CHAPTER

Appendix 1: Camera geometry fundamentals

10

CHAPTER OUTLINE HEAD

10.1 Image geometry

This book has focused on techniques of image processing that use intensity or color values of pixels to enhance and analyze images. Other image techniques include information about the geometry of image acquisition. These techniques are studied on the computer vision area and they are mainly applied to 3D scene analysis (Trucco and Verri, 1998; Hartley and Zisserman, 2001). This appendix 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 modeled.

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 scene into pixels in an image. A camera model uses mathematical representations

Feature Extraction & Image Processing for Computer Vision.

to describe this process. Different models include different aspects of the image formation and they are based on different assumptions or simplifications. This appendix explains basic aspects of common camera geometry models.

10.2 Perspective camera

Figure 10.1 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 sixteenth century to aid artists. Light going through a pinhole projects an image of a scene onto a back screen. The pinhole is called the center of projection. Thus, a pixel is obtained by intersecting the image plane with the line between the 3D point and the center 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 Figure 10.1, the center of projection corresponds to the pinhole. Light passes through the point and it is projected in the image plane. Figure 10.2 shows an alternative configuration where light is focused back to the image plane. The models are equivalent: the image is formed by projecting points through a single point; 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.

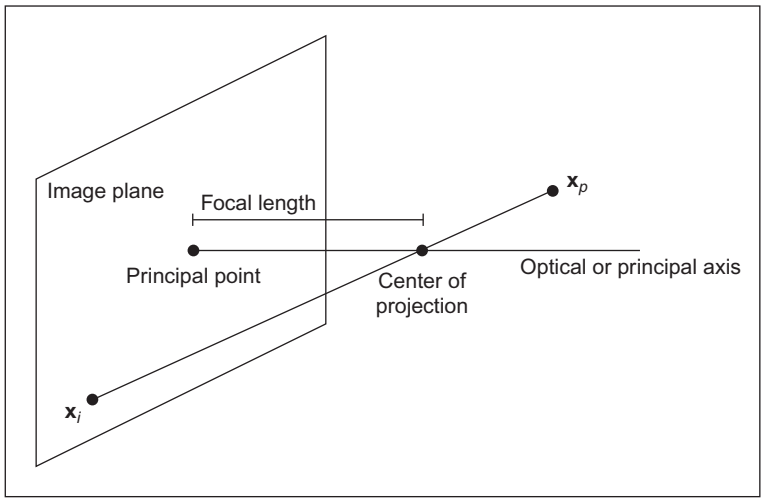

FIGURE 10.1

Pinhole model of perspective camera.

The perspective camera model is formulated by an equation that describes how a point in space is mapped into an image, where the center of projection is behind image plane. This formulation can be developed using algebraic functions, nevertheless the notation is greatly simplified by using matrix representations. In matrix form, points can be represented in Euclidian coordinates, yet a simpler notation is developed using *homogeneous coordinates*. Homogeneous coordinates simplify the formulation since translations and rotations are represented as matrix multiplications. Additionally, homogeneous coordinates represent the projection of points and planes as a simple multiplication. Thus, before formulating the model of the perspective camera, we first review the basic concepts of homogeneous coordinates.

10.3 Perspective camera model

10.3.1 Homogeneous coordinates and projective geometry

Euclidian geometry is algebraically represented by the *Cartesian coordinate system* in which points are defined by tuples of numbers. Each number is related to one axis and a set of axes determine the dimension. This representation is very natural to describe our 3D world and it is very useful on image processing to describe pixels in 2D images. Cartesian coordinates 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.

Projective geometry is algebraically represented by the homogeneous coordinate system. This representation is natural to formulate how we relate camera

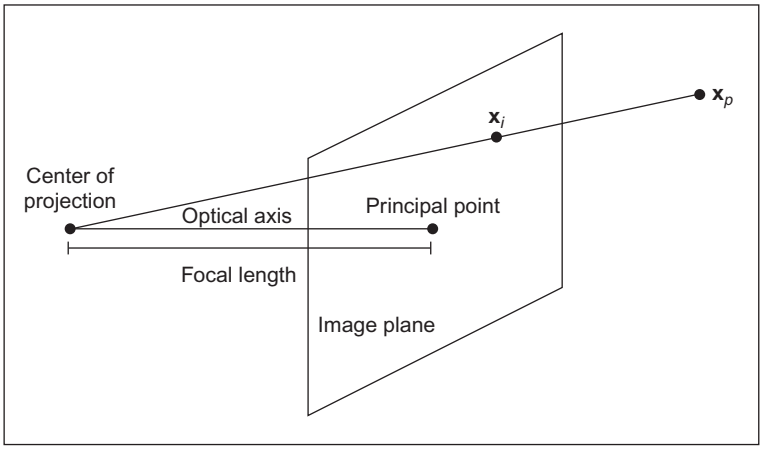

FIGURE 10.2

Perspective camera.

coordinates to "real-world" coordinates: 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 coordinates into the homogeneous coordinates. The 2D point with Cartesian coordinates

$$\mathbf{x}_c = \begin{bmatrix} x & y \end{bmatrix}^T \tag{10.1}$$

is mapped into homogeneous coordinates to the point

$$\mathbf{x}_h = \begin{bmatrix} wx & wy & w \end{bmatrix}^T \tag{10.2}$$

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

$$x = wx/w; \quad y = wy/w \tag{10.3}$$

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

$$\mathbf{x}_c = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{10.4}$$

is mapped into homogeneous form as

$$\mathbf{x}_h = \begin{bmatrix} wx & wy & wz & w \end{bmatrix}^T \tag{10.5}$$

These points are mapped back to Cartesian coordinates by

$$x = wx/w; \quad y = wy/w; \quad z = wz/w \tag{10.6}$$

Although it is possible to map points from Cartesian coordinates to homogeneous coordinates and vice versa, points in both system define different geometric spaces. Cartesian coordinates define the Euclidian space, and the points in homogeneous coordinates define the projective space. The projective space distinguishes a particular class of points defined when the last coordinate 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*.

