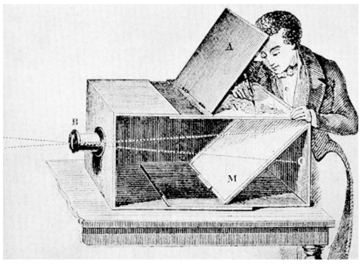
Chapter

11 Image Formation

Everything we see is a perspective,
not the truth.
Marcus Aurelius



In this chapter we discuss how images are formed and captured, the first step in robot and human perception of the world. From images we can deduce the size, shape and position of objects in the world as well as other characteristics such as color and texture which ultimately lead to recognition.

It has long been known that a simple pin-hole is able to create a perfect inverted image on the wall of a darkened room. Some marine mollusks, for example the Nautilus, have pin-hole camera eyes. All vertebrates have a lens that forms an inverted image on the retina where the light-sensitive cells rod and cone cells, shown previously in Fig. 10.6, are arranged. A digital camera is similar in principle – a glass or plastic lens forms an image on the surface of a semiconductor chip with an array of light-sensitive devices to convert light to a digital image.

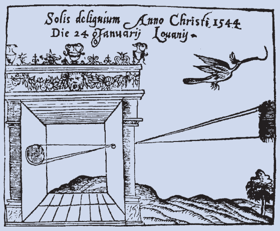
The process of image formation, in an eye or in a camera, involves a *projection* of the 3-dimensional world onto a 2-dimensional surface. The depth information is lost and we can no longer tell from the image whether it is of a large object in the distance or a smaller closer object. This transformation from 3 to 2 dimensions is known as perspective projection and is discussed in Sect. 11.1. Section 11.2 introduces the topic of camera calibration, the estimation of the parameters of the perspective transformation. Sections 11.3 to 11.5 introduce alternative types of cameras capable of wide-angle, panoramic or light-field imaging. Section 11.6 introduces some advanced concepts such as projecting lines and conics, and non-perspective cameras.

11.1 Perspective Camera

11.1.1 Perspective Projection

A small hole in the wall of a darkened room will cast a dim inverted image of the outside world on the opposite wall – a so-called pin-hole camera. The pin-hole camera produces a very dim image since its radiant power is the scene luminance in units of W m^{-2} multiplied by the area of the pin hole. Figure 11.1a shows that only a small fraction of the light leaving the object finds its way to the image. A pin-hole camera has no focus adjustments – all objects are in focus irrespective of distance.

In the 5th century BCE, the philosopher Mozi in ancient China mentioned the effect of an inverted image forming through a pinhole. A camera obscura is a darkened room where a dim inverted image of the world is cast on the wall by light entering through a small hole. They were popular tourist attractions in Victorian times, particularly in Britain, and many are still operating today. (Image on the right from the Drawing with Optical Instruments collection at http://vision.mpiwg-berlin.mpg.de/elib)



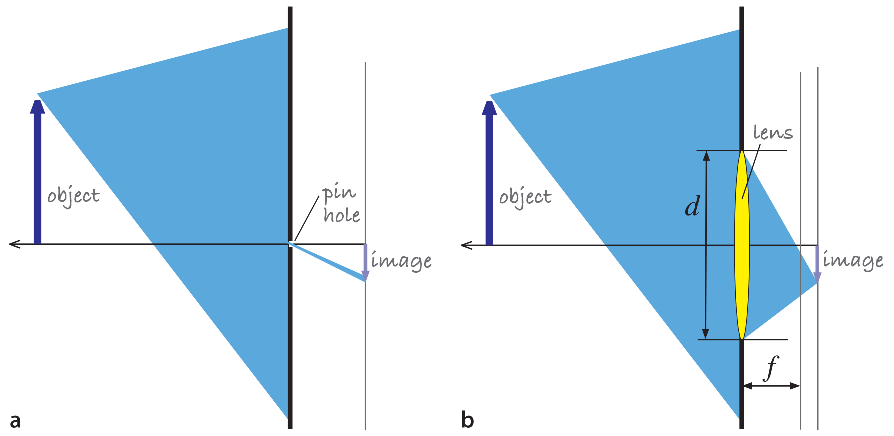


Fig. 11.1.
Light gathering ability of **a** pin-hole camera and **b** a lens

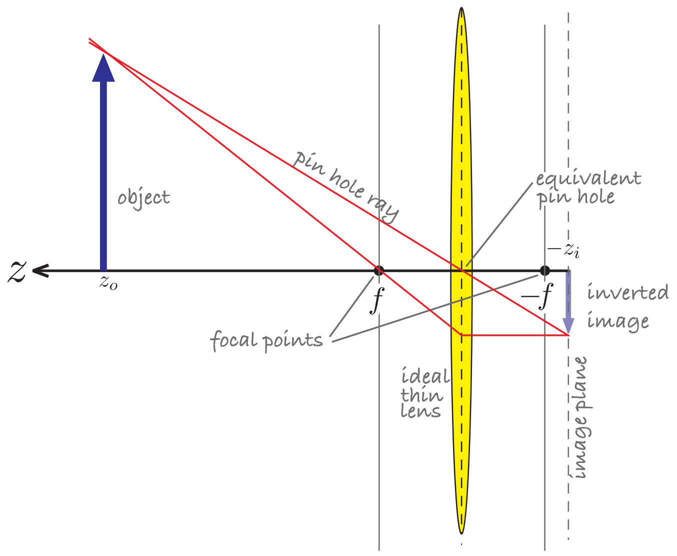


Fig. 11.2.
Image formation geometry for a thin convex lens shown in 2-dimensional cross section. A lens has two focal points at a distance of *f* on each side of the lens. By convention the camera's optical axis is the *z*-axis

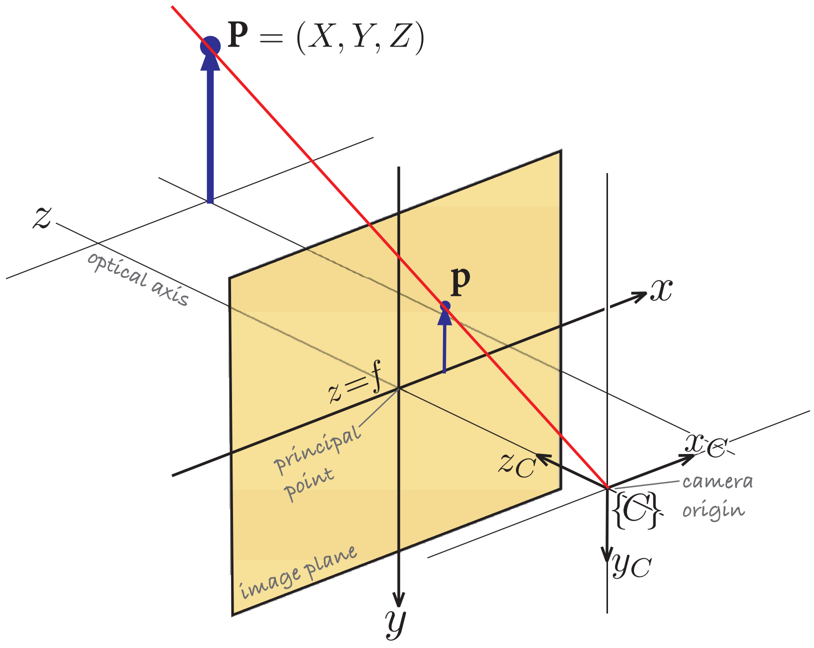


Fig. 11.3.
The central-projection model. The image plane is at a distance *f* in front of the camera's origin, and on which a noninverted image is formed. The camera's coordinate frame is right-handed with the *z*-axis defining the center of the field of view

The key to brighter images is to use an objective lens, as shown in Fig. 11.1b, which collects light from the object over a larger area and directs it to the image. A convex lens can form an image just like a pinhole and the fundamental geometry of image formation for a thin lens◂ is shown in Fig. 11.2. The positive z-axis is the camera's optical axis.

Real camera lenses comprise multiple lens elements but still have focal points on each side of the compound lens assembly.

The z-coordinate of the object and its image, with respect to the lens center, are related by the thin lens equation

$$\frac{1}{z_\mathrm{o}} + \frac{1}{z_\mathrm{i}} = \frac{1}{f} \tag{11.1}$$

where z_o is the distance to the object, z_i the distance to the image, and f is the focal length of the lens.◂ For $z_o > f$ an inverted image is formed on the image plane at $z_i < -f$.

The inverse of focal length is known as diopter. For thin lenses placed close together their combined diopter is close to the sum of their individual diopters.

In a camera the image plane is fixed at the surface of the sensor chip so the focus ring of the camera moves the lens along the optical axis so that it is a distance z_i from the image plane – for an object at infinity $z_i = f$. The downside of using a lens is the need to focus. Our own eye has a single convex lens made from transparent crystallin proteins, and focus is achieved by muscles which change its shape – a process known as accomodation. A high-quality camera lens is a compound lens comprising multiple glass or plastic lenses.

In computer vision it is common to use the central perspective imaging model shown in Fig. 11.3. The rays converge on the origin of the camera frame $\{C\}$ and a noninverted image is *projected* onto the image plane located at $z = f$. The z-axis intersects the image plane at the principal point which is the origin of the 2D image coordinate frame. Using similar triangles we can show that a point at the world coordinates $\mathbf{P} = (X, Y, Z)$ is projected to the image point $\mathbf{p} = (x, y)$ by

$$x = f\frac{X}{Z},\ y = f\frac{Y}{Z} \tag{11.2}$$

which is a projective transformation, or more specifically a perspective projection. This mapping from the 3-dimensional world to a 2-dimensional image has consequences that we can see in Fig. 11.4 – parallel lines converge and circles become ellipses.

More formally we can say that the transformation, from the world to the image plane has the following characteristics:

1. It performs a mapping from 3-dimensional space to the 2-dimensional image plane: $\mathcal{P} : \mathbb{R}^3 \mapsto \mathbb{R}^2$.
2. Straight lines in the world are projected to straight lines on the image plane.
3. Parallel lines in the world are projected to lines that intersect at a vanishing point as shown in Fig. 11.4a. In drawing, this effect is known as foreshortening. The exception are fronto-parallel lines – lines lying in a plane parallel to the image plane – which always remain parallel.

Lens aperture. The *f-number* of a lens, typically marked on the rim, is a dimensionless quantity $F = f/d$ where d is the diameter of the lens (often denoted ϕ on the lens rim). The *f*-number is *inversely* related to the light gathering ability of the lens. To reduce the amount of light falling on the image plane the effective diameter is reduced by a mechanical aperture, or iris, which *increases* the *f*-number. Illuminance on the image plane is inversely proportional to F^2 since it depends on light gathering area. To reduce illuminance by a factor of 2, the *f*-number must be increased by a factor of $\sqrt{2}$ or "one stop". The *f*-number graduations increase by $\sqrt{2}$ at each stop. An *f*-number is conventionally written in the form $f/1.4$ for $F = 1.4$.

Focus and depth of field. Ideally a group of light rays from a point in the scene meet at a point in the image. With imperfect focus the rays instead form a finite sized spot called the circle of confusion which is the point spread function of the optical system. By convention, if the size of the circle is around that of a pixel then the image is *acceptably* focused.

A pin-hole camera has no focus control and always creates a focused image of objects irrespective of their distance. A lens does not have this property – the focus ring changes the distance between the lens and the image plane and must be adjusted so that the object of interest is *acceptably* focused. Photographers refer to depth of field which is the range of object distances for which *acceptably* focused images are formed. Depth of field is high for small aperture settings where the lens is more like a pin-hole, but this means less light and noisier images or longer exposure time and motion blur. This is the photographer's dilemma!

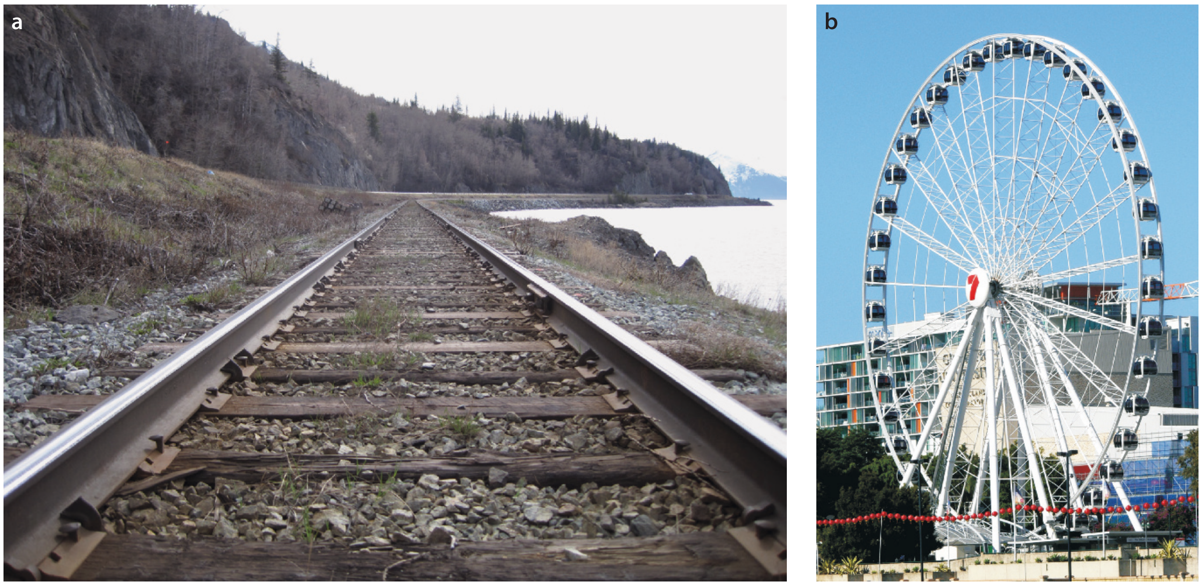


Fig. 11.4. The effect of perspective transformation. **a** Parallel lines converge, **b** circles become ellipses

4. Conics in the world are projected to conics on the image plane. For example, a circle is projected as a circle or an ellipse as shown in Fig. 11.4b.
5. The size (area) of a shape is not preserved and depends on distance.
6. The mapping is not one-to-one and no unique inverse exists. That is, given (x, y) we cannot uniquely determine (X, Y, Z). All that can be said is that the world point lies somewhere along the red projecting ray shown in Fig. 11.3. This is an important topic that we will return to in Chap. 14.
7. The transformation is not conformal – it does not preserve shape since internal angles are not preserved. Translation, rotation and scaling are examples of conformal transformations.

Conic sections, or conics, are a family of curves obtained by the intersection of a plane with a cone. They include circles, ellipses, parabolas and hyperbolas.

11.1.2 Modeling a Perspective Camera

We can write the image-plane point coordinates in homogeneous form $\tilde{\boldsymbol{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$ where

$$\tilde{x} = fX,\ \tilde{y} = fY,\ \tilde{z} = Z$$

or in compact matrix form as

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{11.3}$$

where the nonhomogeneous image-plane coordinates are

$$x = \frac{\tilde{x}}{\tilde{z}},\ y = \frac{\tilde{y}}{\tilde{z}}$$

These are often referred to as the retinal image-plane coordinates. For the case where $f = 1$ the coordinates are referred to as the normalized, retinal or canonical image-plane coordinates.

If we write the world coordinate in homogeneous form as well ${}^C\tilde{\boldsymbol{P}} = (X, Y, Z, 1)^T$ then the perspective projection can be written in *linear* form as

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^C\tilde{\boldsymbol{P}} \tag{11.4}$$

or

$$\tilde{\boldsymbol{p}} = \boldsymbol{C}\, {}^C\tilde{\boldsymbol{P}} \tag{11.5}$$

where $\boldsymbol{C}$ is a 3×4 matrix known as the camera matrix. Note that we have written ${}^C\tilde{\boldsymbol{P}}$ to highlight the fact that this is the coordinate of the point with respect to the camera frame $\{C\}$. The tilde indicates homogeneous quantities and Sect. C.2 provides a refresher on homogeneous coordinates. The camera matrix can be factored

$$\tilde{\boldsymbol{p}} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^C\tilde{\boldsymbol{P}}$$

where the second matrix is the projection matrix.

The Toolbox allows us to create a model of a central-perspective camera. For example

```
>> cam = CentralCamera('focal', 0.015);
```

returns an instance of a `CentralCamera` object with a 15 mm lens. By default the camera is at the origin of the world frame with its optical axis pointing in the world *z*-direction as shown in Fig. 11.3. We define a world point

```
>> P = [0.3, 0.4, 3.0]';
```

in units of meters and the corresponding image-plane coordinates are

```
>> cam.project(P)
ans =
    0.0015
    0.0020
```

The point on the image plane is at (1.5, 2.0) mm with respect to the principal point. This is a very small displacement but it is commensurate with the size of a typical image sensor.

In general the camera will have an arbitrary pose ξ_C with respect to the world coordinate frame as shown in Fig. 11.5. The position of the point with respect to the camera is

$${}^C\boldsymbol{P} = (\ominus\xi_C) \cdot {}^0\boldsymbol{P} \tag{11.6}$$

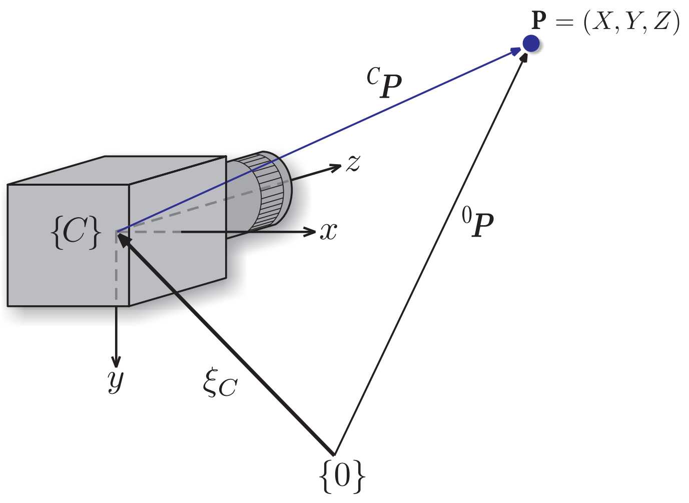


Fig. 11.5.
Camera coordinate frames

or using homogeneous coordinates

$$^{C}\boldsymbol{P} = T_{C}^{-1}\,{}^{0}\boldsymbol{P}$$

We can easily demonstrate this by moving our camera 0.5 m to the *left*

```
>> cam.project(P, 'pose', SE3(-0.5, 0, 0) )
ans =
    0.0040
    0.0020
```

where the third argument is the pose of the camera ξ_C as a homogeneous transformation. We see that the *x*-coordinate has increased from 1.5 mm to 4.0 mm, that is, the image point has moved to the *right*.

