation in depth is lost. This can be seen by combining the matrices $\mathbf{Q_A}$ and $\mathbf{M_A}$ in Eq. (10.53). That is,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.66}$$

or

$$\mathbf{G_A} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.67}$$

Thus Eq. (10.53) becomes

$$\mathbf{A} = \mathbf{V}\,\mathbf{G_A} \tag{10.68}$$

Similar to Eq. (10.55), the matrix $\mathbf{G_A}$ can be written as

$$\mathbf{G_A} = \begin{bmatrix} \mathbf{R_A} & \mathbf{T_A} \\ 0 & 1 \end{bmatrix} \tag{10.69}$$

and it defines the orthographic projection of the rigid transformation $\mathbf{M_A}$ into the image plane. According to Eq. (10.67), we have that the translation is,

$$\mathbf{T_A} = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \tag{10.70}$$

Since we do not have the translation t_z, we cannot determine if objects are far away or close to the camera. The rotation is defined according to Eq. (10.66) as

$$\mathbf{R_A} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & 0 & 0 \end{bmatrix} \tag{10.71}$$

Thus, the eight elements of the affine camera projection matrix are determined by the intrinsic parameters $(u_0, v_0, s_u, s_v, \varphi)$ and the extrinsic parameters $(\alpha, \beta, \gamma, t_x, t_y)$.

10.5 Weak perspective model

The *weak perspective model* defines a geometric mapping that stands between the perspective and the affine models. This model considers the distance between points in the scene is small relative to the focal length. Thus, Eqs. (10.34) and (10.35) are approximated by

$$y_i = \frac{y_p}{\mu_z} f \quad \text{and} \quad x_i = \frac{x_p}{\mu_z} f \tag{10.72}$$

for μ_z is the average z co-ordinate of all the points in a scene.

Fig. 10.11 illustrates two possible geometric interpretations for the relationships defined in Eq. (10.72). Fig. 10.11A, illustrates a two-step process wherein first all points are affine projected to a plane orthogonal to the image plane and at a distance μ_z. Points on this plane are then mapped into the image plane by a perspective projection. The projection on the plane $z = \mu_z$ simply replaces the z co-ordinates of the points by μ_z. Since points are assumed to be close, then this projection is a good approximation of the scene. Thus, the weak perspective model corresponds to a perspective model for scenes approximated by planes parallels to the image plane.

A second geometric interpretation of Eq. (10.72) is illustrated in Fig. 10.11B. In Eq. (10.72), we can combine the values f and μ_z into a single constant. Thus, Eq. (10.72) actually corresponds to a scaled version of Eq. (10.57). In Fig. 10.11B, objects in the scene are first mapped into the image plane by an affine projection and then the image is rescaled by a value f/μ_z. Thus, the affine model can be seen as a particular case of the weak perspective model when $f/\mu_z = 1$.

By following the two geometric interpretations discussed above, the weak perspective model can be formulated by changing the projection equations of the perspective or the affine models. For simplicity, we consider the weak perspective from the affine model. Thus, Eq. (10.56) should include a change in scale. That is,

$$\mathbf{Q_A} = \begin{bmatrix} f/\mu_z & 0 & 0 & 0 \\ 0 & f/\mu_z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.73}$$

By considering the definition in Eq. (10.53), we can move the scale factor in this matrix to the matrix $\mathbf{V}$. Thus the model for the weak-perspective model can be expressed as

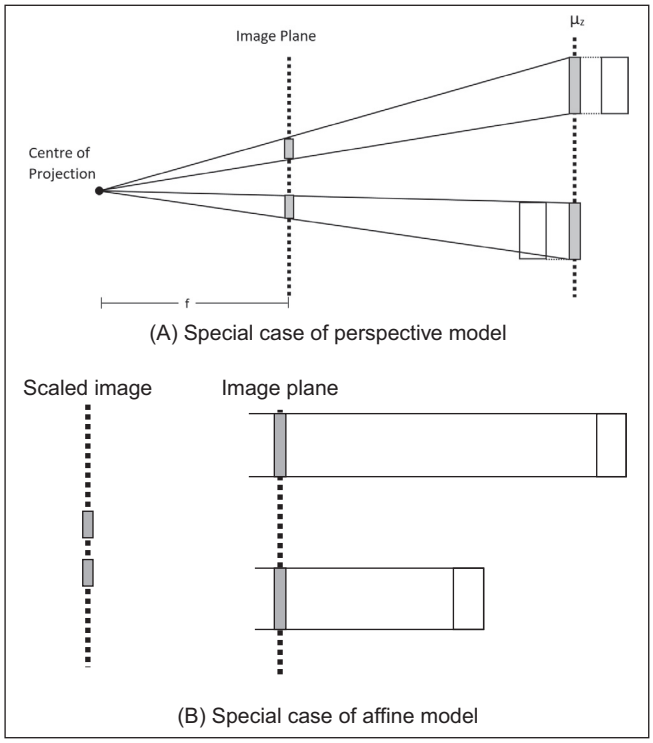


FIGURE 10.11 Weak perspective camera model.

$$\mathbf{P} = \begin{bmatrix} s_u & s_u \cot(\varphi) & u_0 \\ 0 & s_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.74}$$

for

$$s_u = fk_u/\mu_z \text{ and } s_v = fk_v/\mu_z \tag{10.75}$$

Thus the weak perspective is a scaled version of the affine model. The scale is a function of f that defines the distance of the centre of the camera to the image plane and the average distance μ_z.