10.3.1.1 Representation of a line and duality

The homogeneous representation of points has a very interesting connotation that relates points and lines. Let us consider the equation of a 2D line in Cartesian coordinates:

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

The same equation in homogeneous coordinates becomes

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

What is interesting is that points and lines now become indistinguishable. Both a point $\begin{bmatrix} x & y & z \end{bmatrix}^{\mathrm{T}}$ and a line $\begin{bmatrix} A & B & C \end{bmatrix}^{\mathrm{T}}$ are represented by triplets and they can be interchanged in the homogeneous equation of a line. Similarly, in the 3D projective space, points are indistinguishable to planes. This symmetry is known as the duality of the projective space that can be combined with the concept of concurrence and incidence to derivate the principle of duality (Aguado et al., 2000). The principle of duality constitutes an important concept for understanding the geometric relationship in the projective space, and the definition of the line can be used to derive the concept of ideal points.

10.3.1.2 Ideal points

We can use the algebra of homogeneous coordinates to find the intersection of parallel lines, planes, and hyperplanes. For simplicity, let us consider lines in the 2D plane. In the Cartesian coordinates, 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:

$$\begin{aligned} A_1x + B_1y + C_1z = 0 \\ A_2x + B_2y + C_2z = 0 \end{aligned} \tag{10.9}$$

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 from the first, we have

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

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

$$\mathbf{x}_{\mathrm{h}} = \begin{bmatrix} x & y & 0 \end{bmatrix}^{\mathrm{T}} \tag{10.11}$$

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

$$\mathbf{x}_{\mathrm{h}} = \begin{bmatrix} x & y & z & 0 \end{bmatrix}^{\mathrm{T}} \tag{10.12}$$

Since parallel lines are assumed to intersect at infinity, the points with the last coordinate 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 visualized by extending the Euclidian space as shown in Figure 10.3. This figure shows the 2D projective space as a set of points in the 3D Euclidian space. According to Eq. (10.3), points in the homogeneous space are mapped into the Euclidian space when $z = 1$. In the figure, this plane is called the Euclidian plane. Figure 10.3 shows two points in

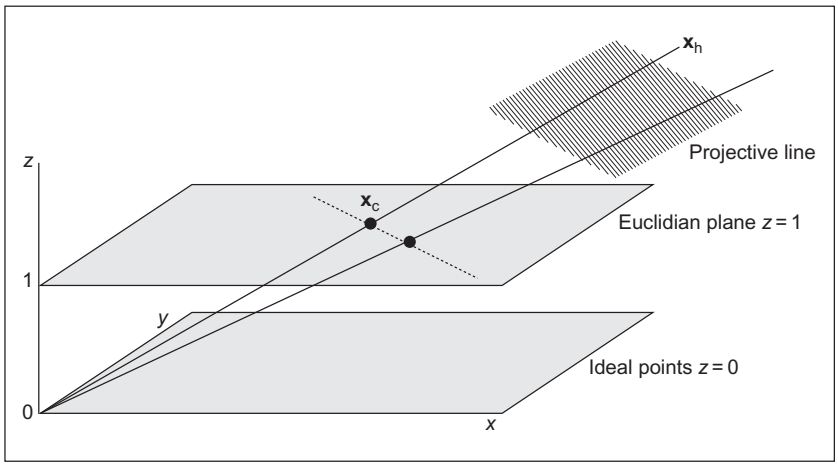

FIGURE 10.3

Model of the 2D projective space.

the Euclidian plane. These points define a line that is shown as a dotted line and it extends to infinity in the plane. In homogeneous coordinates, points in the Euclidian 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 coordinates of the line in the Euclidian plane define the plane between the two rays in the projective space. When two lines intersect in the Euclidian 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, i.e., a point in the plane $z = 0$.

Note that the origin $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ is ambiguous since it can define any point in homogeneous coordinates 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.

10.3.1.3 Transformations in the projective space

In practice, perhaps the most relevant aspect of homogeneous coordinates is the way transformations are algebraically represented. **Transformations** in Cartesian coordinates are known as **similarity** or **rigid** transformations since they do not change angle values. They define **rotations**, changes in **scale** and **translations** (position), and they can be algebraically represented by matrix multiplications and additions. A 2D point $\mathbf{x}_1$ is transformed to a point $\mathbf{x}_2$ by a similarity transformation as

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} s_x \\ s_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{10.13}$$

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

$$\mathbf{x}_2 = \mathbf{R}\,\mathbf{S}\,\mathbf{x}_1 + \mathbf{T} \tag{10.14}$$

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

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, i.e.,

$$\mathbf{x}_2 = \mathbf{A}\,\mathbf{S}\,\mathbf{x}_1 + \mathbf{T} \tag{10.15}$$

Affine transformations do not preserve the value of angles, but they preserve parallel lines. The principles and theorems studied under similarities define Euclidian 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, and they are defined in homogeneous coordinates. A 2D point $\mathbf{x}_1$ is transformed to a point $\mathbf{x}_2$ by a homography as

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} \tag{10.16}$$

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

$$\mathbf{x}_2 = H\,\mathbf{x}_1 \tag{10.17}$$

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

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cos(\theta) & s_x \sin(\theta) & t_x \\ -s_y \sin(\theta) & s_y \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_2 \\ 1 \end{bmatrix} \tag{10.18}$$

or in a more general form as

$$\mathbf{x}_2 = \begin{bmatrix} \mathbf{R}\ \mathbf{S} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \mathbf{x}_1 \tag{10.19}$$

An affine transformation is defined as

$$\mathbf{x}_2 = \begin{bmatrix} \mathbf{A} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \mathbf{x}_1 \tag{10.20}$$

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 10.3.2, this plane defines the Euclidian plane. Thus, these transformations are limited to Euclidian points.

10.3.2 Perspective camera model analysis

The *perspective camera model* uses the algebra of the projective space to describe the way in which space points are mapped into an image plane. The mapping can also be defined using Euclidian transformations, but the algebra becomes too elaborated. By using homogeneous coordinates, 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 a projection transformation as

$$\begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.21}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}\mathbf{x}_p \tag{10.22}$$

Here, we have changed the elements from h to p to emphasize 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 Figure 10.1. Note that the point in the image is in homogeneous form, so the coordinates in the image are given by Eq. (10.3).