11.1.3 Discrete Image Plane

In a digital camera the image plane is a $W \times H$ grid of light-sensitive elements called photosites that correspond directly to the picture elements (or pixels) of the image as shown in Fig. 11.6. The pixel coordinates are a 2-vector (u, v) of nonnegative integers and by convention the origin is at the top-left hand corner of the image plane. In

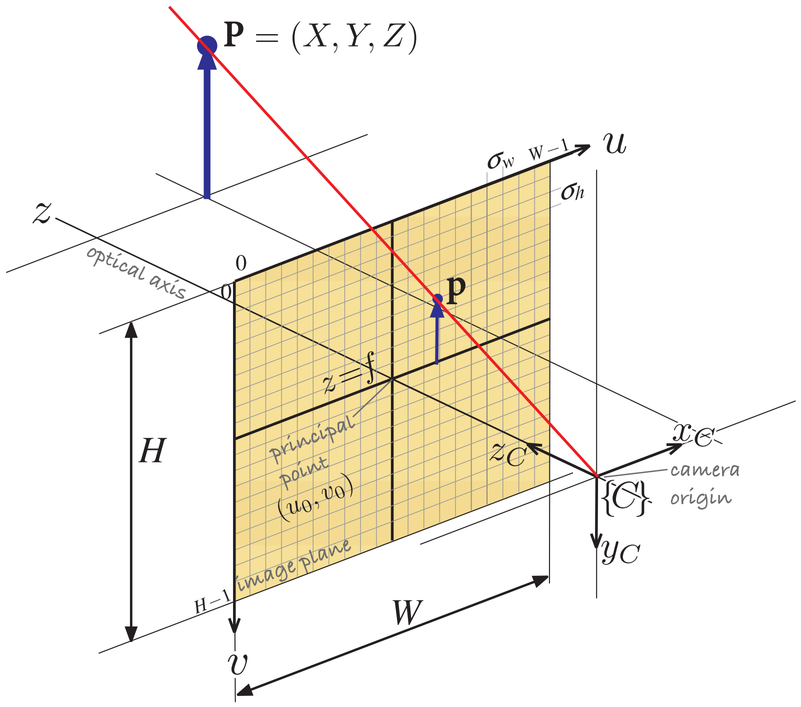


Fig. 11.6. Central projection model showing image plane and discrete pixels

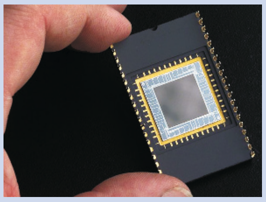


Image sensor. The light-sensitive cells in a camera chip, the photosites (see page 364), are commonly square with a side length in the range 1–10 µm. Professional cameras have large photosites for increased light sensitivity whereas cellphone cameras have small sensors and therefore small less-sensitive photosites. The ratio of the number of horizontal to vertical pixels is the aspect ratio and is commonly 4:3 or 16:9 (see page 366). The dimension of the sensor is measured diagonally across the array and is commonly expressed in inches, e.g. ⅓, ¼ or ½ inch. However the active sensing area of the chip has a diagonal that is typically around ⅔ of the given dimension.

MATLAB® the top-left pixel is (1, 1). The pixels are uniform in size and centered on a regular grid so the pixel coordinate is related to the image-plane coordinate by

$$u = \frac{x}{\rho_w} + u_0,\ v = \frac{y}{\rho_h} + v_0$$

where ρ_w and ρ_h are the width and height of each pixel respectively, and (u_0, v_0) is the principal point – the pixel coordinate of the point where the optical axis intersects the image plane with respect to the new origin. We can write Eq. 11.4 in terms of pixel coordinates by prepending a camera parameter matrix $\boldsymbol{K}$

$$\tilde{\boldsymbol{p}} = \underbrace{\begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\boldsymbol{K}} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^C\tilde{\boldsymbol{P}} \tag{11.7}$$

The matrix $\boldsymbol{K}$ is often written with a finite value at $\boldsymbol{K}_{1,2}$ to represent skew. This accounts for the fact that the u- and v-axes are not orthogonal, which with precise semiconductor fabrication processes is quite unlikely.

where $\tilde{\boldsymbol{p}} = (\tilde{u}, \tilde{v}, \tilde{w})$ is the homogeneous coordinate of the world point **P** in pixel coordinates. The nonhomogeneous image-plane pixel coordinates are

$$u = \frac{\tilde{u}}{\tilde{w}},\ v = \frac{\tilde{v}}{\tilde{w}} \tag{11.8}$$

For example if the pixels are 10 μm square and the pixel array is 1 280 × 1 024 pixels with its principal point at image-plane coordinate (640, 512) then

```
>> cam = CentralCamera('focal', 0.015, 'pixel', 10e-6,↵
 'resolution', [1280 1024], 'centre', [640 512], 'name', 'mycamera')
cam =
 name: mycamera [central-perspective]
  focal length:   0.015
  pixel size:     (1e-05, 1e-05)
  principal pt:   (640, 512)
  number pixels:  1280 x 1024
  pose:           t = (0,0,0), RPY/yxz = (0,0,0) deg
```

which displays the parameters of the camera model including the camera pose `T`. The corresponding nonhomogeneous image-plane coordinates of the previously defined world point are

```
>> cam.project(P)
ans =
   790
   712
```

11.1.4 Camera Matrix

Combining Eq. 11.6 and Eq. 11.7 we can write the camera projection in general form as

$$\begin{aligned} \tilde{\boldsymbol{p}} &= \underbrace{\begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{intrinsic}} \underbrace{\left({}^0\boldsymbol{T}_C\right)^{-1}}_{\text{extrinsic}} \tilde{\boldsymbol{P}} \\ &= \boldsymbol{K}\boldsymbol{P}_0\, {}^0\boldsymbol{T}_C^{-1} \tilde{\boldsymbol{P}} \\ &= \boldsymbol{C}\tilde{\boldsymbol{P}} \end{aligned} \tag{11.9}$$

The terms f/ρ_w and f/ρ_h are the focal length expressed in units of pixels.

where all the terms are rolled up into the camera matrix $\boldsymbol{C}$. This is a 3 × 4 homogeneous transformation which performs scaling, translation and perspective projection. It is often also referred to as the projection matrix or the camera calibration matrix.

We have already mentioned the fundamental ambiguity with perspective projection, that we cannot distinguish between a large distant object and a smaller closer object. We can rewrite Eq. 11.9 as

$$\tilde{\boldsymbol{p}} = (\boldsymbol{C}\boldsymbol{H}^{-1})(\boldsymbol{H}\tilde{\boldsymbol{P}}) = \boldsymbol{C}'\tilde{\boldsymbol{P}}'$$

where $\boldsymbol{H}$ is an arbitrary nonsingular 3×3 matrix. This implies that an infinite number of camera $\boldsymbol{C}'$ and world point coordinate $\tilde{\boldsymbol{P}}'$ combinations will result in the same image-plane projection $\tilde{\boldsymbol{p}}$.

This illustrates the essential difficulty in determining 3-dimensional world coordinates from 2-dimensional projected coordinates. It can only be solved if we have information about the camera or the 3-dimensional object.

The projection can also be written in functional form as

$$\boldsymbol{p} = \mathcal{P}(\boldsymbol{P}, \boldsymbol{K}, \boldsymbol{\xi}_C) \tag{11.10}$$

where $\boldsymbol{P}$ is the point coordinate vector in the world frame. $\boldsymbol{K}$ is the camera parameter matrix and comprises the intrinsic parameters which are the innate characteristics of the camera and sensor such as f, ρ_w, ρ_h, u_0 and v_0. $\boldsymbol{\xi}_C$ is the pose of the camera and comprises a minimum of six parameters – the extrinsic parameters – that describe camera translation and orientation in **SE**(3).

There are 5 intrinsic and 6 extrinsic parameters – a total of 11 independent parameters to describe a camera. The camera matrix has 12 elements so one degree of freedom, the overall scale factor, is unconstrained and can be arbitrarily chosen. In practice the camera parameters are not known and must be estimated using a camera calibration procedure which we will discuss in Sect. 11.2.

The camera intrinsic parameter matrix `K` for this camera is

```
>> cam.K
ans =
   1.0e+03 *
    1.5000         0    0.6400
         0    1.5000    0.5120
         0         0    0.0010
```

The camera matrix is implicitly created when the Toolbox camera object is constructed and for this example is

```
>> cam.C
ans =
   1.0e+03 *
    1.5000         0    0.6400         0
         0    1.5000    0.5120         0
         0         0    0.0010         0
```

The **field of view** of a lens is an open rectangular pyramid, a frustum, that subtends angles θ_h and θ_v in the horizontal and vertical planes respectively. A *normal lens* is one with a field of view around 50°, while a wide angle lens has a field of view >60°. Beyond 110° it is difficult to create a lens that maintains perspective projection, so nonperspective fisheye lenses are required.

For very wide-angle lenses it is more common to describe the field of view as a solid angle which is measured in units of steradians (or sr). This is the area of the field of view projected onto the surface of a unit sphere. A hemispherical field of view is 2π sr and a full spherical view is 4π sr. If we approximate the camera's field of view by a cone with apex angle θ the corresponding solid angle is $2\pi(1 - \cos\theta/2)$ sr. A camera with a field of view greater than a full hemisphere is termed omnidirectional or panoramic.

The camera matrix $\boldsymbol{C} \subset \mathbb{R}^{3\times4}$ has some important structure and properties:

- It can be partitioned $\boldsymbol{C} = (\boldsymbol{M} \mid \boldsymbol{c}_4)$ into a nonsingular matrix $\boldsymbol{M} \subset \mathbb{R}^{3\times3}$ and a vector, where $\boldsymbol{c}_4 = -\boldsymbol{M}\boldsymbol{c}$ and $\boldsymbol{c}$ is the world origin in the camera frame. We can recover this by $\boldsymbol{c} = -\boldsymbol{M}^{-1}\boldsymbol{c}_4$.
- The null space of $\boldsymbol{C}$ is $\tilde{\boldsymbol{c}}$.
- A pixel at coordinate $\boldsymbol{p}$ corresponds to a ray in space parallel to the vector $\boldsymbol{M}^{-1}\tilde{\boldsymbol{p}}$.
- The matrix $\boldsymbol{M} = \boldsymbol{K}\boldsymbol{R}$ is the product of the camera intrinsics and the camera inverse orientation. We can perform an RQ-decomposition of $\boldsymbol{M} = \boldsymbol{R}\boldsymbol{Q}$ where $\boldsymbol{R}$ is an upper-triangular matrix (which is $\boldsymbol{K}$) and an orthogonal matrix $\boldsymbol{Q}$ (which is $\boldsymbol{R}$).◂
- The bottom row of $\boldsymbol{C}$ defines the principal plane, which is parallel to the image plane and contains the camera origin.
- If the rows of $\boldsymbol{M}$ are vectors $\boldsymbol{m}_i$ then
  - $\boldsymbol{m}_3^T$ is a vector normal to the principal plane and parallel to the optical axis and $\boldsymbol{M}\boldsymbol{m}_3^T$ is the principal point in homogeneous form.
  - if the camera has zero skew, that is $K_{1,2} = 0$, then $(\boldsymbol{m}_1 \times \boldsymbol{m}_3) \cdot (\boldsymbol{m}_2 \times \boldsymbol{m}_3) = 0$
  - and, if the camera has square pixels, that is $\rho_u = \rho_v$ then $\|\boldsymbol{m}_1 \times \boldsymbol{m}_3\| = \|\boldsymbol{m}_2 \times \boldsymbol{m}_3\| = f/\rho$

Yes, $\boldsymbol{R}$ has two different meanings here. MATLAB does not provide an RQ-decomposition but it can be determined by transforming the inputs to, and results of, the builtin MATLAB QR-decomposition function `qr`. There are many subtleties in doing this though: negative scale factors in the $\boldsymbol{K}$ matrix or $\det \boldsymbol{R} = -1$, see Hartley and Zisserman (2003), or the implementation of `invC` for details.

The field of view of a camera is a function of its focal length f. A wide-angle lens has a small focal length, a telephoto lens has a large focal length, and a zoom lens has an adjustable focal length. The field of view can be determined from the geometry of Fig. 11.6. In the horizontal direction the half-angle of view is

$$\frac{\theta_h}{2} = \tan^{-1}\frac{W/2\rho_w}{f}$$

where W is the number of pixels in the horizontal direction. We can then write

$$\theta_h = 2\tan^{-1}\frac{W\rho_w}{2f},\ \theta_v = 2\tan^{-1}\frac{H\rho_h}{2f} \tag{11.11}$$

We note that the field of view is also a function of the dimensions of the camera chip which is $W\rho_w \times H\rho_h$. The field of view is computed by the `fov` method of the camera object

```
>> cam.fov() * 180/pi
ans =
   46.2127   37.6930
```

in degrees in the horizontal and vertical directions respectively.

11.1.5 Projecting Points

The `CentralCamera` class is a subclass of the `Camera` class and inherits the ability to project multiple points or lines. Using the Toolbox we create a 3×3 grid of points in the xy-plane with overall side length 0.2 m and centered at (0, 0, 1)

```
>> P = mkgrid(3, 0.2, 'pose', SE3(0, 0, 1.0));
```

which returns a 3×9 matrix with one column per grid point where each column comprises the coordinates in X, Y, Z order. The first four columns are

```
>> P(:,1:4)
ans =
   -0.1000   -0.1000   -0.1000         0
   -0.1000         0    0.1000   -0.1000
    1.0000    1.0000    1.0000    1.0000
```

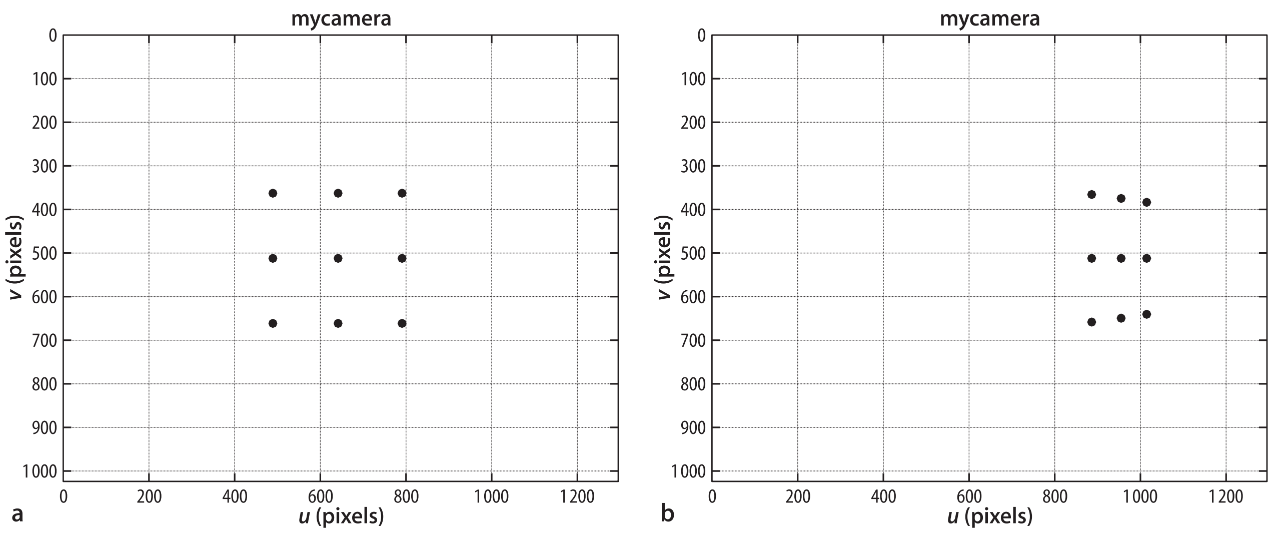


Fig. 11.7. Two views of a planar grid of points. **a** Frontal view, **b** oblique view

By default `mkgrid` generates a grid in the *xy*-plane that is centered at the origin. The optional last argument is a homogeneous transformation that is applied to the default points and allows the plane to be arbitrarily positioned and oriented.

The image-plane coordinates of the vertices are

```
>> cam.project(P)
ans =
   490   490   490   640   640   640   790   790   790
   362   512   662   362   512   662   362   512   662
```

which can also be plotted

```
>> cam.plot(P)
```

giving the virtual camera view shown in Fig. 11.7a. The camera pose

```
>> Tcam = SE3(-1,0,0.5)*SE3.Ry(0.9);
```

results in an oblique view of the plane

```
>> cam.plot(P, 'pose', Tcam)
```

shown in Fig. 11.7b. We can clearly see the effect of perspective projection which has distorted the shape of the square – the top and bottom edges, which are parallel lines, have been projected to lines that converge at a vanishing point.