10.6 Discussion

In this chapter, we have formulated the most common models of camera geometry. However, in addition to perspective and affine camera models there exist other models that consider different camera properties. For example, cameras built from a linear array of

sensors can be modelled by particular versions of the perspective and affine models obtained by considering a 1D image plane. These 1D camera models can also be used to represent stripes of pixels obtained by cameras with 2D image planes, and they have found an important application in mosaic construction from video images.

Besides image plane dimensionality, perhaps the most evident extension of camera models is to consider lens distortions. Small geometric distortions are generally ignored or deal with as noise in computer vision techniques. Strong geometric distortions such as the ones produced by wide-angle or fish-eye lens can be modelled by considering a spherical image plane or by non-linear projections. Camera models can include linear or non-linear distortions. The model of wide-angle cameras has found applications in environment map capture and panoramic mosaics.

The formulation of camera models is the basis of two central problems in computer vision. The first problem is known as camera calibration, and it focuses on computing the camera parameters from image data. There are many camera calibration techniques based on the camera model and different types of data. In general, camera calibration techniques are grouped into two main classes. Strong camera calibration assumes knowledge of the 3D co-ordinates of image points. Weak calibration techniques do not know 3D co-ordinates, but they assume knowledge of the type of motion of a camera. Also, some techniques focus on intrinsic or extrinsic parameters. The second central problem in computer vision is called scene reconstruction and focuses on recovering the co-ordinates of points in the 3D scene from image data.

10.7 Further reading

The main source for further material on this topic is the main textbook [Hartley01]. One of the early texts [Trucco98] has endured for many years and another [Cyganek11] considers recovery of 3D surfaces from stereo images. A more recent textbook [Laga19] 'establishes links between solutions proposed by different communities that studied 3D shape, such as mathematics and statistics, medical imaging, computer vision, and computer graphics'. These textbooks use the camera models to develop computer vision techniques such as reconstruction and motion estimation. There are several papers that review and compare different calibration techniques. A discussion about calibration types is presented in [Salvi02] and [Remondino06]. A survey including camera models, acquisition technologies and computer vision techniques is presented in [Strum11]. Perhaps the most important camera model after the projective camera is the projection defined by an omnidirectional camera. An *omnidirectional camera* has a visual field that covers a hemisphere or (approximately) the entire sphere [Danillidis00]. These types of images cannot be described using the conventional pinhole model because of the very high distortion.

References

[Aguado00] Aguado, A. S., Montiel, E., and Nixon, M. S., On the Intimate Relationship between the Principle of Duality and the Hough Transform, *Proceedings of the Royal Society of London*, **456**, pp 503-526, 2000.

[Abdel-Aziz71] Abdel-Aziz, Y.I., and Karara, H.M., Direct Linear Transformation into Object Space Co-ordinates in Close-Range Photogrammetry. *Proceedings of Symposium on Close-Range Photogrammetry*, pp 1-18, 1971.

[Cyganek11] Cyganek, B., and Siebert, J. P., *An Introduction to 3D Computer Vision Techniques and Algorithms*, John Wiley & Sons, NJ, USA, 2011.

[Daniilidis00] Daniilidis K., Geyer C., Omnidirectional Vision: Theory and Algorithms, *Proceedings of 15th International Conference on Pattern Recognition*, 2000.

[Faugeras87] Faugeras, O. D., and Toscani, G., Camera Calibration for 3D Computer Vision. *Proceedings of International Workshop on Industrial Applications of Machine Vision and Machine Intelligence*, Silken, Japan, pp 240-247, 1987.

[Hartley01] Hartley, R., and Zisserman, A., *Multiple View Geometry in Computer Vision*, Cambridge University Press, Cambridge, UK, 2001.

[Laga19] Laga, H., Guo, Y., Tabia, H., Fisher, R. B., Bennamoun, M., *3D Shape Analysis: Fundamentals, Theory, and Applications*, John Wiley & Sons, NJ, USA, 2019.

[Remondino06] Remondino and C. Fraser. Digital Camera Calibration Methods: Considerations and Comparisons. *Proceedings of ISPRS Commission V Symposium*, **6**, pp 266-272, 2006.

[Salvi02] Salvi J., Armangu X., and Batlle J., A Comparative Review of Camera Calibrating Methods with Accuracy Evaluation, *Pattern Recognition*, **35**, pp 1617−1635, 2002.

[Strum11] Sturm P., Ramalingam S., Tardif J-P., Gasparini S., and Barreto J., Camera Models and Fundamental Concepts Used in Geometric Computer Vision. *Foundations and Trends in Computer Graphics and Vision*, **6**(1-2), pp 1-183, 2011.

[Trucco98] Trucco E., Verri, A., *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, 1998.