The matrix $\mathbf{P}$ models three geometric transformations, so it can be factorized as

$$\mathbf{P} = \mathbf{V}\ \mathbf{Q}\ \mathbf{M} \tag{10.23}$$

The matrix $\mathbf{M}$ transforms the 3D coordinates of $\mathbf{x}_p$ to make them relative to the camera system. That is, it transforms world coordinates into camera coordinates. Note that the point is not transformed, but we obtain its coordinates as if the camera were the origin of the coordinate system.

If the camera is posed in the world by a rotation $\mathbf{R}$ and a translation $\mathbf{T}$, then the transformation between world and camera coordinates is given by the inverse of rotation and translation. We define this matrix as

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \tag{10.24}$$

or more explicitly as

$$\mathbf{M} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.25}$$

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

$$\mathbf{R} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \tag{10.26}$$

Once the points are made relative to the camera frame, the transformation **Q** obtains the coordinates of the point projected in the image. As shown in Figure 10.1, the focal length of a camera defines the distance between the center of projection and the image plane. If f denotes the focal length of a camera, then

$$\mathbf{Q} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.27}$$

To understand this projection, let us consider the way a point is mapped into the camera frame as shown in Figure 10.4. This figure shows 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 center of projection to $\mathbf{x}_p$ and the principal axis is given by

$$\frac{y_i}{f} = \frac{y_p}{z_p} \tag{10.28}$$

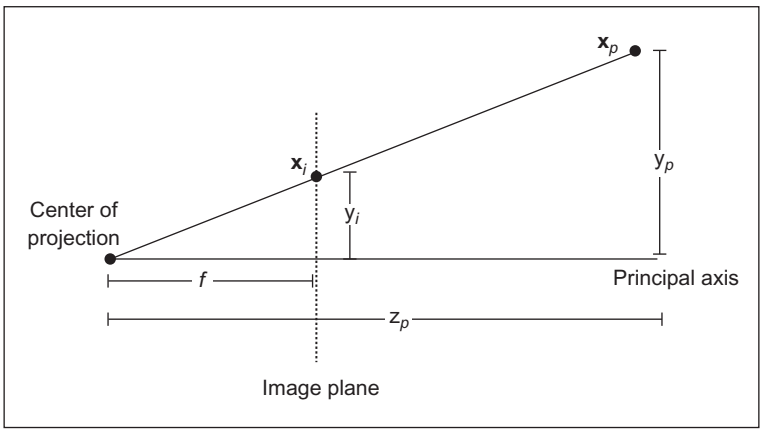

FIGURE 10.4

Projection of a point.

i.e.,

$$y_i = \frac{y_p}{z_p} f \tag{10.29}$$

Using a similar rationale we can obtain the value

$$x_i = \frac{x_p}{z_p} f \tag{10.30}$$

That is, the projection is obtained by multiplying by the focal length and by dividing by the depth of the point. Equation (10.27) multiplies each coordinate by the focal length and copies the depth value into the last coordinate of the point. However, since Eq. (10.21) is in homogeneous coordinates, the depth value is actually used as divisor when obtaining coordinates of the point according to Eq. (10.3). Thus projection can be simply defined by a matrix multiplication factor defined in Eq. (10.27).

The factors **M** and **Q** define the coordinates of a point in the image plane. However, the coordinates in an image are given in pixels. Thus, the last factor **V** is used to change from image coordinates 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

$$\mathbf{V} = \begin{bmatrix} k_u & k_u \cot(\varphi) & u_0 \\ 0 & k_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.31}$$

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.

Figure 10.5 shows the transformation in Eq. (10.31). The image plane is shown as a dotted rectangle, but it actually extends to infinity. The image is delineated by the axis u and v. A point (x_1,y_1) in the image plane has coordinates (u_1,v_1) in the image frame. As discussed in Figure 10.1, the coordinates (x_1,y_1) are relative to the principal point (u_0,v_0). As shown in Figure 10.5, the skew displaces the point form (u_0,v_0) by an amount given by

$$a_1 = y_1 \cot(\varphi); \quad c_1 = y_1/\sin(\varphi) \tag{10.32}$$

Thus, the new coordinates of the point after skew are

$$(x_1 + y_1 \cot(\varphi), \quad c_1 = y_1/\sin(\varphi)) \tag{10.33}$$

To convert these coordinates 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). That is

$$u_1 = k_u x_1 + k_u y_1 \cot(\varphi) + u_0; \quad v_1 = k_v y_1/\sin(\varphi) + v_0 \tag{10.34}$$

These algebraic equations are expressed in matrix form by Eq. (10.31).

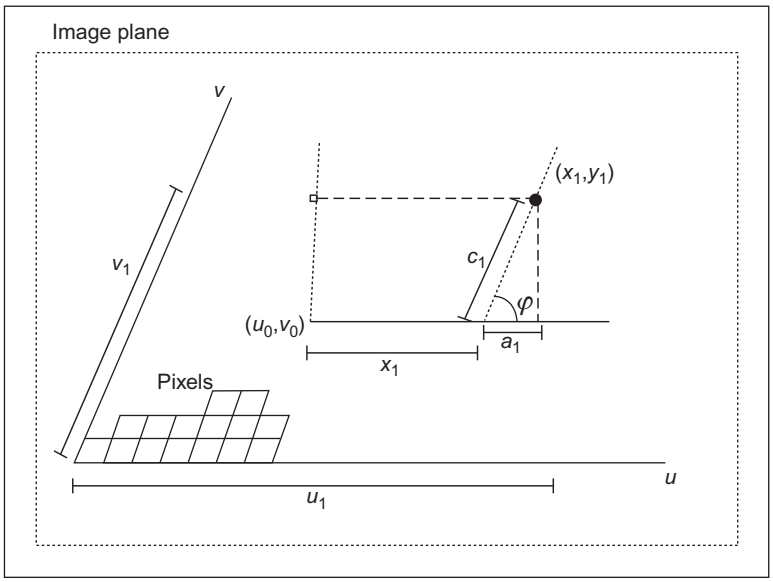

FIGURE 10.5

Image plane to pixels transformation.