The vanishing point for a line can be determined from the projection of its ideal line. The top and bottom lines of the grid are parallel to the world *x*-axis or the vector (1, 0, 0). The corresponding ideal line has homogeneous coordinates (1, 0, 0, 0) and exists at infinity due to the final zero element. The vanishing point is the projection of this vector

```
>> cam.project([1 0 0 0]', 'pose', Tcam)
ans =
   1.0e+03 *
    1.8303
    0.5120
```

which is (1 803, 512) and just to the right of the visible image plane.

The `plot` method can optionally return the image-plane coordinates

```
>> p = cam.plot(P, 'pose', Tcam)
```

just like the `project` method. For the oblique viewing case the image-plane coordinates

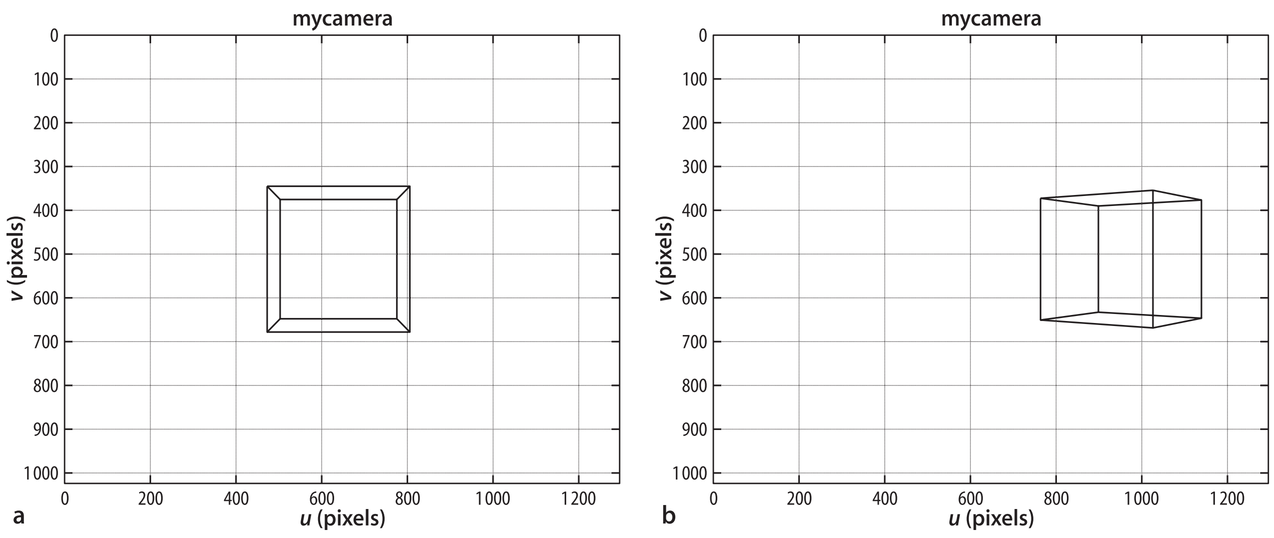


Fig. 11.8. Line segment representation of a cube. **a** Frontal view, **b** oblique view

```
>> p(:,1:4)
ans =
  887.7638  887.7638  887.7638  955.2451
  364.3330  512.0000  659.6670  374.9050
```

have a fractional component which means that the point is not projected to the center of the pixel. However a pixel responds to light equally over its surface area so the discrete pixel coordinate can be obtained by rounding.

This is not strictly true for CMOS sensors where transistors reduce the light-sensitive area by the fill factor – the fraction of each photosite's area that is light sensitive.

A 3-dimensional object, a cube, can be defined and projected in a similar fashion. The vertices of a cube with side length 0.2 m and centered at (0, 0, 1) can be defined by

```
>> cube = mkcube(0.2, 'pose', SE3(0, 0, 1) );
```

which returns a 3×8 matrix with one column per vertex. The image-plane points can be plotted as before by

```
>> cam.plot(cube);
```

Alternatively we can create an *edge* representation of the cube by

```
>> [X,Y,Z] = mkcube(0.2, 'pose', SE3(0, 0, 1), 'edge');
```

and display it

```
>> cam.mesh(X, Y, Z)
```

as shown in Fig. 11.8 along with an oblique view generated by

```
>> Tcam = SE3(-1,0,0.5)*SE3.Ry(0.8);
>> cam.mesh(X, Y, Z, 'pose', Tcam);
```

The edges are in the same 3-dimensional mesh format as generated by MATLAB built-in functions such as `sphere`, `ellipsoid` and `cylinder`.

The elements of the mesh (i, j) have coordinates $(X_{i,j}, Y_{i,j}, Z_{i,j})$.

Successive calls to `plot` will redraw the points or line segments and provides a simple method of animation. The short piece of code

```
theta = [0:500]/100*2*pi;
[X,Y,Z] = mkcube(0.2, [], 'edges');
for th=theta
   T_cube = SE3(0, 0, 1.5)*SE3.rpy(th*[1.1 1.2 1.3])
   cam.mesh( X, Y, Z, 'objpose', T_cube ); drawnow
end
```

shows a cube tumbling in space. The cube is defined with its center at the origin and its vertices are transformed at each time step.

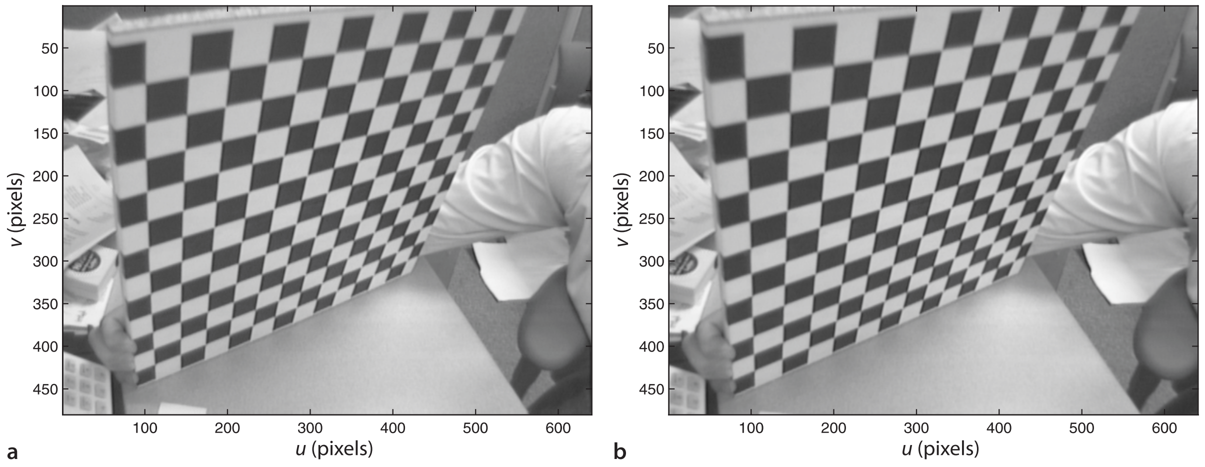


Fig. 11.9. Lens distortion. **a** Distorted image, the curvature of the top row of the squares is quite pronounced, **b** undistorted image. This is calibration image #19 from Bouguet's Camera Calibration Toolbox (Bouguet 2010)

11.1.6 Lens Distortion

No lenses are perfect and the low-cost lenses used in many webcams are far from perfect. Lens imperfections result in a variety of distortions including chromatic aberration (color fringing), spherical aberration or astigmatism (variation in focus across the scene), and geometric distortions where points on the image plane are displaced from where they should be according to Eq. 11.3.

Geometric distortion is generally the most problematic effect that we encounter for robotic applications, and comprises two components: radial and tangential. Radial distortion causes image points to be translated along radial lines from the principal point. The radial error is well approximated by a polynomial

$$\delta r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \cdots \tag{11.12}$$

where r is the distance of the image point from the principal point. Barrel distortion occurs when magnification decreases with distance from the principal point which causes straight lines near the edge of the image to curve outward. Pincushion distortion occurs when magnification increases with distance from the principal point and causes straight lines near the edge of the image to curve inward. Tangential distortion, or decentering distortion, occurs at right angles to the radii but is generally less significant than radial distortion. Examples of a distorted and undistorted image are shown in Fig. 11.9.

The coordinate of the point (u, v) after distortion is given by

$$u^d = u + \delta_u,\ v^d = v + \delta_v \tag{11.13}$$

where the displacement is

$$\begin{pmatrix}\delta_u \\ \delta_v\end{pmatrix} = \underbrace{\begin{pmatrix} u\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots\right) \\ v\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots\right)\end{pmatrix}}_{\text{radial}} + \underbrace{\begin{pmatrix} 2p_1 uv + p_2\left(r^2 + 2u^2\right) \\ p_1\left(r^2 + 2v^2\right) + 2p_2 uv\end{pmatrix}}_{\text{tangential}} \tag{11.14}$$

This displacement vector can be plotted for different values of (u, v) as shown in Fig. 11.13b. The vectors indicate the displacement required to *correct* the distortion at different points in the image, in fact $(-\delta_u, -\delta_v)$, and shows dominant radial distortion.

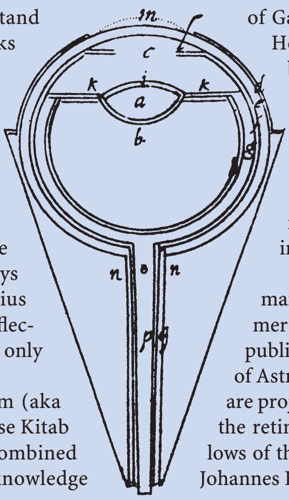
In practice three coefficients are sufficient to describe the radial distortion and the distortion model is parameterized by $(k_1, k_2, k_3, p_1, p_2)$ which are considered as addi-

It has taken humankind a long time to understand light, color and human vision. The Ancient greeks had two schools of thought. The emission theory, supported by Euclid and Ptolemy, held that sight worked by the eye emitting rays of light that interacted with the world somewhat like the sense of touch. The intromission theory, supported by Aristotle and his followers, had physical forms entering the eye from the object.

Euclid of Alexandria (325–265) arguably got the geometry of image formation correct, but his rays emananted from the eye, not the object. Claudius Ptolemy (100–170) wrote Optics and discussed reflection, refraction, and color but today there remains only a poor Arabic translation of his work.

The Arab philosopher Hasan Ibn al-Haytham (aka Alhazen, 965–1040) wrote a seven-volume treatise Kitab al-Manazir (Book of Optics) around 1020. He combined the mathematical rays of Euclid, the medical knowledge of Galen, and the intromission theories of Aristotle. He wrote that "from each point of every colored body, illuminated by any light, issue light and color along every straight line that can be drawn from that point". He understood refraction but believed the eye's lens, not the retina, received the image – like many early thinkers he struggled with the idea of an inverted image on the retina. A Latin translation of his work was a great influence on later European scholars.

It was not until 1604 that geometric optics and human vision came together when the German astronomer and mathematician Johannes Kepler (1571–1630) published Astronomiae Pars Optica (The Optical Part of Astronomy). He was the first to recognize that images are projected inverted and reversed by the eye's lens onto the retina – the image being corrected later "in the hollows of the brain". (Image from Astronomiae Pars Optica, Johannes Kepler, 1604)



tional intrinsic parameters. Distortion can be modeled by the `CentralCamera` class using the `'distortion'` option, for example

```
>> cam = CentralCamera('focal', 0.015, 'pixel', 10e-6, ...
    'resolution', [1280 1024], 'centre', [512 512], ...
    'distortion', [k1 k2 k3 p1 p2] )
```

11.2 Camera Calibration

The camera projection model Eq. 11.9 has a number of parameters that in practice are unknown. In general the principal point is *not* at the center of the photosite array. The focal length of a lens is only accurate◂ to 4% of what it purports to be, and is only correct if the lens is focused at infinity. It is also common experience that the intrinsic parameters change if a lens is detached and reattached, or adjusted for focus or aperture.◂ The only intrinsic parameters that it may be possible to obtain are the photosite dimensions ρ_w and ρ_h from the sensor manufacturer's data sheet. The extrinsic parameters, the camera's pose, raises the question of where exactly is the center point of the camera.

According to ANSI Standard PH3.13-1958 "Focal Length Marking of Lenses".

Changing focus shifts the lens along the optical axis. In some designs, changing focus rotates the lens so if it is not perfectly symmetric this will move the distortions with respect to the image plane. Changing the aperture alters the parts of the lens that light rays pass through and hence the distortion that they incur.

Camera calibration is the process of determining the camera's intrinsic parameters and the extrinsic parameters with respect to the world coordinate system. Calibration techniques rely on sets of world points whose relative coordinates are known and whose corresponding image-plane coordinates are also known. State-of-the-art techniques such as Bouguet's Calibration Toolbox for MATLAB (Bouguet 2010) simply require a number of images of a planar chessboard target such as shown in Fig. 11.12. From this, as discussed in Sect. 11.2.4, the intrinsic parameters (including distortion parameters) can be estimated as well as the relative pose of the chessboard in each image. Classical calibration techniques require a single view of a 3-dimensional calibration target but are unable to estimate the distortion model. These methods are however easy to understand and they start our discussion in the next section.

11.2.1 Homogeneous Transformation Approach

The homogeneous transform method allows direct estimation of the camera matrix $\boldsymbol{C}$ in Eq. 11.9. The elements of this matrix are functions of the intrinsic and extrinsic parameters. Setting $\tilde{\boldsymbol{p}} = (u, v, 1)$, expanding equation Eq. 11.9 and substituting into Eq. 11.8 we can write

$$\begin{aligned} &C_{11}X + C_{12}Y + C_{13}Z + C_{14} - C_{31}uX - C_{32}uY - C_{33}uZ - C_{34}u = 0 \\ &C_{21}X + C_{22}Y + C_{23}Z + C_{24} - C_{31}vX - C_{32}vY - C_{33}vZ - C_{34}v = 0 \end{aligned} \tag{11.15}$$

where (u, v) are the pixel coordinates corresponding to the world point (X, Y, Z) and C_{ij} are elements of the unknown camera matrix.

Calibration requires a 3-dimensional target such as shown in Fig. 11.10. The position of the center of each marker (X_i, Y_i, Z_i), $i \in [1, N]$ with respect to the target frame $\{T\}$ must be known, but $\{T\}$ itself is not known. An image is captured and the *corresponding* image-plane coordinates (u_i, v_i) are determined. Assuming that $C_{34} = 1$ we stack the two equations of Eq. 11.15 for each of the N markers to form the matrix equation

$$\begin{pmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 \\ & & & & & \vdots & & & & & \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -u_NX_N & -u_NY_N & -u_NZ_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -v_NX_N & -v_NY_N & -v_NZ_N \end{pmatrix} \begin{pmatrix} C_{11} \\ C_{12} \\ \vdots \\ C_{33} \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ \vdots \\ u_N \\ v_N \end{pmatrix} \tag{11.16}$$

which can be solved for the camera matrix elements $C_{11} \cdots C_{33}$. Equation 11.16 has 11 unknowns and for solution requires that $N \geq 6$. Often more than 6 points will be used leading to an over-determined set of equations which is solved using least squares.

If the points are coplanar then the left-hand matrix of Eq. 11.16 becomes rank deficient. This is why the calibration target must be 3-dimensional, typically an array of dots or squares on two or three planes as shown in Fig. 11.10.

We will illustrate this with an example. The calibration target is a cube, the markers are its vertices and its coordinate frame $\{T\}$ is parallel to the cube faces with its origin at the center of the cube. The coordinates of the markers with respect to $\{T\}$ are

```
>> P = mkcube(0.2);
```

Now the calibration target is at some "unknown pose" ${}^{C}\xi_T$ with respect to the camera which we choose to be

```
>> T_unknown = SE3(0.1, 0.2, 1.5) * SE3.rpy(0.1, 0.2, 0.3);
```

Next we create a perspective camera whose parameters we will attempt to estimate

```
>> cam = CentralCamera('focal', 0.015, ...
     'pixel', 10e-6, 'resolution', [1280 1024], 'noise', 0.05);
```

We have also specified that zero-mean Gaussian noise with $\sigma = 0.05$ is added to the (u, v) coordinates to model camera noise and errors in the computer vision algorithms. The image-plane coordinates of the calibration target at its "unknown" pose are

```
>> p = cam.project(P, 'objpose', T_unknown);
```

Now using just the object model `P` and the observed image features `p` we estimate the camera matrix

```
>> C = camcald(P, p)
maxm residual 0.066733 pixels.
C =
  853.0895 -236.9378  634.2785  740.0438
  222.6439  986.6900  295.7327  712.0152
   -0.1304    0.0610    0.6495    1.0000
```

The maximum residual in this case is less than 0.1 pixel, that is, the worst error between the projection of a world point using the camera matrix $\boldsymbol{C}$ and the actual image-plane location is very small.

Where is the camera's center? A compound lens has many cardinal points including focal points, nodal points, principal points and planes, entry and exit pupils. The entrance pupil is a point on the optical axis of a compound lens system that is its center of perspective or its *no-parallax point*. We could consider it to be the *virtual pinhole*. Rotating the camera and lens about this point will not change the relative geometry of targets at different distances in the perspective image.

Rotating about the entrance pupil is important in panoramic photography to avoid parallax errors in the final, stitched panorama. A number of web pages are devoted to discussion of techniques for determining the position of this point. Some sites even tabulate the position of the entrance pupil for popular lenses. Much of this online literature refers to this point incorrectly as the *nodal point* even though the techniques given do identify the *entrance pupil*.

Depending on the lens design, the entrance pupil may be behind, within or in front of the lens system.

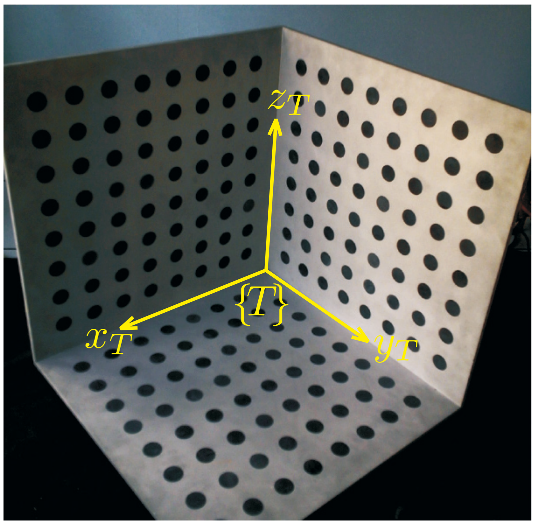


Fig. 11.10. A 3D calibration target showing its coordinate frame $\{T\}$. The centroids of the circles are taken as the calibration points. Note that the calibration circles are situated on three planes (photo courtesy of Fabien Spindler)

Linear techniques such as this cannot estimate lens distortion parameters. The distortion will introduce errors into the camera matrix elements but for many situations this might be acceptably low. Distortion parameters are often estimated using a non-linear optimization over all parameters, typically 16 or more, with the linear solution used as the initial parameter estimate.

11.2.2 Decomposing the Camera Calibration Matrix

The elements of the camera matrix are functions of the intrinsic and extrinsic parameters. However given a camera matrix most of the parameter values can be recovered.

The null space of $\boldsymbol{C}$ is the world origin in the camera frame. Using data from the example above this is

```
>> null(C)'
ans =
    0.0809   -0.1709   -0.8138    0.5495
```

which is expressed in homogeneous coordinates that we can convert to Cartesian form

```
>> h2e(ans)'
ans =
    0.1472   -0.3110   -1.4809
```

which is close to the true value

```
>> T_unknown.inv.t'
ans =
    0.1464   -0.3105   -1.4772
```

To recover orientation as well as the intrinsic parameters we can *decompose* the previously estimated camera matrix

```
>> est = invcamcal(C)
est =
name: invcamcal [central-perspective]
  focal length:   1504
  pixel size:     (1, 0.9985)
  principal pt:   (646.8, 504.4)
  pose:           t = (0.147, -0.311, -1.48), RPY/zyx↵
   = (-1.87, -12.4, -16.4) deg
```

which returns a `CentralCamera` object with its parameters set to values that result in the same camera matrix. We note immediately that the focal length is very large compared to the true focal length of our lens which was 0.015 m, and that the pixel sizes are very large. From Eq. 11.9 we see that focal length and pixel dimensions always appear together as factors f/ρ_w and f/ρ_h. The function invcamcal has set $\rho_w = 1$ but the ratios of the estimated parameters

These quantities have units of pixels since ρ has units of m pixel^{-1}. It is quite common in the literature to consider $\rho = 1$ and the focal length is given in pixels. If the pixels are not square then different focal lengths f_u and f_v must be used for the horizontal and vertical directions respectively.

```
>> est.f/est.rho(1)
ans =
   1.5044e+03
```

are very close to the ratio for the true parameters of the camera

```
>> cam.f/cam.rho(2)
ans =
   1.500e+03
```

The small error in the estimated parameter values is due to the noisy image-plane coordinate values that we used in the calibration process.

The pose of the estimated camera is with respect to the calibration target $\{T\}$ and is therefore ${}^T\hat{\xi}_C$. The true pose of the target with respect to the camera is ${}^C\xi_T$. If our estimation is accurate then ${}^C\xi_T \oplus {}^T\hat{\xi}_C$ will be 0. We earlier set the variable T_unknown equal to ${}^C\xi_T$ and for our example we find that

```
>> trprint(T_unknown*est.T)
t = (4.13e-05, -4.4e-05, -0.00386),↵
 RPY/zyx = (0.296, 0.253, -0.00557) deg
```

which is the relative pose between the true and estimated camera pose. The camera pose is estimated to better than 5 mm in position and a fraction of a degree in orientation.

We can plot the calibration markers as small red spheres

```
>> hold on; plot_sphere(P, 0.03, 'r')
>> trplot(eye(4,4), 'frame', 'T', 'color', 'b', 'length', 0.3)
```

as well as $\{T\}$ which we have set at the world origin. The estimated pose of the camera can be superimposed

```
>> est.plot_camera()
```

and the result is shown in Fig. 11.11. The problem of determining the pose of a camera with respect to a calibration object is an important problem in photogrammetry known as the camera location determination problem.

The option 'frustum' shows the camera as a rectangular pyramid, such as shown in Fig. 11.13a, rather than a camera icon.

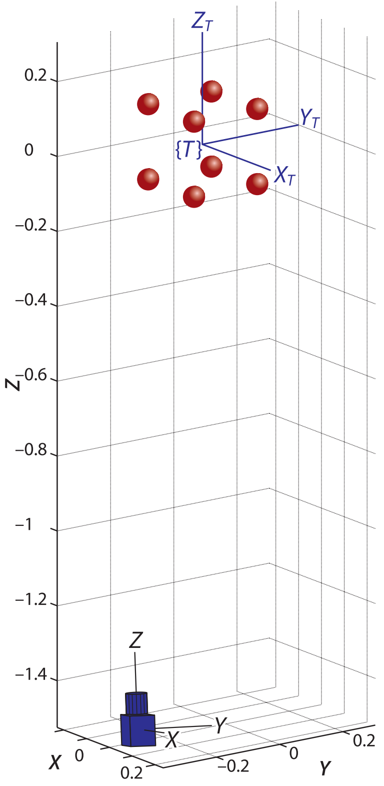


Fig. 11.11. Calibration target points and estimated camera pose with respect to the target frame $\{T\}$ which is assumed to be at the origin

11.2.3 Pose Estimation

The pose estimation problem is to determine the pose ${}^C\xi_T$ of a target's coordinate frame $\{T\}$ with respect to the camera. The geometry of the target is known, that is, we know the position of a number of points (X_i, Y_i, Z_i), $i \in [1, N]$ on the target with respect to $\{T\}$. The camera's intrinsic parameters are also known. An image is captured and the *corresponding* image-plane coordinates (u_i, v_i) are determined using computer vision algorithms.

Estimating the pose using (u_i, v_i), (X_i, Y_i, Z_i) and camera intrinsic parameters is known as the Perspective-n-Point problem or PnP for short. It is a simpler problem than camera calibration and decomposition because there are fewer parameters to estimate. To illustrate pose estimation we will create a calibrated camera with known parameters

```
>> cam = CentralCamera('focal', 0.015, 'pixel', 10e-6, ...
    'resolution', [1280 1024], 'centre', [640 512]);
```

The object whose pose we wish to determine is a cube with side lengths of 0.2 m and the coordinates of the markers with respect to $\{T\}$ are

```
>> P = mkcube(0.2);
```

which we can consider a simple geometric model of the object. The object is at some arbitrary but unknown pose ${}^{C}\xi_{T}$ pose with respect to the camera

```
>> T_unknown = SE3(0.1, 0.2, 1.5) * SE3.rpy(0.1, 0.2, 0.3);
>> T_unknown.print
t = (0.1, 0.2, 1.5), RPY/zyx = (5.73, 11.5, 17.2) deg
```

The image-plane coordinates of the object's points at its unknown pose are

```
>> p = cam.project(P, 'objpose', T_unknown);
```

Now using just the object model `P`, the observed image features `p` and the calibrated camera `cam` we estimate the relative pose ${}^{C}\xi_{T}$ of the object

```
>> T_est = cam.estpose(P, p).print
t = (0.1, 0.2, 1.5), RPY/zyx = (5.73, 11.5, 17.2) deg
```

which is the same (to four decimal places) as the unknown pose `T_unknown` of the object.

In reality the image features coordinates will be imperfectly estimated by the vision system and we would model this by adding zero-mean Gaussian noise to the image feature coordinates as we did in the camera calibration example.

11.2.4 Camera Calibration Toolbox

A popular and practical tool for calibrating cameras using a planar chessboard target is the Camera Calibration Toolbox. A number of images, typically twenty, are taken of the target at different distances and orientations as shown in Fig. 11.12.

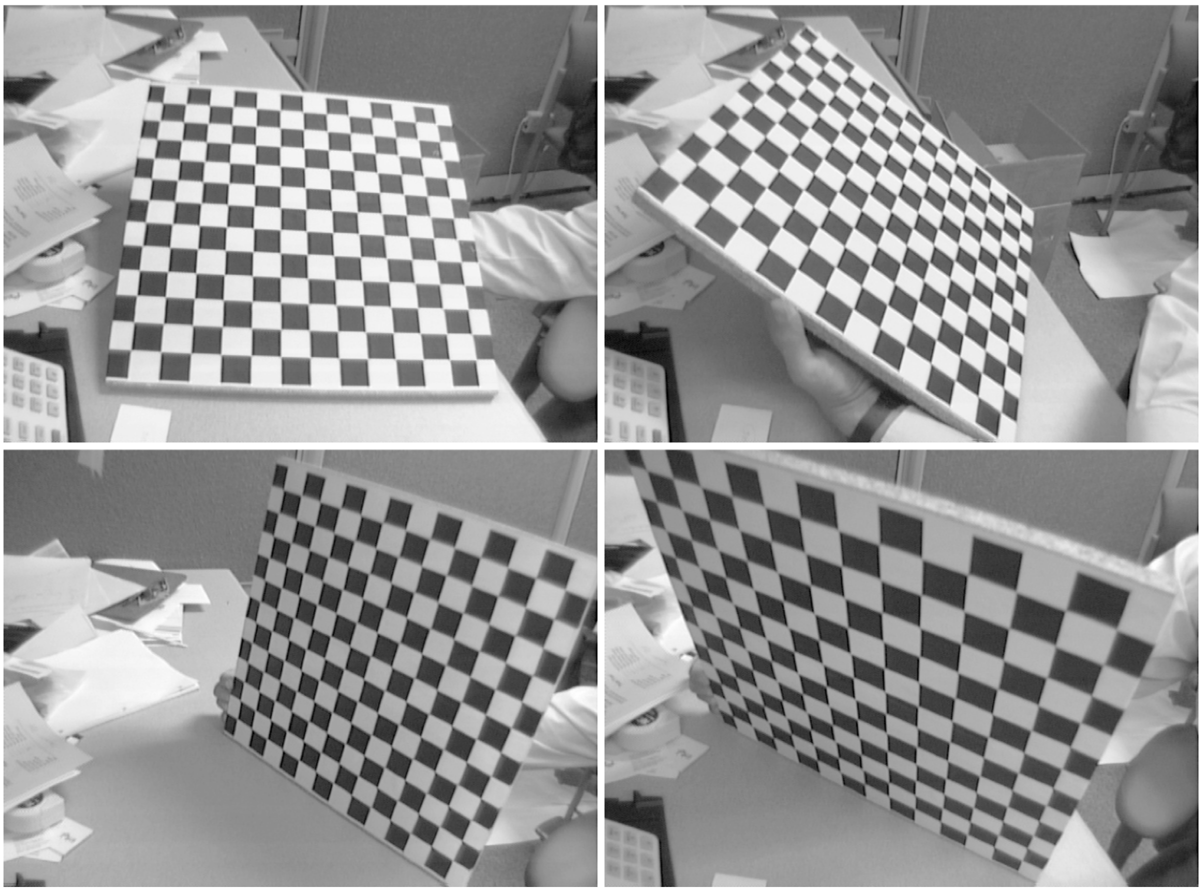


Fig. 11.12. Example frames from Bouguet's Calibration Toolbox showing the calibration target in many different orientations. These are images 2, 5, 9, 18 from the Calibration Toolbox example

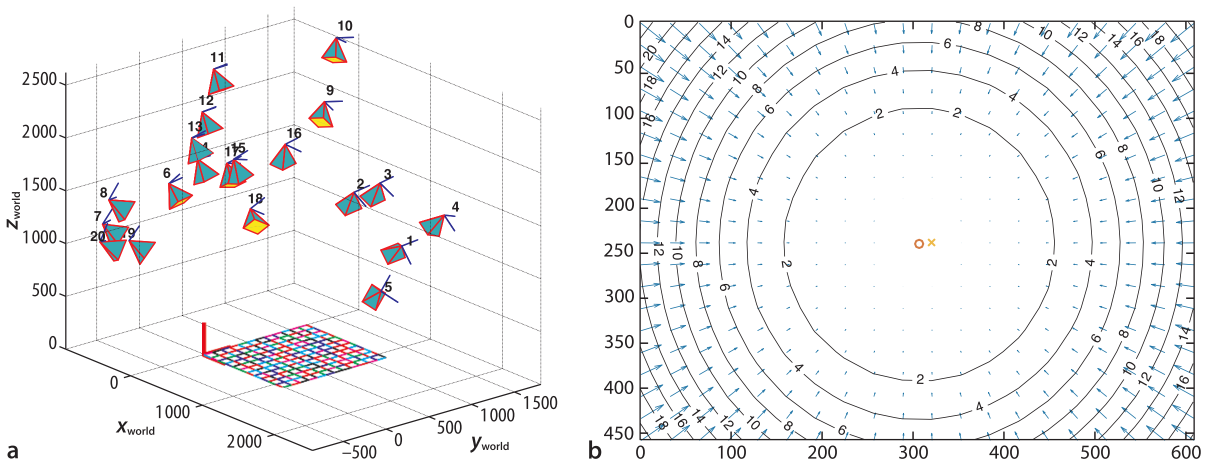


Fig. 11.13. Calibration results from the example in Bouguet's Calibration Toolbox. **a** The estimated camera pose relative to the target for each calibration image, **b** the distortion map with vectors showing how points in the image will move due to distortion

The calibration tool is launched by

```
>> calib_gui
```

and a graphical user interface (GUI) is displayed. The first step is to load the images using the Image Names button. The second step is the Extract Grid Corners button which prompts you to pick the corners of the calibration target in each of the images. This is a little tedious but needs to be done carefully. The final step, the Calibration button, uses the calibration target information to estimate the camera parameter values

The GUI is optional, and the Toolbox functions can be called from inside your own programs. The function `calib_gui_normal` shows the mapping from GUI button names to Calibration Toolbox function names. Note that most of the functions are actually scripts and program state variables are kept in the workspace.

```
Focal Length:          fc = [ 657.39071   657.74678 ]↵
  ± [ 0.37195   0.39793 ]
Principal point:       cc = [ 303.22367   242.74729 ]↵
 ± [ 0.75632   0.69189 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]↵
   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25541   0.12617   -0.00015   0.00006↵
   0.00000 ] ± [ 0.00290   0.01154   0.00016   0.00015   0.00000 ]
Pixel error:          err = [ 0.13355   0.13727 ]
```

For each parameter the uncertainty (3σ bounds) is estimated and displayed.

The camera pose relative to the target is estimated for each calibration image and can be displayed using the Show Extrinsic button. This target-centric view is shown in Fig. 11.13a indicates the estimated relative pose of the camera for each input image.

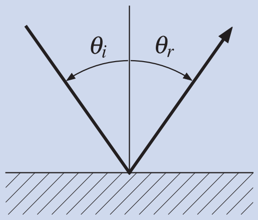
The distortion vector `kc` contains the parameters in the order $(k_1, k_2, p_1, p_2, k_3)$ – note that k_3 is out of sequence. The distortion map can be displayed by

```
>> visualize_distortions
```

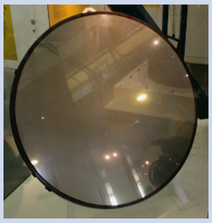
and is shown in Fig. 11.13b. This indicates the displacement from true to distorted image-plane coordinates which in this case is predominately in the radial direction. This is consistent with k_1 and k_2 being orders of magnitude greater than p_1 and p_2 which is typical for most lenses. The Undistort Image button can be used to undistort a set of images and a distorted and undistorted image are compared in Fig. 11.9b. The details of this transformation using image warping will be discussed in Sect. 12.7.4.

11.3 Wide Field-of-View Imaging

We have discussed perspective imaging in quite some detail since it is the model of our own eyes and almost all cameras that we encounter. However perspective imaging constrains us to a fundamentally limited field of view. The thin lens equation (11.1) is singular for points with $Z = f$ which limits the field of view to at most one hemisphere – real lenses



Specular reflection occurs with a mirror-like surface. Incoming rays are reflected such that the angle of incidence equals the angle of reflection or $\theta_r = \theta_i$. Speculum is Latin for mirror and speculum metal (⅔ copper, ⅓ tin) is an alloy that can be highly polished. It was used by Newton and Herschel for the curved mirrors in their reflecting telescopes. See also *Lambertian reflection* on page 309. (The 48 inch speculum mirror from Herschel's 40 foot telescope, completed in 1789, is now in the British Science Museum, photo by Mike Peel (mikepeel.net) licensed under CC-BY-SA)



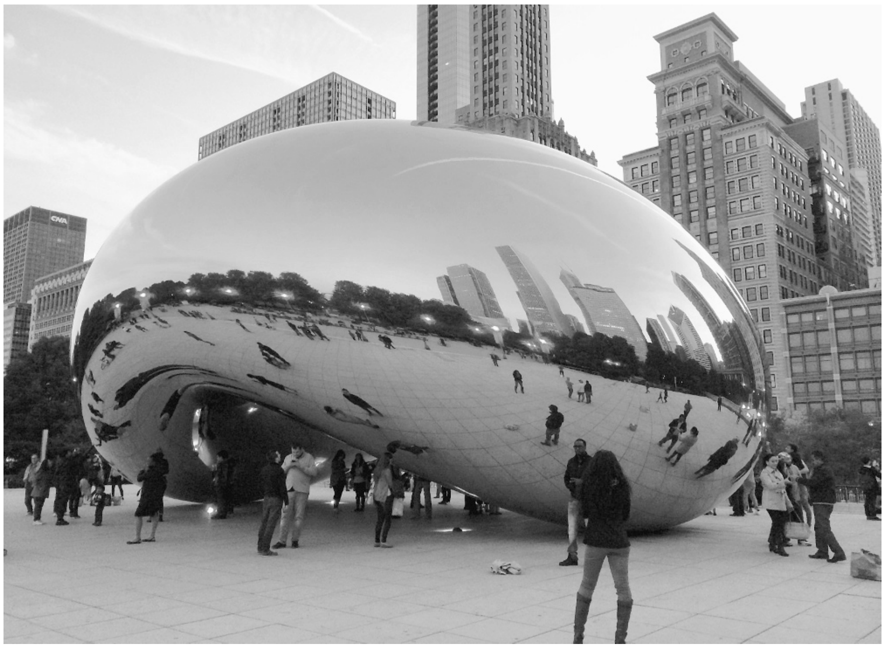


Fig. 11.14. Images formation by reflection from a curved surface (*Cloud Gate*, Chicago, Anish Kapoor, 2006). Note that straight lines have become curves

achieve far less. As the focal length decreases radial distortion is increasingly difficult to eliminate and eventually a limit is reached beyond which lenses cannot practically be built. The only way forward is to drop the constraint of perspective imaging. In Sect. 11.3.1 we describe the geometry of image formation with wide-angle lens systems.