10.3.3 Parameters of the perspective camera model

The perspective camera model in Eq. (10.21) 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, while 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, we need to set up 12 parameters to determine the elements of the projection matrix. However, one parameter can be eliminated by combining the matrices **V** and **Q**. That is, the projection matrix in Eq. (10.23) can be written as

$$\mathbf{P} = \begin{bmatrix} k_u & k_u \cot(\varphi) & u_0 \\ 0 & k_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.35}$$

or

$$\mathbf{P} = \begin{bmatrix} s_u & s_u \cot(\varphi) & u_0 \\ 0 & s_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{bmatrix} \tag{10.36}$$

for

$$s_u = fk_u; \quad s_v = fk_v \tag{10.37}$$

Thus, the camera model is actually defined by 11 camera parameters (α, β, γ, t_x, t_y, t_z, u_0, v_0, s_u, s_v, φ).

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, and φ, and the extrinsic are α, β, γ, t_x, t_y, and t_z. Generally, the intrinsic parameters do not change from scene to 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.

10.4 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**. Figure 10.6 shows how the perspective and affine camera models map points into the image plane. The figure shows 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 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 shown in Figure 10.6, in the limit when the center of projection is infinitely far away from the image plane, the lines do not intersect and the objects have the same projected area in the image.

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, i.e., planes of objects with similar depth. Also affine models are simple and thus algorithms are more stable, and an affine camera is linear since it does not include the projection division as given in Eqs (10.28−10.30).

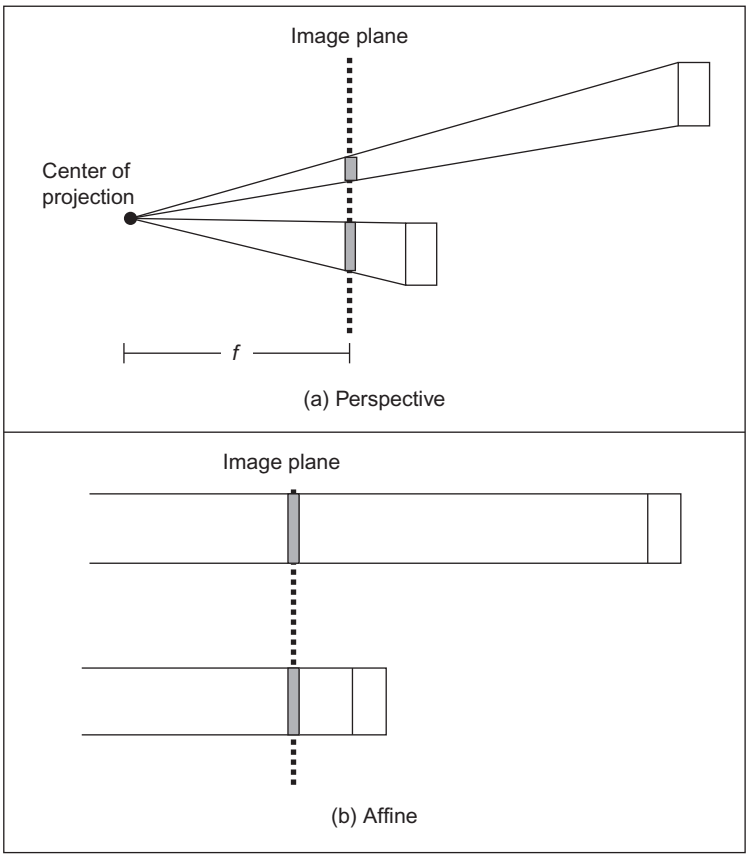

FIGURE 10.6

Perspective and affine camera models.

10.4.1 Affine camera model

For the affine camera model, Eq. (10.21) is changed to

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.38}$$

This equation can be written in short form as

$$\mathbf{x}_i = \mathbf{P}_\mathbf{A}\mathbf{x}_p \tag{10.39}$$

Here, we use the subindex **A** to indicate that the affine camera transformation is given by a special form of the projection **P**. The last row in Eq. (10.39) can be omitted. It is shown in the notation to emphasize 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 Euclidian coordinates. That is, the affine camera maps points from the projective space to the Euclidian plane.

Similar to the projection transformation, the transformation **A** can be factorized 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.

$$\mathbf{A} = \mathbf{V}\,\mathbf{Q}_\mathbf{A}\,\mathbf{M}_\mathbf{A} \tag{10.40}$$

Here, the subindex **A** indicates that these matrices are the affine versions of the transformations defined in Eq. (10.23). We start by a rigid transformation as defined in Eq. (10.25). As in the case of the perspective model, this transformation is defined by the position of the camera and makes the coordinates of a point in 3D space relative to the camera frame:

$$\mathbf{M}_\mathbf{A} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.41}$$

i.e.,

$$\mathbf{M}_A = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \tag{10.42}$$

The last row is added so the transformation $\mathbf{Q}_\mathbf{A}$ can have four rows. We need four rows in $\mathbf{Q}_\mathbf{A}$ in order to define a parallel projection into the image plane. Similar to the transformation **Q**, the transformation $\mathbf{Q}_\mathbf{A}$ 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

$$\mathbf{Q}_\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.43}$$

This defines a projection when the focal length is set to infinity. Intuitively, you can see that when transforming a point $\mathbf{x}_p^\mathrm{T} = [\,x_p \quad y_p \quad z_p \quad 1\,]$ by Eq. (10.43), the x and y coordinates are copied and the depth z_p value does not change the projection. Thus, Eqs (10.29) and (10.30) for the affine camera become

$$x_i = x_p; \quad y_i = y_p \tag{10.44}$$

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. (10.40) provides the pixel coordinates of points in the image plane. This process is exactly the same in the perspective and affine models and it is defined by Eq. (10.31).