An alternative to refractive optics is to use a reflective surface to form the image such as shown in Fig. 11.14. Newtonian telescopes are based on reflection from concave mirrors rather than refraction by lenses. Mirrors are free of color fringing and are easier to scale to larger sizes than a lens. Nature has also evolved reflective optics – the spookfish and some scallops (see page 285) have eyes based on reflectors formed from guanine crystals. In Sect. 11.3.2 we describe the geometry of image formation with a combination of lenses and mirrors.

The cost of cameras is decreasing so an alternative approach is to combine the output of multiple cameras into a single image, and this is briefly described in Sect. 11.5.1.

11.3.1 Fisheye Lens Camera

A fisheye lens image in shown in Fig. 11.17, and we create a model using the notation shown in Fig. 11.15 where the camera is positioned at the origin **O** and its optical axis is the z-axis. The world point **P** is represented in spherical coordinates (R, θ, ϕ), where θ is the angle outward from the optical axis and ϕ is the angle of rotation around the optical axis. We can write

$$R = \sqrt{X^2 + Y^2 + Z^2},\ \theta = \cos^{-1}\frac{R}{Z},\ \phi = \tan^{-1}\frac{Y}{X}$$

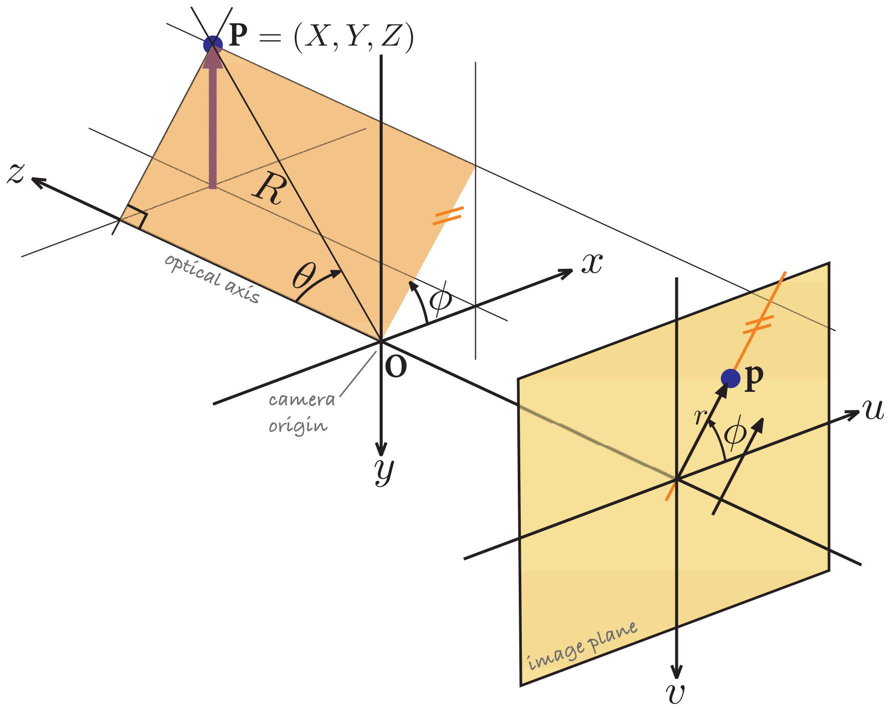


Fig. 11.15.
Image formation for a fisheye lens camera. The world point **P** is represented in spherical coordinates (R, θ, ϕ) with respect to the camera's origin

Table 11.1.
Fisheye lens projection models

Mapping	Equation
Equiangular	$r = k\theta$
Stereographic	$r = k\tan(\theta/2)$
Equisolid	$r = k\sin(\theta/2)$
Polynomial	$r = k_1\theta + k_2\theta^2 + \cdots$

On the image plane of the camera we represent the projection **p** in polar coordinates (r, ϕ) with respect to the principal point, where $r = r(\theta)$. The Cartesian image-plane coordinates are

$$u = r(\theta)\cos\phi,\ v = r(\theta)\sin\phi$$

and the exact nature of the function $r(\theta)$ depends on the type of fisheye lens. Some common projection models are listed in Table 11.1 and all have a scaling parameter k.

Using the Toolbox we can create a fisheye camera

```
>> cam = FishEyeCamera('name', 'fisheye', ...
         'projection', 'equiangular', ...
         'pixel', 10e-6, ...
         'resolution', [1280 1024])
```

which returns an instance of a `FishEyeCamera` object which is a subclass of the Toolbox's `Camera` object and polymorphic with the `CentralCamera` class discussed earlier. If k is not specified, as in this example, then it is computed such that a hemispheric field of view is projected into the maximal circle on the image plane. As is the case for perspective cameras the parameters such as principal point and pixel dimensions are generally not known and must be estimated using a calibration procedure.

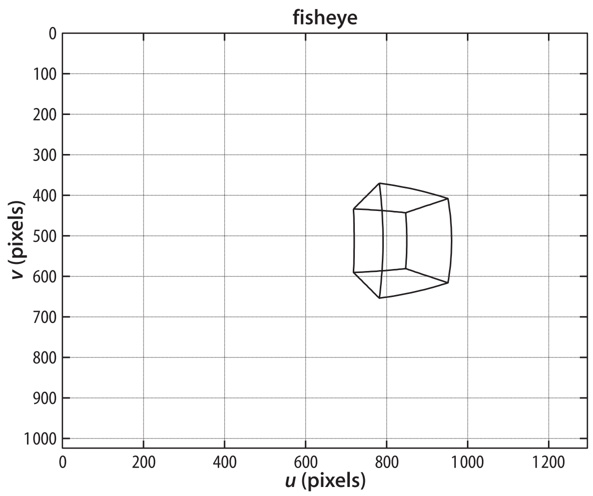


Fig. 11.16. A cube projected using the `FishEyeCamera` class. The straight edges of the cube are curves on the image plane

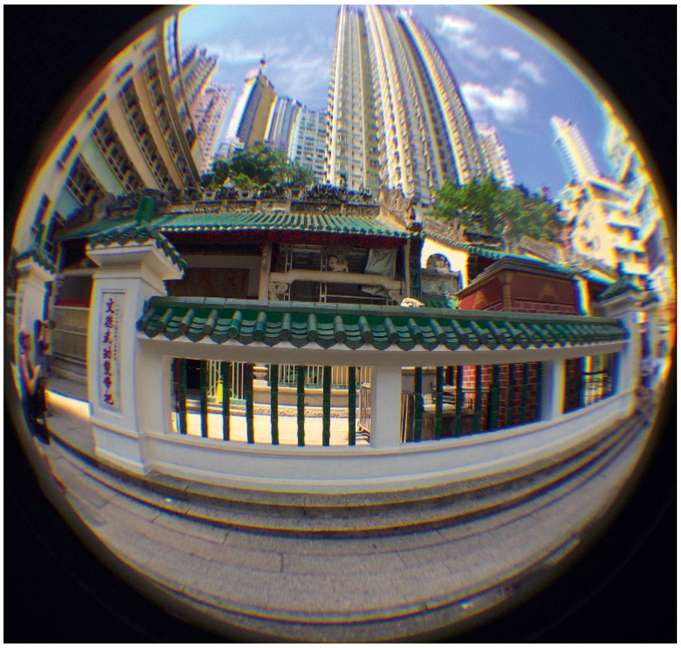


Fig. 11.17. Fisheye lens image. Note that straight lines in the world are no longer projected as straight lines. Note also that the field of view is mapped to a circular region on the image plane

We create an edge-based model of a cube with side length 0.2 m

```
>> [X,Y,Z] = mkcube(0.2, 'centre', [0.2, 0, 0.3], 'edge');
```

and project it to the fisheye camera's image plane

```
>> cam.mesh(X, Y, Z)
```

and the result is shown in Fig. 11.16. We see that straight lines in the world are no longer straight lines in the image.

Wide angle lenses are available with 180° and even 190° field of view, however they have some practical drawbacks. Firstly, the spatial resolution is lower since the camera's pixels are spread over a wider field of view. We also note from Fig. 11.17 that the field of view is a circular region which means that nearly 25% of the rectangular image plane is effectively wasted. Secondly, outdoors images are more likely to include bright sky so the camera will automatically reduce its exposure which can result in some nonsky parts of the scene being very underexposed.

11.3.2 Catadioptric Camera

A catadioptric imaging system comprises both reflective and refractive elements▶, a mirror and a lens, as shown in Fig. 11.18a. An example catadioptric image is shown in Fig. 11.18b.

From the Greek for curved mirrors (catoptrics) and lenses (dioptrics).

The geometry shown in Fig. 11.19 is fairly complex. A ray is constructed from the point **P** to the focal point of the mirror at **O** which is the origin of the camera system. The ray has an elevation angle of

$$\theta = \tan^{-1}\frac{Z}{X^2+Y^2} + \frac{\pi}{2}$$

upward from the optical axis and intersects the mirror at the point **M**. The reflected ray makes an angle ψ with respect to the optical axis which is a function of the incoming ray angle, that is $\psi(\theta)$. The relationship between θ and ψ is determined by the tangent to the mirror at the point **M** and is a function of the shape of the mirror. Many different mirror shapes are used for catadioptric imaging including spherical, parabolic, elliptical and hyberbolic. In general the function $\psi(\theta)$ is nonlinear but an interesting class of mirror is the equiangular mirror for which

$$\theta = \alpha\psi$$

The reflected ray enters the camera lens at angle ψ from the optical axis, and from the lens geometry we can write

$$r = \lambda \tan\psi$$

which is the distance from the principal point. The polar coordinate of the image-plane point is $\boldsymbol{p} = (r, \phi)$ and the corresponding Cartesian coordinate is

$$u = r\cos\phi,\ v = r\sin\phi$$

where ϕ is the azimuth angle

$$\phi = \tan^{-1}\frac{Y}{X}$$

In Fig. 11.19 we have assumed that all rays pass through a single focal point or viewpoint – **O** in this case. This is referred to as central imaging and the resulting image

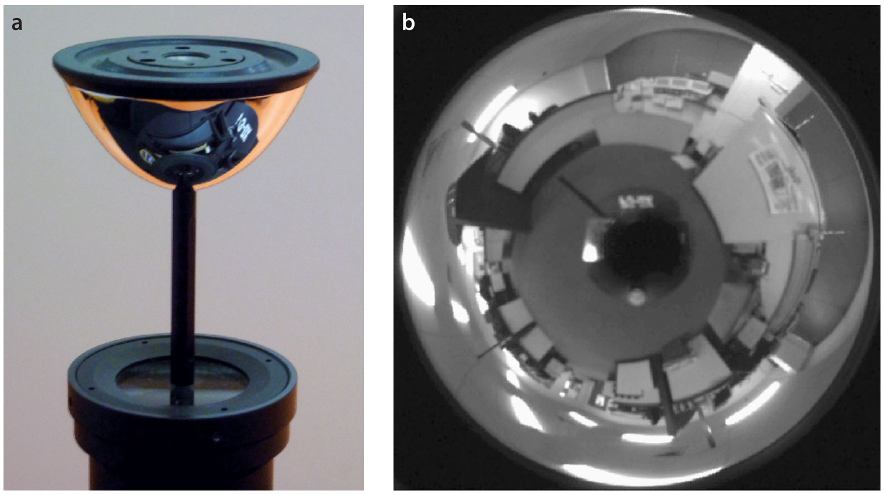


Fig. 11.18. Catadioptric imaging. **a** A catadioptric imaging system comprising a conventional perspective camera is looking upward at the mirror; **b** Catadioptric image. Note the dark spot in the center which is the support that holds the mirror above the lens. The floor is in the center of the image and the ceiling is at the edge (photos by Michael Milford)

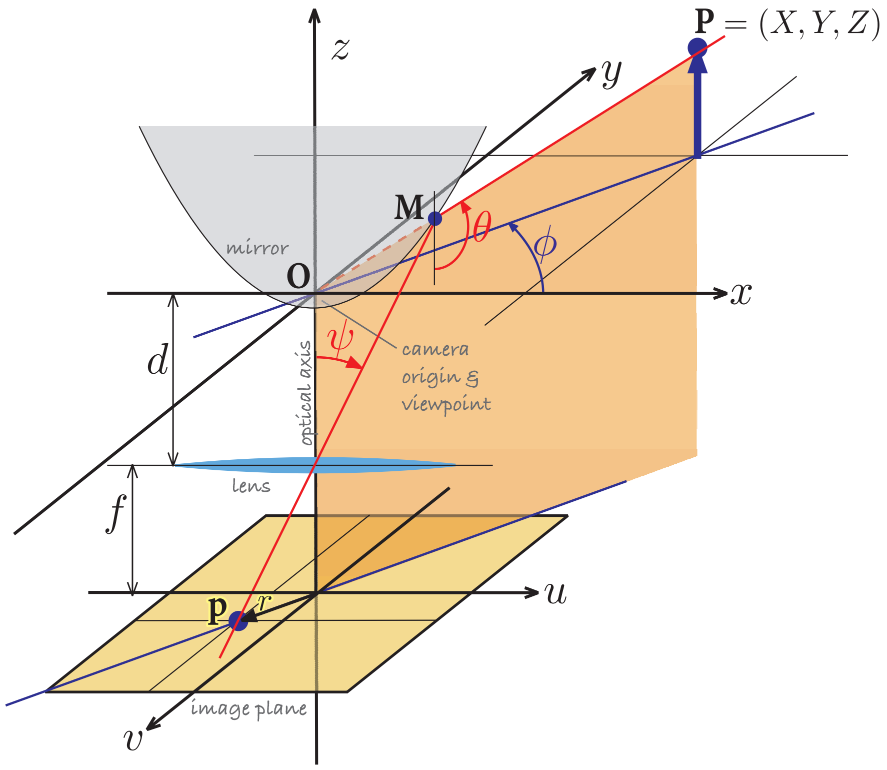


Fig. 11.19. Catadioptric image formation. A ray from point **P** at elevation angle θ and azimuth ϕ toward **O** is reflected from the mirror surface at **M** and is projected by the lens on to the image plane at **p**

can be correctly transformed to a perspective image. The equiangular mirror does not meet this constraint and is therefore a noncentral imaging system – the focal point varies with the angle of the incoming ray and lies along a short locus within the mirror known as the caustic. Conical, spherical and equiangular mirrors are all noncentral. In practice the variation in the viewpoint is very small compared to the world scale and many such mirrors are well approximated by the central model.

The Toolbox provides a model for catadioptric cameras. For example we can create an equiangular catadioptric camera

```
>> cam = CatadioptricCamera('name', 'panocam', ...
         'projection', 'equiangular', ...
         'maxangle', pi/4, ...
         'pixel', 10e-6, ...
         'resolution', [1280 1024])
```

which returns an instance of a `CatadioptricCamera` object which is a subclass of the Toolbox's `Camera` object and polymorphic with the `CentralCamera` class discussed earlier. The option `maxangle` specifies the maximum elevation angle θ from which the parameters α and f are determined such that the maximum elevation angle corresponds to a circle that maximally fits the image plane. The parameters can be individually specified using the options `'alpha'` and `'focal'`. Other supported projection models include parabolic and spherical and each camera type has different options as described in the online documentation.

We create an edge-based cube model

```
>> [X,Y,Z] = mkcube(1, 'centre', [1, 1, 0.8], 'edge');
```

which we project onto the image plane

```
>> cam.mesh(X, Y, Z)
```

and the result is shown in Fig. 11.20.

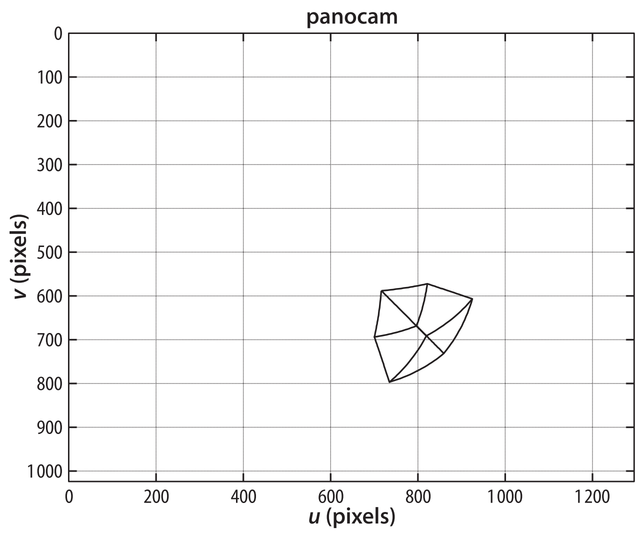


Fig. 11.20. A cube projected with an equiangular catadioptric camera

Catadioptric cameras have the advantage that they can view 360° in azimuth but they also have some practical drawbacks. They share many of the problems of fisheye lenses such as reduced spatial resolution, wasted image-plane pixels and exposure control. In some designs there is also a blind spot due to the mirror support which is commonly a central stalk or a number of side supports.

11.3.3 Spherical Camera

The fisheye lens and catadioptric systems just discussed guide the light rays from a large field of view onto an image plane. Ultimately the 2-dimensional image plane is a limiting factor and it is advantageous to consider instead an image *sphere* as shown in Fig. 11.21.

The world point **P** is projected by a ray to the origin of a unit sphere. The projection is the point **p** where the ray intersects the surface of the sphere. If we write $p = (x, y, z)$ then

$$x = \frac{X}{R},\ y = \frac{Y}{R},\ \text{and } z = \frac{Z}{R} \tag{11.17}$$

where $R = \sqrt{X^2 + Y^2 + Z^2}$ is the radial distance to the world point. The surface of the sphere is defined by $x^2 + y^2 + z^2 = 1$ so one of the three Cartesian coordinates is redundant. A minimal two-parameter representation for a point on the surface of a sphere (ϕ, θ) comprises the angle of colatitude measured down from the North pole

$$\theta = \sin^{-1} r,\quad \theta \in [0, \pi] \tag{11.18}$$

where $r = \sqrt{x^2 + y^2}$, and the azimuth angle (or longitude)

$$\phi = \tan^{-1}\frac{y}{x},\quad \phi \in [-\pi, \pi) \tag{11.19}$$

Conversely, the Cartesian coordinates for the point $\mathbf{p} = (\phi, \theta)$ are given by

$$x = \sin\theta\cos\phi,\ y = \sin\theta\sin\phi,\ z = \cos\theta \tag{11.20}$$

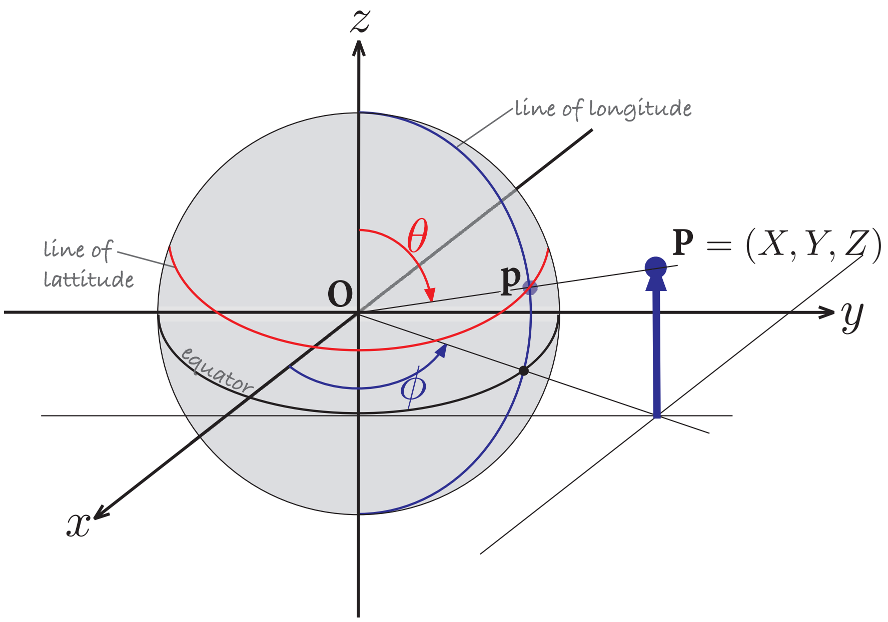


Fig. 11.21. Spherical image formation. The world point **P** is mapped to **p** on the surface of the unit sphere and represented by the angles of colatitude θ and longitude ϕ

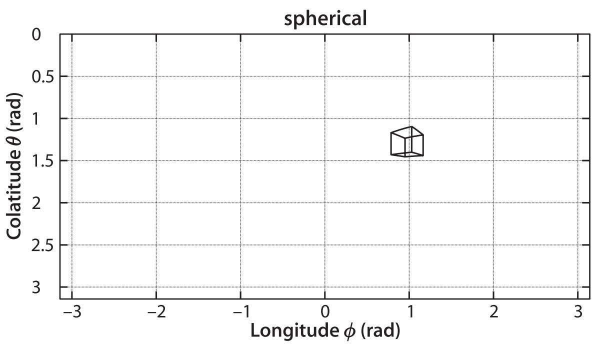


Fig. 11.22. Cube projected by a spherical camera. The spherical image plane is represented in Cartesian coordinates

Using the Toolbox we can create a spherical camera

```
>> cam = SphericalCamera('name', 'spherical')
```

which returns an instance of a `SphericalCamera` object which is a subclass of the Tool-box's `Camera` object and polymorphic with the `CentralCamera` class discussed earlier.

As previously we can create an edge-based cube model

```
>> [X,Y,Z] = mkcube(1, 'centre', [2, 3, 1], 'edge');
```

and project it onto the sphere

```
>> cam.mesh(X, Y, Z)
```

and this is shown in Fig. 11.22. To aid visualization the spherical image plane has been unwrapped into a rectangle – lines of longitude and latitude are displayed as vertical and horizontal lines respectively. The top and bottom edges correspond to the north and south poles respectively.

It is not yet possible to buy a spherical camera although prototypes have been demonstrated in several laboratories. The spherical camera is more useful as a conceptual construct to simplify the discussion of wide-angle imaging. As we show in the next section we can transform images from perspective, fisheye or catadioptric camera onto the sphere where we can treat them in a uniform manner.

11.4 Unified Imaging

We have introduced a number of different imaging models in this chapter. Now we will discuss how to transform an image captured with one type of camera to the image that would have been captured with a different type of camera. For example, given a fisheye lens projection we will generate the corresponding projection for a spherical camera or a perspective camera. The unified imaging model provides a powerful framework to consider very different types of cameras such as standard perspective, catadioptric and many types of fisheye lens.

The unified imaging model is a two-step process and the notation is shown in Fig. 11.23. The first step is spherical projection of the world point **P** to the surface of the unit sphere $\mathbf{p}'$ as discussed in the previous section and described by Eq. 11.17 to Eq. 11.18. The view point **O** is the center of the sphere which is a distance m from the image plane along its normal z-axis. The single view point implies a *central* camera.

In the second step the point $\mathbf{p}' = (\theta, \phi)$ is reprojected to the image plane **p** using the view point **F** which is at a distance ℓ along the z-axis above **O**. The polar coordinates of the image-plane point are $\mathbf{p} = (r, \phi)$ where

$$r = \frac{(\ell + m)\sin\theta}{\ell - \cos\theta} \tag{11.21}$$

The unified imaging model has only two parameters m and ℓ and these are a function of the type of camera as listed in Table 11.2. For a perspective camera the two view points **O** and **F** are coincident and the geometry becomes the same as the central perspective model shown in Fig. 11.3.

For catadioptric cameras with mirrors that are conics the focal point **F** lies between the center of the sphere and the north pole, that is, $0 < \ell < 1$. This projection model is somewhat simpler than the catadioptric camera geometry shown in Fig. 11.19. The imaging parameters are written in terms of the conic parameters eccentricity ε and latus rectum $4p$.▸

The length of a chord parallel to the directrix and passing through the focal point.

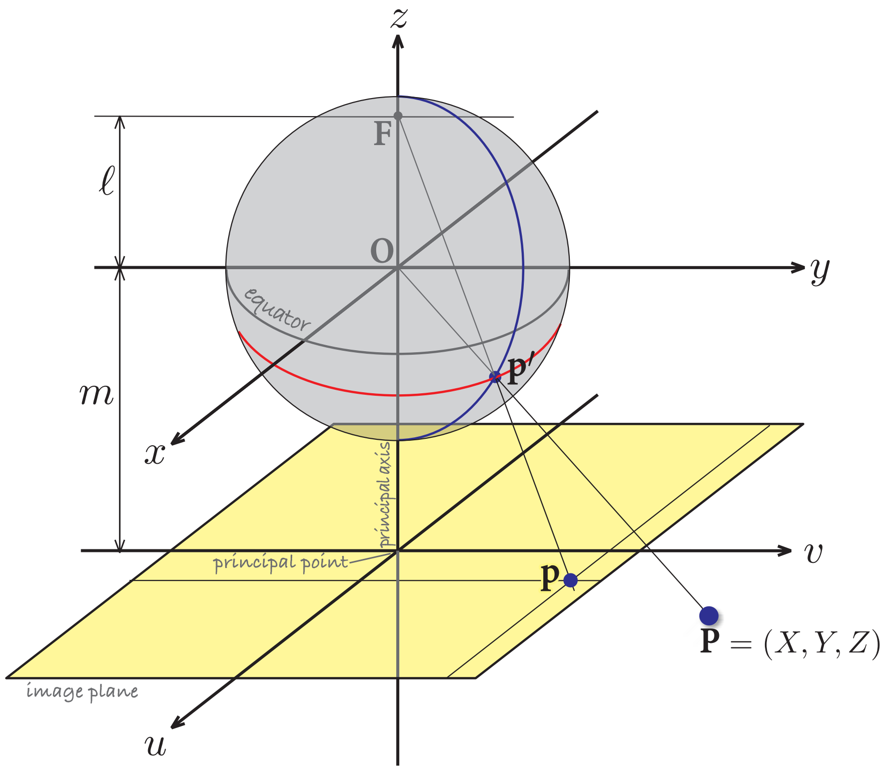


Fig. 11.23. Unified imaging model of Geyer and Daniilidis (2000)

Table 11.2. Unified imaging model parameters ℓ and m according to camera type. ε is the eccentricity of the conic and $4p$ is the latus rectum

Imaging	ℓ	m
Perspective	0	f
Stereographic	1	f
Fisheye	>1	f
Catadioptric (elliptical, $0 < \varepsilon < 1$)	$\frac{2\varepsilon}{1+\varepsilon^2}$	$\frac{2\varepsilon(2p-1)}{1+\varepsilon^2}$
Catadioptric (parabolic, $\varepsilon = 1$)	1	$2p - 1$
Catadioptric (hyperbolic, $\varepsilon > 1$)	$\frac{2\varepsilon}{1+\varepsilon^2}$	$\frac{2\varepsilon(2p-1)}{1+\varepsilon^2}$

The projection with **F** at the north pole is known as stereographic projection and is used in many fields to project the surface of a sphere onto a plane. Many fisheye lenses are extremely well approximated by **F** above the north pole.

11.4.1 Mapping Wide-Angle Images to the Sphere

We can use the unified imaging model in reverse. Consider an image captured by a wide field of view camera such as the fisheye image shown in Fig. 11.24a. If we know the location of **F** then we can project each point from the image onto the sphere to create a spherical image, even though we do not have a spherical camera.

In order to achieve this inverse mapping we need to know some parameters of the camera that captured the image. A common feature of images captured with a fisheye lens or catadioptric camera is that the outer bound of the image is a circle. This circle can be found and its center estimated quite precisely – this is the principal point. A variation of the camera calibration procedure of Sect. 11.2.4 is applied which uses corresponding world and image-plane points from the planar calibration target shown in Fig. 11.24a. This particular camera has a field of view of 190 degrees and its calibration parameters have been estimated to be: principal point (528.1214, 384.0784), $\ell = 2.7899$ and $m = 996.4617$.

We will illustrate this using the image shown in Fig. 11.24a

```
>> fisheye = iread('fisheye_target.png', 'double', 'grey');
```

and we also define the domain of the input image

```
>> [Ui,Vi] = imeshgrid(fisheye);
```

We will use image warping to achieve this mapping. Warping is discussed in detail in Sect. 12.7.4 but we will preview the approach here. The output domain covers the entire sphere with longitude from $-\pi$ to $+\pi$ radians and colatitude from 0 to π radians with 500 steps in each direction

```
>> n = 500;
>> theta_range = linspace(0, pi, n);
>> phi_range = linspace(-pi, pi, n);
>> [Phi,Theta] = meshgrid(phi_range, theta_range);
```

For warping we require a function that returns the coordinates of a point in the input image given the coordinates of a point in the output spherical image. This function is the second step of the unified imaging model Eq. 11.21 which we implement as

```
>> r = (l+m)*sin(Theta) ./ (l-cos(Theta));
```

from which the corresponding Cartesian coordinates in the input image are

```
>> U = r.*cos(Phi) + u0;
>> V = r.*sin(Phi) + v0;
```

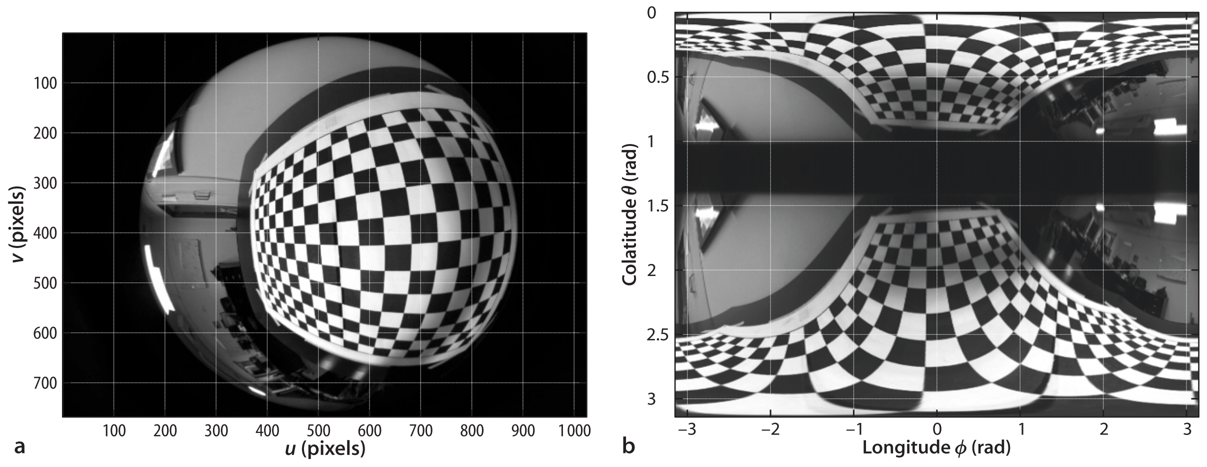


Fig. 11.24. Fisheye image of a planar calibration target. **a** Fisheye image (image courtesy of Peter Hansen); **b** Image warped to (ϕ, θ) coordinates

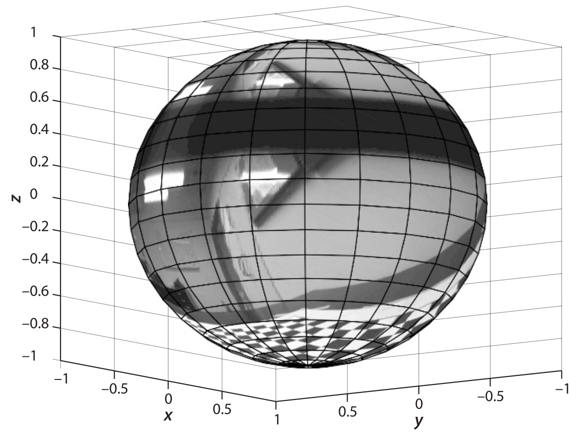


Fig. 11.25. Fisheye image mapped to the unit sphere. We can clearly see the planar grid lying on a table, the ceiling light, a door and a whiteboard

The image warp is performed with a single MATLAB builtin function

```
>> spherical = interp2(Ui, Vi, fisheye, U, V);
```

where the first three arguments define the input domain, and the last two arguments are the coordinates for which grey-scale values will be interpolated from the input image and returned. We display the result

```
>> idisp(spherical)
```

which is shown in Fig. 11.24b. The image appears reflected about the equator and this is because the mapping from a point on the image plane to the sphere is double valued – since **F** is above the north pole the ray intersects the sphere twice. The top and bottom row of this image corresponds to the principal point, while the dark band above the equator corresponds to the circular outer edge of the input image.

The image is extremely distorted but this coordinate system is very convenient to texture map onto a sphere

```
>> sphere
>> h = findobj('Type', 'surface');
>> set(h, 'CData', flipud(spherical), 'FaceColor', 'texture');
>> colormap(gray)
```

and this is shown in Fig. 11.25. Using the MATLAB figure toolbar we can rotate the sphere and look at the image from different view points.

Any wide-angle image that can be expressed in terms of central imaging parameters can be similarly projected onto a sphere. So too can multiple perspective images obtained from a camera array such as shown in Fig. 11.27.

11.4.2 Mapping from the Sphere to a Perspective Image

Given a spherical image we now want to reconstruct a perspective view in a particular direction. We can think of this as looking out, from inside the sphere, at a small surface area which is close to flat and approximates a perspective camera view. This is the second step of the unified imaging model where **F** is at the center of the sphere in which case Fig. 11.23 becomes similar to Fig. 11.3. The perspective camera's optical axis is the negative *z*-axis of the sphere.

For this example we will use the spherical image created in the previous section. We wish to create a perspective image of 1 000 × 1 000 pixels and with a field-of-view of 45°. The field of view can be written in terms of the image width *W* and the unified imaging parameter *m* as

$$\theta_{\text{FOV}} = 2\tan^{-1}\frac{W}{2m}$$

For a 45° field-of-view we require

```
>> W = 1000;
>> m = W / 2 / tan(45/2*pi/180)
m =
   1.2071e+03
```

and for perspective projection we require

```
>> l = 0;
```

We also require the principal point to be in the center of the image

```
>> u0 = W/2; v0 = W/2;
```

The domain of the output image will be

```
>> [Uo,Vo] = meshgrid(0:W-1, 0:W-1);
```

The polar coordinate (r, ϕ) of each point in the output image is

```
>> [phi,r ]= cart2pol(Uo-u0, Vo-v0);
```

and the corresponding spherical coordinates (ϕ, θ) are

```
>> Phi_o = phi;
>> Theta_o = pi - atan(r/m);
```

We now warp from spherical coordinates to the perspective image plane

```
>> perspective = interp2(Phi, Theta, spherical, Phi_o, Theta_o);
```

and the result

```
>> idisp(perspective)
```

is shown in Fig. 11.26a. This is the view from a perspective camera at the center of the sphere looking down through the south pole. We see that the lines on the chessboard calibration target are now straight as we would expect from a perspective image.