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 shown in Figure 10.7. This figure is similar to the figure used to explain Eq. (10.27). The difference is that in the previous model, the center of the camera frame is in the center of projection, and in Figure 10.7 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 particular position in the camera, but it can be arbitrarily set. When set in the image plane, as shown in Figure 10.7, the z camera coordinate of a point defined their depth in the image plane. Thus, Eq. (10.28) is replaced by

$$\frac{y_i}{f} = \frac{h}{z_p} \tag{10.45}$$

From Figure 10.7, we can see that $y_p = y_i + h$. Thus,

$$y_p = y_i + z_p \frac{y_i}{f} \tag{10.46}$$

Solving for y_i, we have

$$y_i = \frac{f\, y_p}{f + z_p} \tag{10.47}$$

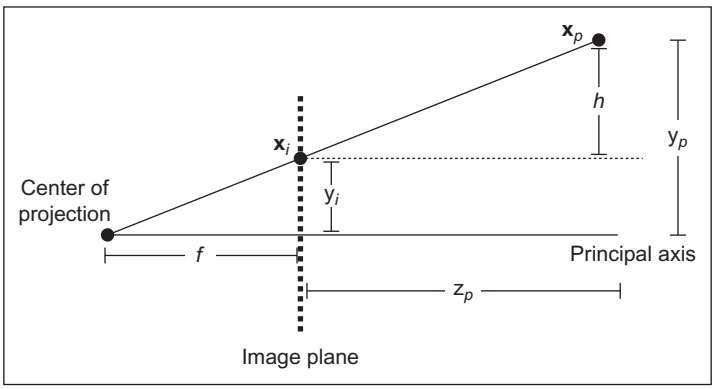

FIGURE 10.7

Projection of a point.

We can use a similar development to find the x_i coordinate. That is,

$$x_i = \frac{f\, x_p}{f + z_p} \tag{10.48}$$

Using homogeneous coordinates, Eqs (10.47) and (10.48) can be written in matrix form as

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & f \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.49}$$

This equation is an alternative to Eq. (10.27); it represents a perspective projection. The difference is that in Eq. (10.49), we assume that the camera axis is located at the principal point of a camera. Using Eq. (10.49), 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. (10.29) becomes an affine model when f is set to be infinite, we define $B = 1/f$. Thus,

$$y_i = \frac{y_p}{1 + B\, z_p}; \quad x_i = \frac{x_p}{1 + B\, z_p} \tag{10.50}$$

or

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & B & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.51}$$

When f tends to infinity, B tends to zero. Thus, the projection in Eq. (10.51) for affine camera becomes

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{10.52}$$

The transformation in this equation is defined in Eq. (10.43). 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 as expressed in Eq. (10.38) 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 (α, β, γ, t_x, t_y, t_z, u_0, v_0, k_u, k_v, and φ) defined in the matrices in Eq. (10.40). 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. (10.40), i.e.,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.53}$$

or

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

Thus, Eq. (10.40) becomes

$$\mathbf{A} = \mathbf{V}\,\mathbf{G_A} \tag{10.55}$$

Similar to Eq. (10.42), the matrix $\mathbf{G_A}$ can be written as

$$\mathbf{G_A} = \begin{bmatrix} \mathbf{R_A} & \mathbf{T_A} \\ 0 & 1 \end{bmatrix} \tag{10.56}$$

and it defines the orthographic projection of the rigid transformation $\mathbf{M_A}$ into the image plane. According to Eq. (10.53),

$$\mathbf{T_A} = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \tag{10.57}$$

Since we do not have t_z, we cannot determine if they are far away or close to the camera. A small object does not mean it is an object far away. According to Eq. (10.53), we also have

$$\mathbf{R_A} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & 0 & 0 \end{bmatrix} \tag{10.58}$$

Thus, the eight elements of the affine camera projection matrix are determined by the intrinsic parameters (u_0, v_0, s_u, s_v, φ) and the extrinsic parameters (α, β, γ, 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 the

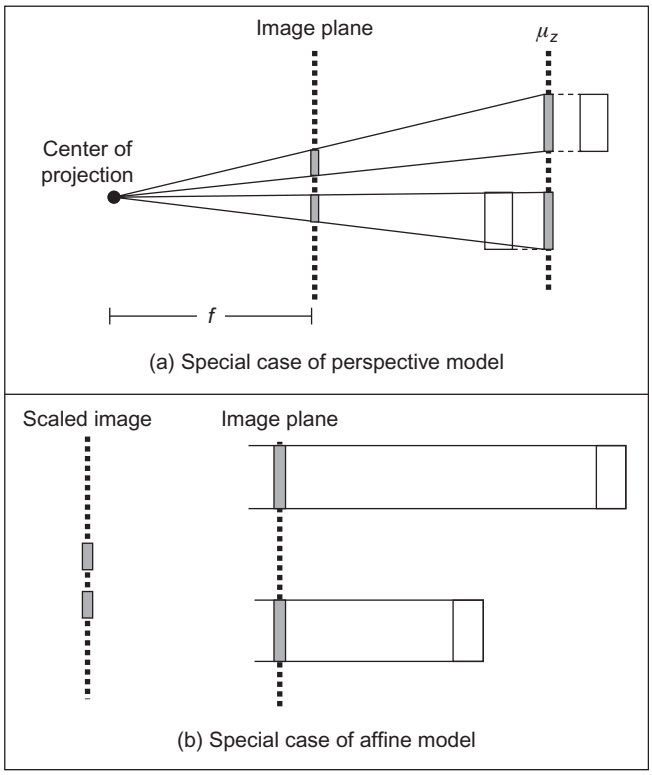

FIGURE 10.8

Weak perspective camera model.

points in the scene is small relative to the focal length. Thus, Eqs (10.29) and (10.30) are approximated by

$$y_i = \frac{y_p}{\mu_z} f; \quad x_i = \frac{x_p}{\mu_z} f \tag{10.59}$$

for μ_z is the average z coordinate of all the points in a scene.

Figure 10.8 shows two possible geometric interpretations for the relationships defined in Eq. (10.59). Figure 10.8(a) shows 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 coordinates of the points by μ_z. Since points are assumed to be close, this projection is a good approximation of the scene. Thus, the weak perspective model corresponds to a perspective model for scenes approximated by planes parallel to the image plane.