Of course we are not limited to just looking along the negative *z*-axis of the sphere. In Fig. 11.25 we can see some other features of the room such as a door, a whiteboard and some ceiling lights. We can point our virtual perspective camera in their direction by first rotating the spherical image

From a single wide-angle image we can create a perspective view in any direction without having any mechanical pan/tilt mechanism – it's just computation. In fact multiple users could look in different directions simultaneously.

```
>> spherical = sphere_rotate(spherical, SE3.Ry(0.9)*SE3.Rz(-1.5) );
```

so that the negative *z*-axis now points toward the distant wall. Repeating the warp process we obtain the result shown in Fig. 11.26b in which we can clearly see a door and a whiteboard.◀

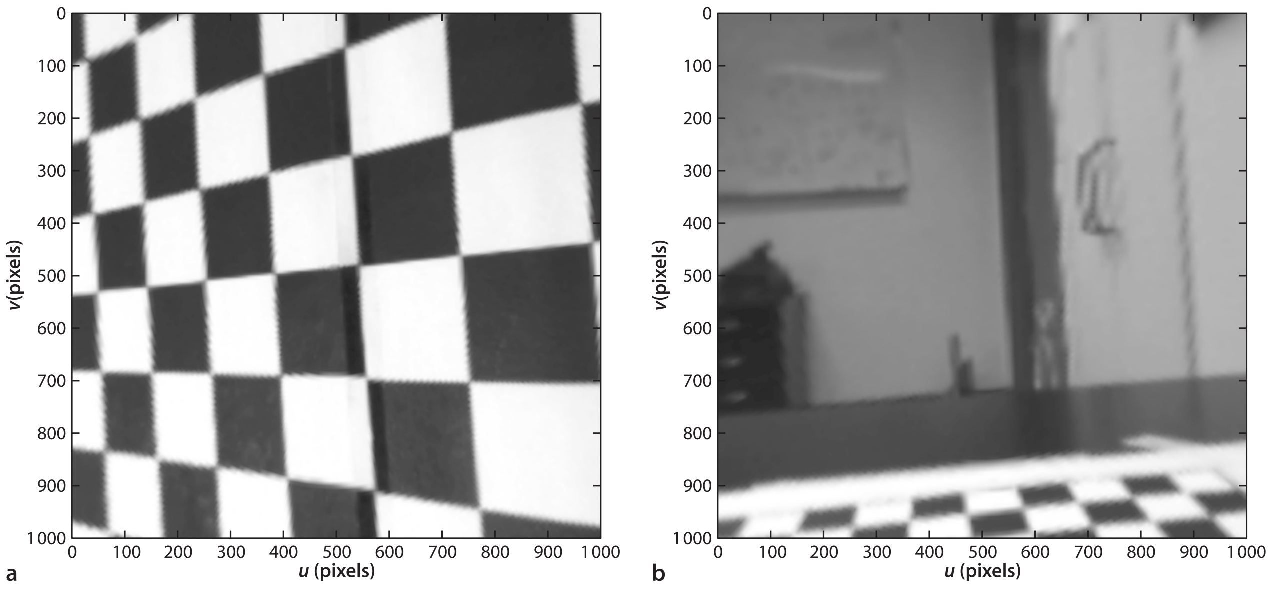


Fig. 11.26. Perspective projection of spherical image Fig. 11.25 with a field of view of 45°. **a** Note that the lines on the chessboard are now straight. **b** This view is looking toward the door and whiteboard

The original wide-angle image contains a lot of detail though it can be hard to see because of the distortion. After mapping the image to the sphere we can create a virtual perspective camera view (where straight lines in the world are straight) along any line of sight. This is only possible if the original image was taken with a central camera that has a single viewpoint. In theory we cannot create a perspective image from a noncentral wide-angle image but in practice if the caustic is small the parallax errors introduced into the perspective image will be negligible.

11.5 Novel Cameras

11.5.1 Multi-Camera Arrays

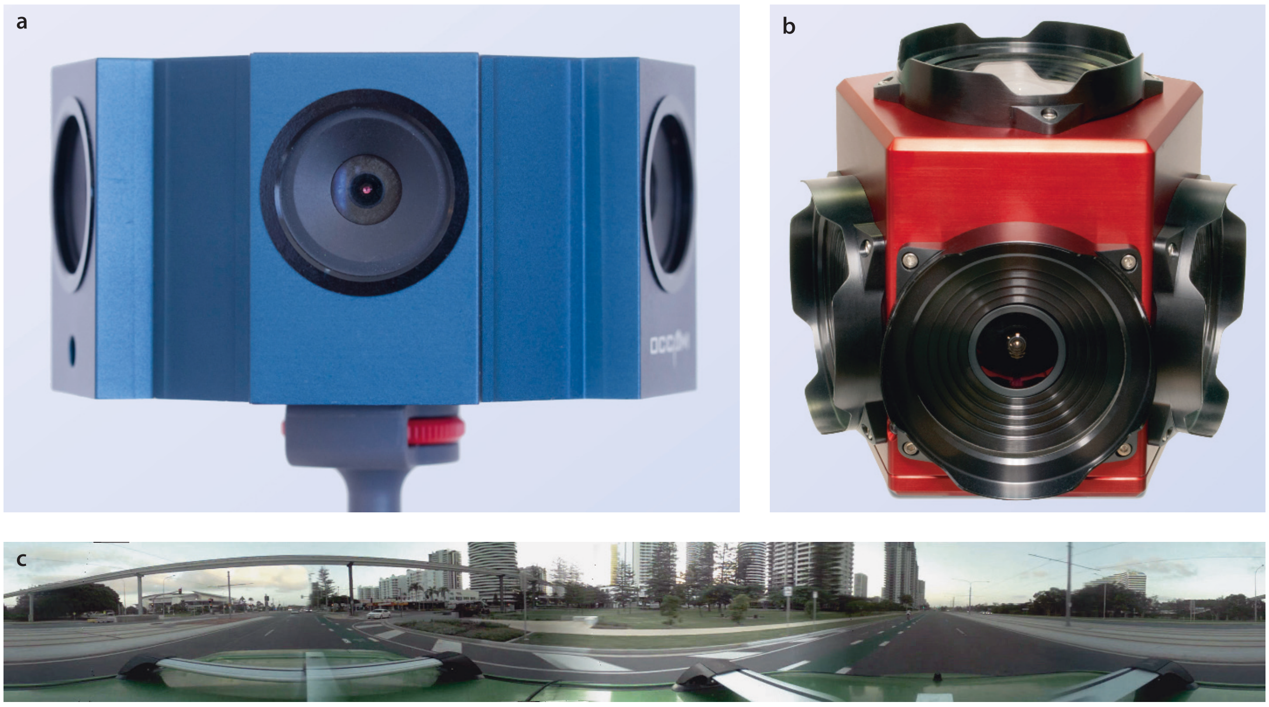
The cost of cameras and computation continues to fall making it feasible to do away with the unusual and expensive lenses and mirrors discussed so far, and instead use computation to stitch together the images from a number of cameras onto a cylindrical or spherical image plane. One such camera is shown in Fig. 11.27a and uses five cameras to capture a 360° panoramic view as shown in Fig. 11.27c. The camera in Fig. 11.27b uses six cameras to achieve an almost spherical field of view.

These camera arrays are not central cameras since light rays converge on the focal points of the individual cameras, not the center of the camera assembly. This can be problematic when imaging objects at short range but in typical use the distance between camera focal points, the caustic, is small compared to distances in the scene. The different viewpoints do have a real advantage however when it comes to capturing the light field.

11.5.2 Light-Field Cameras

As we discussed in the early part of this chapter a traditional perspective camera – analog or digital – captures a representation of a scene using the two dimensions of the film or sensor. We can think of the captured image as a 2-dimensional function $\mathcal{L}(X, Y)$ that describes the light emitted by the 3D scene. The function is scalar $\mathcal{L}(\cdot) \in \mathbb{R}$ for the monochrome case and vector-valued $\mathcal{L}(\cdot) \in \mathbb{R}^3$ for a tristimulus color representation.▸

We could add extra dimensions to represent polarization of the light.



▲ **Fig. 11.27.** Omnidirectional camera array. **a** Five perspective cameras provide a 360° panorama with a 72° vertical field of view (camera by Occam Vision Group). **b** Panoramic camera array uses six perspective cameras to provide 90% of a spherical field of view. **c** A seamless panoramic image (3 760 × 480 pixels) as output by the camera **a** (photographs **a** and **c** by Edward Pepperell; image **b** courtesy of Point Grey Research)

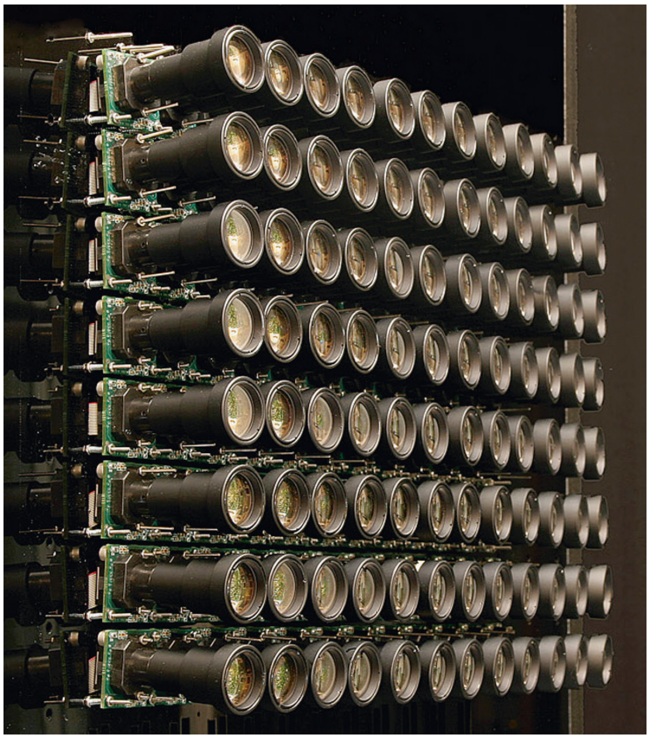
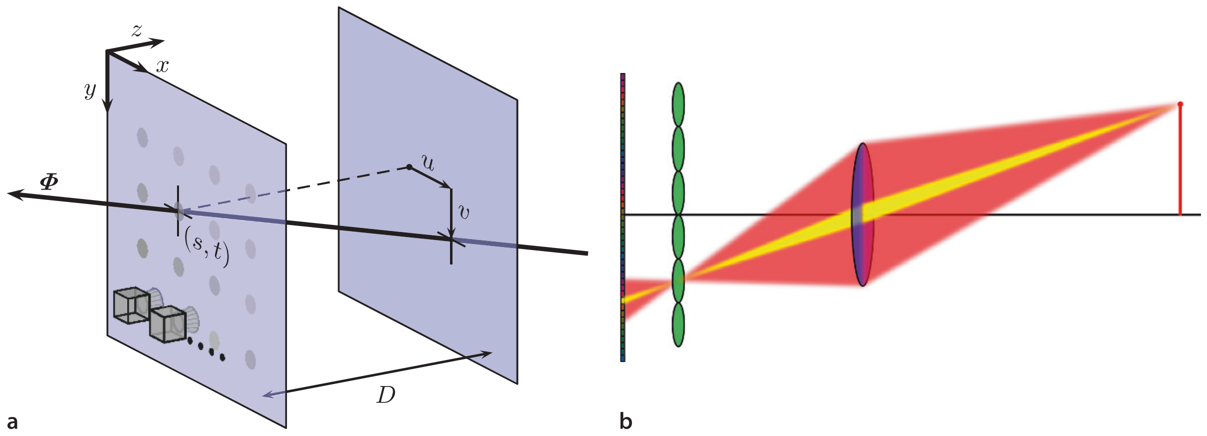


Fig. 11.28. An 8×12 camera array as described in Wilburn et al. (2005) (photo courtesy of Marc Levoy, Stanford University)

The pin-hole camera of Fig. 11.1 allows only a very small number of light rays to pass through the aperture, yet space is filled with innumerable light rays that provide a richer and more complete description of the world. This detailed geometric distribution of light is called the plenoptic function.◂ Luminance is really a function of position and direction in 3-dimensional space, for example $\mathcal{L}(X, Y, Z, \theta, \phi)$.

The word plenoptic comes from the Latin word plenus meaning full or complete.



Unintuitively lines in 3D space have only four parameters, see Sect. C.1.2.2, so the plenoptic function can be written as $\mathcal{L}(s, t, u, v)$ using the 2-plane parameterization shown in Fig. 11.29a. The traditional camera image is just a 2-dimensional *slice* of the full plenoptic function.

Fig. 11.29. **a** The light ray Φ passes through the image plane at point (u, v) and the center of the camera at (s, t). This is similar to the central projection model shown in Fig. 11.3. Any ray can be described by two points, in this case (u, v) and (s, t). **b** Path of light rays from object through main objective lens and lenslet array to the pixel array (figures courtesy Donald G. Dansereau)

If the scene contains obstructions then the rays become finite-length line segments, this increases the dimensionality of the light field from 4D to 5D. However to record such a light field the camera would have to be simultaneously at every position in the scene without obscuring anything, which is impossible.

Although the concepts behind the light field have been around for decades, it is only in recent years that the technology to capture light fields has become widely available. Early light-field cameras were arrays of regular cameras arranged in a plane, such as shown in Fig. 11.28, or on a sphere surrounding the scene, but these tended to be physically large, complex and expensive to construct. More recently low-cost and compact light-field cameras based on microlens arrays have come on to the market. One selling point for consumer light-field cameras has been the ability to refocus the image *after* taking the picture but the light-field image has many other virtues including synthesizing novel views, 3D reconstruction, low-light imaging and seeing through particulate obscurants.

The microlens array is a regular grid of tiny lenses, typically comprising hundreds of thousands of lenses, which is placed a fraction of a millimeter above the surface of the camera's photosensor array. The main objective lens focuses an image onto the surface of the microlens array as shown in Fig. 11.29b. The microlens directs incoming light to one of a small, perhaps 8×8, patch of pixels according to its direction. The resulting image captures information about both the origin of the ray (the lenslet) and its direction (the particular pixel beneath the lenslet). By contrast, in a standard perspective camera all the rays, irrespective of direction, contribute to the value of the pixel. The light-field camera pixels are sometimes referred to as *raxels* and the resolution of these cameras is typically expressed in megarays.

The raw image from the sensor array looks like Fig. 11.30a but can be *decoded* into a 4-dimensional light field, as shown in Fig. 11.30b, and used to render novel views.

11.6 Advanced Topics

11.6.1 Projecting 3D Lines and Quadrics

In Sect. 11.1 we projected 3D points to the image plane, and we projected 3D line segments by simply projecting their endpoints and joining them on the image plane. How would we project an arbitrary line in 3-dimensional space?

The first issue we confront is how to represent such a line and there are many possibilities which are discussed in Sect. C.1.2.2. One useful parameterization is Plücker coordinates – a 6-vector with many similarities to twists.