A second geometric interpretation of Eq. (10.59) is shown in Figure 10.8(b). In Eq. (10.59), we can combine the values f and μ_z into a single constant. Thus, Eq. (10.59) actually corresponds to a scaled version of Eq. (10.44). In Figure 10.8(b), 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. Additionally, it can also be formulated by considering the camera model presented in Section 10.3.2. For simplicity, we consider the weak perspective model from the affine model. Thus, Eq. (10.43) should include a change in scale, i.e.,

$$\mathbf{Q_A} = \begin{bmatrix} f/\mu_z & 0 & 0 & 0 \\ 0 & f/\mu_z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.60}$$

By considering the definition in Eq. (10.40), we can move the scale factor in this matrix to matrix $\mathbf{V}$. Thus, the model for the weak perspective model can be expressed as

$$\mathbf{P} = \begin{bmatrix} s_u & s_u \cot(\varphi) & u_0 \\ 0 & s_v \sin(\varphi) & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.61}$$

for

$$s_u = fk_u/\mu_z; \quad s_v = fk_v/\mu_z \tag{10.62}$$

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 center of the camera to the image plane and the average distance μ_z.

10.6 Example of camera models

This section explains the mapping of points into an image frame for the perspective and affine camera models. Code 10.1 contains the functions used to create figures: `DrawPoints`, `DrawImagePoints` `DrawWorldFrame`, `DrawImagePlane`, `DrawPerspectiveProjectionLines`, and `DrawAffineProjectionLines`.

The function `DrawPoints` draws a set of points given its three world coordinates. In our example, it is used to draw points in the world. The function `DrawImagePoints` draws points on the image plane. It uses homogeneous coordinates, so it implements Eq. (10.3). The function `DrawWorldFrame` draws the three axes given the location of the origin. This is used to illustrate the world frame

and the origin is always set to zero. The function `DrawImagePlane` draws a rectangle that represents the image plane. It also draws a point to exemplify the location of the center of the camera. In our examples, we assume the center of the camera is at the position of the focal point, for both the perspective and affine models.

The functions `DrawPerspectiveProjectionLines` and `DrawAffineProjection Lines` are used to explain the projection of points into the image plane for the perspective and affine projections, respectively. These functions take a vector of points and trace a line to the image plane that represents the projection. For perspective, the lines intersect the focal point, and for affine they are projected parallel to the image plane.

```
%*********************************************
%Draw a set of points
%---------------------------------------------
function DrawPoints(P,colour)
%---------------------------------------------

[r,c]=size(P);
plot3(P(1,1:c),P(2,1:c),P(3,1:c),'+','color',colour);

%*********************************************
%Draw a set of Image points
%---------------------------------------------
function DrawImagePoints(P,C,colour)
%---------------------------------------------

[r,c]=size(P);

for column=1:c
   P(1:r-1,column)=P(1:r-1,column)/P(3,column);
end

plot(P(1,1:c),P(2,1:c),'+','color',colour);
axis([0 C(10) 0 C(11)]);

%*********************************************
%Draw a co-ordinate frame
%---------------------------------------------
function DrawWorldFrame(x0,x1,y0,y1,z0,z1);
%---------------------------------------------

axis equal; %same aspect ratio
axis([x0,x1,y0,y1,z0,z1]);

xlabel('X','FontSize',14);
ylabel('Y','FontSize',14);
zlabel('Z','FontSize',14);
```

CODE 10.1

Drawing functions.

```
grid on;
hold on;

%***********************************************
%Draw an image plane
%-----------------------------------------------
function DrawImagePlane(C,dx,dy);
%-----------------------------------------------
%Draw camera origin
plot3(C(1),C(2),C(3),'o');  %optic centre
plot3(C(1),C(2),C(3),'+');  %optic centre

%co-ordinates of 4 points on the
%image plane (to draw a rectangle)
p(1,1)=-dx/2; p(2,1)=-dy/2; p(3,1)=C(7); p(4,1)=1;
p(1,2)=-dx/2; p(2,2)=+dy/2; p(3,2)=C(7); p(4,2)=1;
p(1,3)=+dx/2; p(2,3)=+dy/2; p(3,3)=C(7); p(4,3)=1;
p(1,4)=+dx/2; p(2,4)=-dy/2; p(3,4)=C(7); p(4,4)=1;

%CW: Camera to world transformation
CW=CameraToWorld(C);

%transform co-ordinates to world coordinates
P(:,1)=CW*p(:,1);
P(:,2)=CW*p(:,2);
P(:,3)=CW*p(:,3);
P(:,4)=CW*p(:,4);

%draw image plane
patch(P(1,:),P(2,:),P(3,:),[.9,.9,1]);

%***********************************************
%Draw a line between optical centre and 3D points
%
%-----------------------------------------------------
function DrawPerspectiveProjectionLines (o,P,colour);
%-----------------------------------------------------

[r,c]=size(P);
for i=1:c
  plot3([o(1) P(1,i)],[o(2) P(2,i)],[o(3) P(3,i)],'color',colour);
%optic centre
end;

%***********************************************
%Draw a line between image plane and 3D points
%
%-----------------------------------------------
function DrawAffineProjectionLines(z,P,colour);
%-----------------------------------------------
```

CODE 10.1

(Continued)

```
[r,c]=size(P);
for column=1:c
  plot3([P(1,column) P(1,column)],[P(2,column) P(2,column)],[z(3)
P(3,column)],'color',colour);  %optic centre
end;
```

CODE 10.1

(Continued)

In the example, we group the camera parameters in the vector $\mathbf{C} = [x_0, y_0, z_0, a, b, g, f, u_0, v_0, k_x, k_y]$. The first six elements define the location and rotation parameters. The value of f defines the focal length. The remaining parameters define the location of the optical center and the pixel size. For simplicity, we assume there is no skew. Code 10.3 contains the functions that compute camera transformations from camera parameters. The function `CameraToWorld` computes a matrix that defines the position of the camera. It poses the camera in the world. Its inverse is computed in the function `WorldToCamera` and it defines the matrix in Eq. (10.25). The inverse is simply obtained by the transpose of the rotation and by changing the signs of the translation. The matrix obtained from `WorldToCamera` can be used to obtain the coordinates of world points in the camera frame.

The function `ImageToCamera` in Code 10.2 obtains the inverse of the transformation defined in Eq. (10.31). This is used to draw points in pixel coordinates. The pixels coordinates are converted in world coordinates that are then drawn to show its position.

```
%*********************************************
%Compute matrix that transforms co-ordinates
%in the camera frame to the world frame%
%---------------------------------------------
 function CW=CameraToWorld(C);
%---------------------------------------------

%rotation
Rx=[cos(C(6))  sin(C(6))  0
   -sin(C(6))  cos(C(6))  0
       0         0        1];

Ry=[cos(C(5))     0     sin(C(5))
       0          1        0
   -sin(C(5))     0     cos(C(5))];
```

CODE 10.2

Transformation function.

```
Rz=[1         0           0
    0     cos(C(4))   sin(C(4))
    0    -sin(C(4))   cos(C(4))];

%translation
T=[C(1) C(2) C(3)]';

%transformation
CW=(Rz*Ry*Rx);
CW(:,4)=T;

%*********************************************
%Compute matrix that transforms co-ordinates
%in the world frame to the camera frame
%---------------------------------------------
 function WC=WorldToCamera(c);
%---------------------------------------------

%translation T'=-R'T
%for R'=inverse rotation
Rx=[cos(C(6)) -sin(C(6))  0
    sin(C(6))  cos(C(6))  0
       0         0        1];

Ry=[cos(C(5))    0   -sin(C(5))
       0         1      0
    sin(C(5))    0    cos(C(5))];

Rz=[1        0           0
    0     cos(C(4))  -sin(C(4))
    0     sin(C(4))   cos(C(4))];

T=[-c(1) -c(2) -c(3)]';

%transformation
WC=(Rz*Ry*Rx); %rotation inverse
Tp=WC*T;       %translation
WC(:,4)=Tp;    %compose homegeneous form

%*********************************************
%Convert from homegeneous co-ordinates in pixels
%to distance co-ordinates in the camera frame
%---------------------------------------------
function p=ImageToCamera(P,C);
%---------------------------------------------

%inverse of K
Ki=[1/C(10)        0     -C(8)/C(10)
      0         1/C(11)  -C(9)/C(11)
      0            0         1 ];
```

CODE 10.2

(Continued)

```
%co-ordinate in distance units
p=Ki*P;

%co-ordinates in the image plane
p(1,:)=p(1,:)./p(3,:);
p(2,:)=p(2,:)./p(3,:);
p(3,:)=p(3,:)./p(3,:);

%the third co-ordinate gives the depth
%the focal length C(7) defines depth
p(3,:)=p(3,:).*C(7);

%include homegeneous co-ordinates
p(4,:)=p(1,:)/p(1,:);
```

CODE 10.2

(Continued)

Code 10.3 contains two functions that compute the projection matrices for the perspective and affine camera models. Both functions start by computing the matrix that transforms the world points into the camera frame. For the affine model, the dimensions of the matrix transformation are augmented according to Eq. (10.42). For the perspective model, the world-to-camera matrix multiplied by the projection is defined in Eq. (10.27). The affine model implements the projection defined in Eq. (10.43). In the perspective and affine functions, coordinates are transformed to pixels by the transformation defined in Eq. (10.31).

```
%***********************************************
%Obtain the projection matrix parameters
%from the camera position
%-----------------------------------------------
 function M=PerspectiveProjectionMatrix(C);
%-----------------------------------------------

%World to camera
WC=WorldToCamera(C);

%Project point in the image
F=[ C(7)   0      0
      0    C(7)   0
      0     0     1 ];
```

CODE 10.3

Camera models.

```
%Distance units to pixels
K=[ C(10)    0      C(8)
     0     C(11)    C(9)
     0       0       1 ];

%Projection matrix
M=K*F*WC;

%*********************************************
%Obtain the projection matrix parameters
%from the camera position
%---------------------------------------------
 function M=AffineProjectionMatrix(C);
%---------------------------------------------

%world to camera
WC=WorldToCamera(C);
WC(4,1:4)=[0 0 0 1];

%project point in the image
F=[ 1    0     0    0
    0    1     0    0
    0    0     0    1];
%distance units to pixels
K=[ C(10)    0      C(8)
     0     C(11)    C(9)
     0       0       1 ];

%projection matrix
M=K*F*WC;
```

CODE 10.3

(Continued)

Code 10.4 uses the previous functions to generate figures that show the projection of a pair of points in the image plane for the perspective and affine models. The camera is defined with a translation of 1 in y and the focal length is 0.5 from the camera plane. The image is defined to be 100×100 pixels, and the principal point is in the middle of the image, i.e., at pixel of coordinates (50,50). After the definition of the camera, the code defines two 3D points in homogeneous form. These points will be used to explain how camera models map world points into images.

The example first draws a frame to represent the world frame. The parameters are chosen to show the image plane and the world points. These are drawn using the functions `DrawWorldFrame` and `DrawImagePlane` defined in Code 10.1. After the drawing, the code computes the projection matrix by calling the function `PerspectiveProjectionMatrix` as discussed in Code 10.3. This transformation is used to project the 3D points into the image. In the code, the matrix **UV** contains

```
%*********************************************
%Example of the computation projection of points
%for the perspective and affine camera models
%---------------------------------------------
 function ProjectionExample();
%---------------------------------------------

%C=[x0,y0,z0,a,b,g,f,u0,v0,kx,ky]
%
%Camera parameters:
%x0,y0,z0: location
%a,b,g   : orientation
%f       : focal length
%u0,v0   : optical centre
%kx,ky   : pixel size

C=[0,1,0,0,0,0,0.5,50,50,100,100];

%3D points in homogeneous form
XYZ=[ 0,   .2     %x
      1,   .6     %y
     1.7   2      %z
      1    1];

%Perspective example
figure(1);
clf;

%Draw world frame
DrawWorldFrame(-.5,2,-.5,2,-.5,2);

%Draw camera
DrawImagePlane(C,1,1);

%Draw world points
DrawPoints(XYZ,[0,0,0]);

%Perspective projection matrix
P=PerspectiveProjectionMatrix(C);

%Project into camera frame, in pixels
UV=P*XYZ;

%Convert to camera co-ordinates
PC=ImageToCamera(UV,C);

%Convert to world frame
MI=CameraToWorld(C);
PW=MI*PC;

%Draw Projected points in world frame
DrawPoints(PW,[1,0,0]);
```