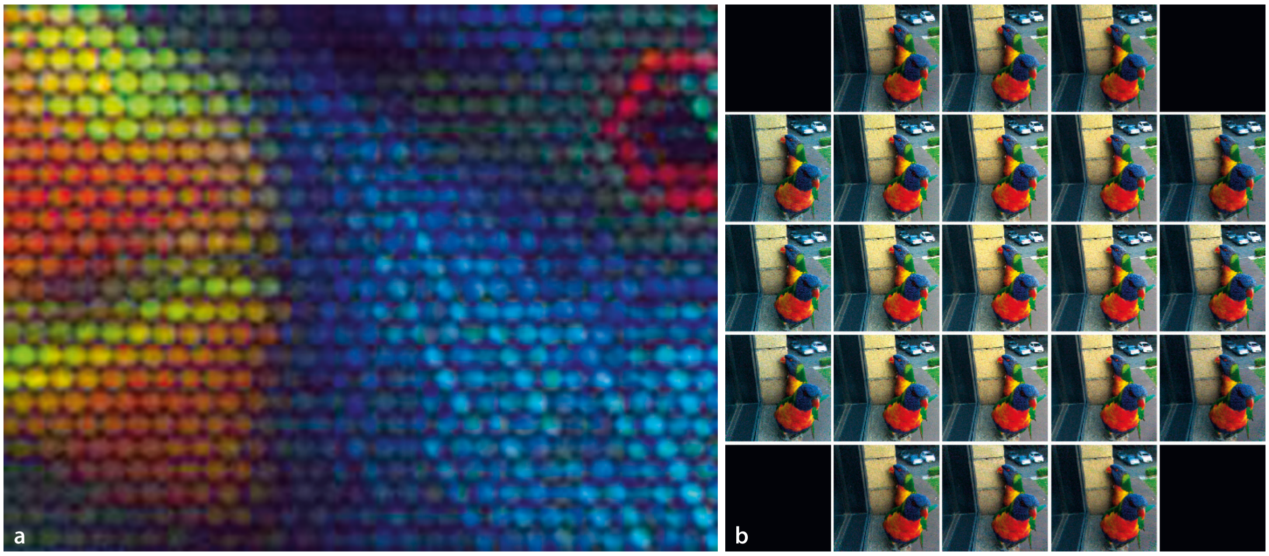


Fig. 11.30. a Closeup of image formed on the sensor array by the lenslet array; **b** array of images rendered from the light field for different camera view points (figures courtesy Donald G. Dansereau)

We can easily create a Plücker line using the Toolbox. A line that passes through (0, 0, 1) and (1, 1, 1) would be

```
>> L = Plucker([0 0 1], [1 1 1])
L =
{ -1  1  0; -1  -1  0 }
```

which returns a `Plucker` object that is represented as a 6-vector with two components: a moment vector and a direction vector. Options can be used to specify a line using a point and a direction or the intersection of two planes. The direction of the Plücker line is the vector

```
>> L.w'
ans =
    -1    -1     0
```

The `Plucker` object also has methods for plotting as well as determining the intersection with planes or other Plücker lines. There are many representations of a Plücker line including the 6-vector used above, a minimal 4-vector, and a skew-symmetric 4×4 matrix computed using the `L` method. The latter is used to project the line by

$$\ell = \vee\left(\boldsymbol{C}[\boldsymbol{L}]_\times \boldsymbol{C}^T\right)$$

where $\boldsymbol{C}$ is the camera matrix, and results in a 2-dimensional line expressed in homogeneous coordinates. Observing this line with the default camera

```
>> cam = CentralCamera('default');
>> l = cam.project(L)'
l =
     1    -1     0
```

results in a diagonal line across the image plane. We could plot this using `plot_homline` or on the camera's virtual image plane by

```
>> cam.plot(l)
```

Quadrics, short for quadratic surfaces, are a rich family of 3-dimensional surfaces. There are 17 standard types including spheres, ellipsoids, hyperboloids, paraboloids, cylinders and cones all described by

$$\tilde{\boldsymbol{x}}^T \boldsymbol{Q} \tilde{\boldsymbol{x}} = 0$$

where $\boldsymbol{Q} \in \mathbb{R}^{4\times4}$ is symmetric. The *outline* of the quadric is projected to the image plane by

$$\boldsymbol{c}^* = \boldsymbol{C}\boldsymbol{Q}^*\boldsymbol{C}^T$$

where $(\cdot)^*$ represents the adjugate operation, see Appendix B, and $\boldsymbol{c}$ is a matrix representing a conic section on the image plane

$$\tilde{\boldsymbol{p}}^T \boldsymbol{c} \tilde{\boldsymbol{p}} = 0$$

which can be written as

$$Au^2 + Buv + Cv^2 + Du + Ev + F = 0$$

where

$$\boldsymbol{c} = \left(\begin{array}{cc|c} A & B/2 & D/2 \\ B/2 & C & E/2 \\ \hline D/2 & E/2 & F \end{array}\right)$$

The determinant of the top-left submatrix indicates the type of conic: negative for a hyperbola, 0 for a parabola and positive for an ellipse.

To demonstrate this we will define a camera looking toward the origin

```
>> cam = CentralCamera('default', 'pose', SE3(0.2,0.1, -5)*SE3.Rx(0.2));
```

and define a unit sphere at the origin

```
>> Q = diag([1 1 1 -1]);
```

then compute its projection to the image plane

```
>> Qs = inv(Q)*det(Q); % adjugate
>> cs = cam.C * Qs * cam.C';
>> c = inv(cs)*det(cs);  % adjugate
```

which is a 3×3 matrix describing a conic. The determinant

```
>> det(c(1:2,1:2))
ans =
   2.2862e+14
```

is positive indicating an ellipse, and a simple way to plot this is using the Symbolic Math Toolbox™

```
>> syms x y real
>> ezplot([x y 1]*c*[ x y 1]', [0 1024 0 1024])
>> set(gca, 'Ydir', 'reverse')
```

11.6.2 Nonperspective Cameras

The camera matrix Eq. 11.9 represents a special subset of all possible camera matrices – finite projective or Euclidean cameras – where the left-hand 3×3 matrix is nonsingular. The camera projection matrix $\boldsymbol{C}$ from Eq. 11.9 can be written generally as

$$\boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & 1 \end{pmatrix}$$

which has arbitrary scale so one element, typically $\boldsymbol{C}_{3,4}$ is set to one – this matrix has 11 unique elements or 11 DOF. We could think of every possible matrix as corresponding to some type of camera, but most of them would produce wildly distorted images.

Orthographic or parallel projection is a simple perspective-free projection of 3D points onto a plane, like a "plan view". For small objects close to the camera this projection can be achieved using a telecentric lens. The apparent size of an object is independent of its distance.

For the case of an aerial robot flying high over relatively flat terrain the variation of depth, the depth relief, Δ_Z is small compared to the average depth of the scene $\bar{Z}$, that is $\Delta_Z \ll \bar{Z}$. We can use a scaled-orthographic projection which is an orthographic projection followed by uniform scaling $m = f / \bar{Z}$.

These two nonperspective cameras are special cases of the more general affine camera model which is described by a matrix of the form

$$\boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

that can be factorized as

$$\boldsymbol{C} = \underbrace{\begin{pmatrix} m_x & s & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsic}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{projection}} \underbrace{\begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}}_{\text{extrinsic}}$$

It can be shown that the principal point is undefined for such a camera model, simplifying the intrinsic matrix, but we have introduced a skew parameter to handle the case of nonorthogonal sensor axes. The projection matrix is different compared to the perspective case in Eq. 11.9 – the column of zeros has moved from column 4 to column 3. This zero column effectively deletes the third row and column of the extrinsic matrix resulting in

$$\boldsymbol{C} = \underbrace{\begin{pmatrix} m_x & s & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{3\text{ DOF}} \underbrace{\begin{pmatrix} [\boldsymbol{R}]_{2\times 3} & \begin{matrix} t_x \\ t_y \end{matrix} \\ 000 & 1 \end{pmatrix}}_{5\text{ DOF}}$$

The 2×3 submatrix of the rotation matrix has 6 elements but 3 constraints – the two rows have unit norms and are orthogonal – and therefore has 3 DOF.

and has at most 8 DOF◂. The independence from depth is very clear since t_z does not appear. The case where skew $s = 0$ and $m_x = m_y = 1$ is orthographic projection and has only 5 DOF, while the scaled-orthographic case when $s = 0$ and $m_x = m_y$ has 6 DOF. The case where $m_x \neq m_y$ is known as weak perspective projection, although this term is sometimes also used to describe scaled-orthographic projection.

11.7 Wrapping Up

We have discussed the first steps in the computer vision process – the formation of an image of the world and its conversion to an array of pixels which comprise a digital image. The images with which we are familiar are perspective projections of the world in which 3 dimensions are compressed into 2 dimensions. This leads to ambiguity about object size – a large object in the distance looks the same as a small object that is close. Straight lines and conics are unchanged by this projection but shape distortion occurs – parallel lines can appear to converge and circles can appear as ellipses. We have modeled the perspective projection process and described it in terms of eleven parameters – intrinsic and extrinsic. Geometric lens distortion adds additional lens parameters. Camera calibration is the process of estimating these parameters and two approaches have been introduced. We also discussed pose estimation where the pose of an object of known geometry can be estimated from a perspective projection obtained using a calibrated camera.

Perspective images are limited in their field of view and we discussed several wide-angle imaging systems based on the fisheye lens, catadioptrics and multiple cameras. We also discussed the ideal wide-angle camera, the spherical camera, which is currently still a theoretical construct. However it can be used as an intermediate representation in the unified imaging model which provides one model for almost all camera geometries. We used the unified imaging model to convert a fisheye camera image to a spherical image and then to a perspective image along a specified view axis. Finally we covered some more recent camera developments such as panoramic camera arrays and light-field cameras.

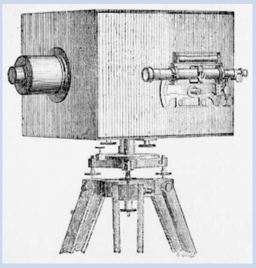
In this chapter we treated imaging as a problem of pure geometry with a small number of world points or line segments. In the next chapter we will discuss the acquisition and processing of images sourced from files, cameras and the web.

Further Reading and Resources

Computer vision textbooks such as Szeliski (2011), Hartley and Zisserman (2003), Forsyth and Ponce (2011) and Gonzalez and Woods (2008) provide deeper coverage of the topics introduced in this chapter. Many topics in geometric computer vision have also been studied by the photogrammetric community, but different language is used. For example camera calibration is known as camera resectioning, and pose estimation is known as space resectioning. The revised classic textbook by DeWitt and Wolf (2000) is a thorough and readable introduction to photogrammetry.

Camera calibration. The homogeneous transformation calibration (Sutherland 1974) approach of Sect. 11.2.1 is also known as the direct linear transform (DLT) in the photogrammetric literature. The Toolbox implementation `camcald` requires that the centroids of the calibration markers have already been determined which is a nontrivial problem (Corke 1996b, § 4.2). It also cannot estimate lens distortion. Wolf (1974) describes extensions to the linear camera calibration with models that include up to 18 parameters and suitable nonlinear optimization estimation techniques. A more concise description of nonlinear calibration is provided by Forsyth and Ponce (2011). Hartley and Zisserman (2003) describe how the linear calibration model can be obtained using features such as lines within the scene.

There are a number of good camera calibration toolboxes available on the web. The MATLAB Toolbox, discussed in Sect. 11.2.4, is by Jean-Yves Bouguet and available from http://www.vision.caltech.edu/bouguetj/calib_doc. It has extensive online documentation and includes example calibration images which were used in Sect. 11.2.4. Several tools build on this and automatically find the chessboard target which is otherwise tedious to locate in every image, for example the AMCC and RADOCC Toolboxes and the MATLAB Camera Calibrator App included with the Computer Vision System Toolbox™. The MATLAB Toolbox by Janne Heikkilä is available at http://www.ee.oulu.fi/~jth/calibr/ and works for planar or 3D targets with circular dot features and estimates lens distortion.



Photogrammetry is the science of understanding the geometry of the world from images. The techniques were developed by the French engineer Aimé Laussedat (1819–1907) working for the Army Corps of Engineers in the 1850s. He produced the first measuring camera and developed a mathematical analysis of photographs as perspective projections. He pioneered the use of aerial photography as a surveying tool to map Paris – using rooftops as well as unmanned balloons and kites.

Photogrammetry is normally concerned with making maps from images acquired at great distance but the subfield of close-range or terrestrial photogrammetry is concerned with camera to object distances less than 100 m which is directly relevant to robotics. (Image from La Métrophotographie, Aimé Laussedat, 1899)

Pose estimation is a classic and hard problem in computer vision and for which there exists a very large literature. The approaches can be broadly divided into analytic and iterative solutions. Assuming that lens distortion has been corrected the analytic solutions for three and four noncollinear points are given by Fischler and Bolles (1981), DeMenthon and Davis (1992) and Horaud et al. (1989). Typically multiple solutions exist but for four coplanar points there is a unique solution. Six or more points always yield unique solutions, as well as the intrinsic camera calibration parameters. Iterative solutions were described by Rosenfeld (1959) and Lowe (1991). A more recent discussion based around the concept of bundle adjustment is provided by Triggs et al. (2000). The pose estimation in the Toolbox is a wrapper around an efficient noniterative perspective n-point pose estimator described by Lepetit et al. (2009) and available at http://cvlab.epfl.ch/EPnP. Pose estimation requires a geometric model of the object and such computer vision approaches are known as *model-based vision*. An interesting historical perspective on model-based vision is the 1987 video by the late Joe Mundy which is available at http://www.archive.org/details/JosephMu1987.

Wide field-of-view cameras. There is recent and growing interest in this type of camera and today good quality lightweight fisheye lenses and catadioptric camera systems are available. Nayar (1997) provides an excellent motivation for, and introduction to, wide-angle imaging. A very useful online resource is the catadioptric sensor design page at http://www.math.drexel.edu/~ahicks/design and a page of links to research groups, companies and workshops at http://www.cis.upenn.edu/~kostas/omni.html. Equiangular mirror systems were described by Chahl and Srinivasan (1997) and Ollis et al. (1999). Nature's solution, the reflector-based scallop eye, is described in Colicchia et al. (2009). A number of workshops on Omnidirectional Vision have been held, starting in 2000, and their proceedings are a useful introduction to the field. The book of Daniilidis and Klette (2006) is a collection of papers on nonperspective imaging and Benosman and Kang (2001) is another, earlier, published collection of papers. Some information is available through CVonline at http://homepages.inf.ed.ac.uk/rbf/CVonline in the section *Image Physics*.

A MATLAB Toolbox for calibrating wide-angle cameras by Davide Scaramuzza is available at https://sites.google.com/site/scarabotix/ocamcalib-toolbox. It is inspired by, and similar in usage, to Bouguet's Toolbox for perspective cameras. Another MATLAB Toolbox, by Juho Kannala, handles wide angle central cameras and is available at http://www.ee.oulu.fi/~jkannala/calibration.

The unified imaging model was introduced by Geyer and Daniilidis (2000) in the context of catadioptric cameras. Later it was shown (Ying and Hu 2004) that many fisheye cameras can also be described by this model. The fisheye calibration of Sect. 11.4.1 was described by Hansen et al. (2010) who estimates ℓ and m rather than a polynomial function $r(\theta)$ as does Scaramuzza's Toolbox.

There is a huge and growing literature on light-field imaging but as yet no textbook. A great introduction to light fields and its application to robotics is the thesis by Dansereau (2014). The same author has a MATLAB Toolbox available at http://mathworks.com/matlabcentral/fileexchange/49683. An interesting description of an early camera array is given by Wilburn et al. (2005) and the associated video demonstrates many capabilities. Light-field imaging is a subset of the larger, and growing, field of computational photography.

Toolbox Notes

The Toolbox camera classes `CentralCamera`, `FishEyeCamera` and `SphericalCamera` are all derived from the abstract superclass `Camera`. Common methods of all classes are shown in Table 11.3.

Table 11.3. Common methods for all Toolbox camera classes

Method	Description
`p = cam.project(P)`	Project the world point `P` to the image plane `p`
`cam.plot(P)`	Plot the world points defined by the columns of `P`
`cam.mesh(X,Y,Z)`	Plot the mesh defined by `X`, `Y` and `Z`
`cam.showpose(T)`	Show the camera at specified pose
`cam.clf`	Clear the current camera view
`cam.hold`	Hold the current camera view, future calls to `plot` add to the camera view
`cam.name`	Property: name of camera
`cam.T`	Property: default camera transform (read and write)

Table 11.4. Common options for camera class constructors

Option	Description
`'name', name`	The name of the camera which is displayed in the window's title bar
`'resolution', npix`	The dimensions of the image. `npix` is a scalar for a square image or a 2-vector
`'centre', pp`	The coordinate of the principal point
`'pixel', rho`	The dimensions of the pixel. `rho` is a scalar for a square pixel or a 2-vector
`'noise', sigma`	The standard-deviation of Gaussian noise added to the image-plane coordinates
`'pose', T`	The default pose of the camera. Default is as shown in Fig. 11.2
`'image', im`	Set camera image-plane dimensions according to the image dimensions and display the image

The virtual camera view has similar behavior to a MATLAB figure. By default `plot` and `mesh` will redraw the camera's view. If no camera view exists one will be created. The methods `clf` and `hold` are analogous to the MATLAB commands `clf` and `hold`.

The constructor of all camera classes accepts a number of option arguments which are listed in Table 11.4. Specific camera subclasses have unique options which are described in the online documentation. With no arguments the default `CentralCamera` parameters are for a 1 024 × 1 024 image, 8 mm focal length lens and 10 μm square pixels. If the principal point is not set explicitly it is assumed to be in the middle of the image plane.

Exercises

1. Create a central camera and a cube target and visualize it for different camera and cube poses. Create and visualize different 3D mesh shapes such as created by the MATLAB functions `cylinder` and `sphere`.
2. Write a script to fly the camera in an orbit around the cube, always facing toward the center of the cube.
3. Write a script to fly the camera through the cube.
4. Create a central camera with lens distortion and which is viewing a 10 × 10 planar grid of points. Vary the distortion parameters and see the effect this has on the shape of the projected grid. Create pincushion and barrel distortion.
5. Repeat the homogeneous camera calibration exercise of Sect. 11.2.1 and the decomposition of Sect. 11.2.2. Investigate the effect of the number of calibration points, noise and camera distortion on the calibration residual and estimated target pose.

6. Determine the solid angle for a rectangular pyramidal field of view that subtends angles θ_h and θ_v.
7. Do example 1 from Bouguet's Camera Calibration Toolbox.
8. Calibrate the camera on your computer.
9. Derive Eq. 11.14.
10. For the camera calibration matrix decomposition example (Sect. 11.2.2) determine the roll-pitch-yaw orientation error between the true and estimated camera pose.
11. Pose estimation (Sect. 11.2.3)
    a) Repeat the pose estimation exercise for different object poses (closer, further away).
    b) Repeat for different levels of camera noise.
    c) What happens as the number of points is reduced?
    d) Does increasing the number of points counter the effects of increased noise?
    e) Change the intrinsic parameters of the camera `cam` before invoking the `estpose` method. What is the effect of changing the focal length and the principal point by say 5%.
12. Repeat exercises 2 and 3 for the fisheye camera and the spherical camera.
13. With reference to Fig. 11.19 derive the function $\psi(\theta)$ for a parabolic mirror.
14. With reference to Fig. 11.19 derive the equation of the equiangular mirror $z(x)$ in the xz-plane.
15. Quadrics
    a) Write a routine to plot a quadric given a 4×4 matrix. Hint use `meshgrid` and `isosurface`.
    b) Write code to compute the quadric matrix for a sphere at arbitrary location and of arbitrary radius.
    c) Write code to compute the quadric matrix for an arbitrary circular cylinder.
    d) Write numeric MATLAB code to plot the planar conic section described by a 3×3 matrix.
16. Project an ellipsoidal or spherical quadric to the image plane. The result will be the implicit equation for a conic – write code to plot the implicit equation.