CODE 10.4

Main example.

```
%Draw projection lines
DrawPerspectiveProjectionLines([C(1),C(2),C(3)],XYZ,[.3,.3,.3]);

%Draw image points
figure(2);
clf;
DrawImagePoints(UV,C,[0,0,0]);

%Affine example
figure(3);
clf;

%Draw world frame
DrawWorldFrame(-.5,2,-.5,2,-.5,2);

%draw camera
DrawImagePlane(C,1,1);

%3D points in homogeneous form
DrawPoints(XYZ,[0,1,0]);

%Affine projection matrix
P=AffineProjectionMatrix(C);

%Project into camera frame, in pixels
UV=P*XYZ;

%Convert to camara co-ordinates
PC=ImageToCamera(UV,C);

%Convert to world frame
PW=MI*PC;

%Draw Projected points in world frame
DrawPoints(PW,[0,0,0]);

%Draw projection lines
DrawAffineProjectionLines([C(1),C(2),C(3)],XYZ,[.3,.3,.3]);

%Draw image points
figure(4);
clf;
DrawImagePoints(UV,C,[0,0,0]);
```

CODE 10.4

(Continued)

the coordinates of the points in pixels. To draw these points in the 3D space, first they are converted to the camera coordinates by calling `ImageToCamera` and then they are converted to the world frame. The function `DrawPerspectiveProjectionLines` draws the lines from the world points to the center of projection.

The result of the perspective projection example is shown in Figure 10.9. Here we can see the projection lines pass through the points obtained by the projection matrix. The image shown in Figure 10.9(b) was obtained by calling the function `DrawImagePoints` defined in Code 10.1. This function draws the points obtained by the projection matrix. One of the points is projected into the center of the image. This is because its x and y coordinates are the same as the principal point.

The last two figures created in Code 10.4 show the projection for the affine matrix. The process is similar to the perspective example, but they use the projection obtained by the function `AffineProjectionMatrix` defined in Code 10.3. The resultant figures are shown in Figure 10.9. Here we can see that the projection matrix transforms the points by following rays perpendicular to the image plane. As such, the points in Figure 10.9(b) are further apart than the points in Figure 10.10(b). In the perspective model, the distance between the points depends on the distance from the image plane, while in the affine model this information is lost.

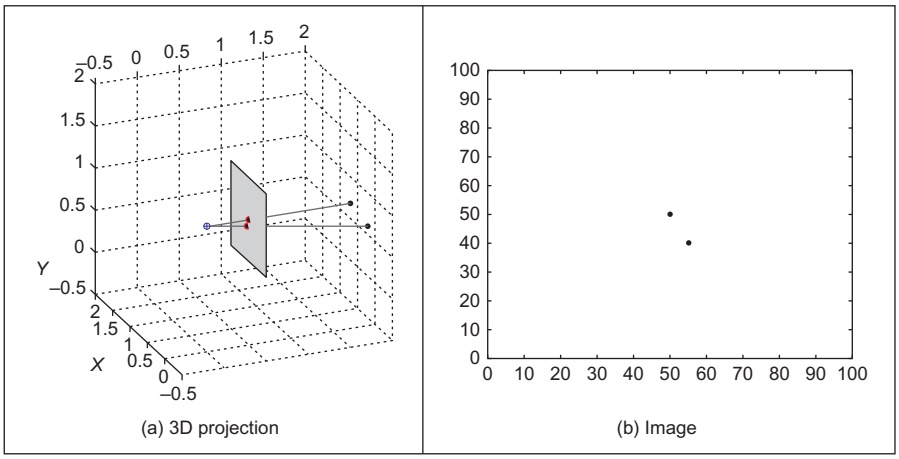

FIGURE 10.9

Perspective camera example.

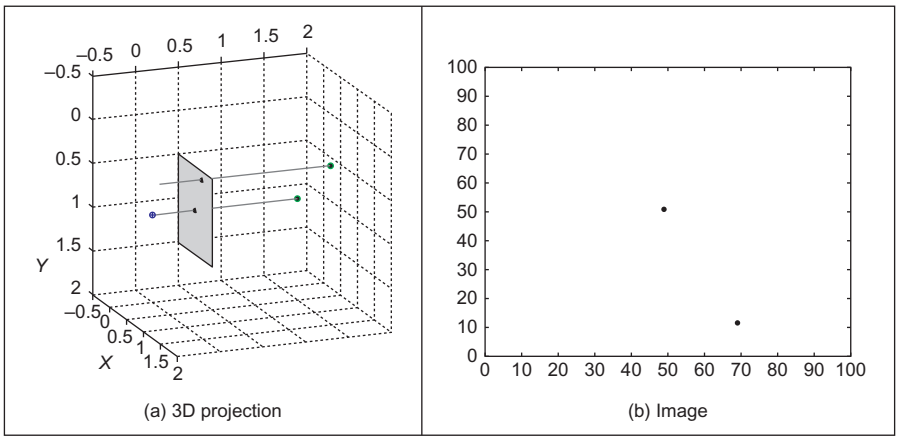

FIGURE 10.10

Affine camera example.

10.7 Discussion

In this appendix, 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 modeled 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 dealt with as noise in computer vision techniques. Strong geometric distortions such as the ones produced by wide-angle or fish-eye lens can be modeled by considering a spherical image plane or by nonlinear projections. 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 of computer vision. The first problem is known as camera calibration and it centers on computing the camera parameters from image data. There are many camera calibration techniques based on the camera model and different types of data. However, camera calibration techniques are grouped in two main classes. Strong camera calibration assumes knowledge of the 3D coordinates of image points. Weak calibration techniques do not know 3D coordinates, 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 centers on recovering the coordinates of points in the 3D scene from image data. There are techniques developed for each camera model.

10.8 References

Aguado, A.S., Montiel, E., Nixon, M.S., 2000. On the intimate relationship between the principle of duality and the Hough transform. Proc. R. Soc. Lond. A 456, 503−526.

Hartley, R., Zisserman, A., 2001. Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge, UK.

Trucco, E., Verri, A., 1998. Introductory Techniques for 3-D Computer Vision. Prentice Hall, Upper Saddle River, NJ.