XI
UNDERSTANDING GEOMETRY

Let's face it, the world is three-dimensional (3D), and vision systems capture only two-dimensional (2D) images. Understanding the projection of the 3D world into 2D images, and how to reverse this projection to recover the 3D structure of the scene, is one of the most important topics in the study of vision (both natural vision and computer vision). Geometry is therefore a fundamental tool in computer vision.

This collection of chapters will cover many aspects of 3D vision:

- **Chapter 38** introduces homogeneous and heterogeneous coordinate systems and how to use them to model geometric transformations.
- **Chapter 39** describes camera models (intrinsic and extrinsic camera parameters) and camera calibration.
- **Chapter 40** describes how to recover 3D information from stereo images.
- **Chapter 41** describes homographies and the application to build image panoramas.
- **Chapter 42** describes how to recover 3D information using only a single image.

The material in this set of chapters will come handy as soon as you have to deal with 3D scenes (i.e., always).

38Representing Images and Geometry

38.1Introduction

Before we dive into the material of this chapter, let's start by questioning the way in which we have been representing images up to now. In most of the chapters, we have represented images as ordered arrays of pixel values, each pixel described by its grayscale value (or three arrays for color images). We will call this representation the **ordered array**.

$$\boldsymbol{\ell} = \begin{bmatrix} \ell[1,1] & \dots & \\ \vdots & \ell[n,m] & \vdots \\ & \dots & \ell[N,M] \end{bmatrix} \tag{38.1}$$

Each value is a sample on a regular spatial grid. In this notation s represents the pixel intensity at one location. This is the representation we are most used to and the typical representation used when taking a signal processing perspective. It makes it simpler to describe operations such as convolution.

However, an image can be represented in different ways, making explicit certain aspects of the information present on the input. What else could we do to represent an image? Another very different, but equivalent, representation is to encode an image as a collection of points, indicating its color and location explicitly. In this case, an image is represented as a **set of pixels**:

$$\{[\ell_i, x_i, y_i] : i\} \tag{38.2}$$

where ℓ_i is the pixel intensity (or color) recorded at location (x_i, y_i). This representation makes the geometry explicit. It might seem like a trivial representation but it makes some operations easy (and others complex). For instance, we can apply geometric transformations easily by directly working with the spatial coordinates. Imagine you want to translate an image, described by equation (38.2), to the right by one pixel, you can do that easily by simply creating the new image defined by the set of translated pixels: $\{[\ell_i, x_i + 1, y_i] : i\}$.

> Representing images as sets of points is very common in geometry-based computer vision. We will use this representation extensively in the following chapters.

Although both previous representations might seem equivalent, the set representation makes it easy to deal with other image geometries where the points are not on a regular array.

That representation can also be extended by representing geometry in different ways such as using homogeneous coordinates, or positional encoding. We will discuss homogeneous coordinates in this chapter.

The previous two representations are not the only ways in which we can represent images. Another option is to represent an image as a continuous function whose input is a location (x, y) and it output is an intensity, or a color, ℓ:

$$\ell(x, y) = f_\theta(x, y) \tag{38.3}$$

This image representation is commonly used when we want to make image priors more explicit. The function f_θ is parameterized by the coefficients θ. This function becomes especially interesting when the parameters θ are different than the original pixel values. They will have to be estimated from the original image. But once learned, the function f should be able to take as input continuous spatial variables.

These three representations induce different ways of thinking about architectures to process the visual input. These representations are not opposed and can be used simultaneously.

The ordered array of pixels is the format that computers take as input when visualizing images. However, a set of pixels is the most common format when thinking about three-dimensional (3D) geometry and image formation. Therefore, it is always important to be familiar with how to transform any representation into an ordered array.

We will start by introducing homogeneous coordinates, an important tool that will simplify the formulation of perspective projection.

38.2Homogeneous and Heteregenous Coordinates

Homogeneous coordinates represent a vector of length N by a vector of length $N + 1$. The transformation rule from heterogeneous to homogeneous coordinates is simply to add an additional coordinate, 1, to the heterogeneous coordinates as shown here:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{38.4}$$

It is the same if the vector has three dimensions:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{38.5}$$

We will refer to conventional Cartesian coordinate descriptions of a point in 2D, such as (x, y), or 3D, such as (x, y, z), as **heterogeneous coordinates** and write with rounded brackets in this chapter. We denote their corresponding **homogeneous coordinate** representations by square bracketed vectors.

August Ferdinand Möbius, a German mathematician and astronomer, introduced homogeneous coordinates in 1827. He also cocreated the famous Möbius strip.

The homogeneous coordinates have the additional rule that all (non-zero) scalings of a homogeneous coordinate vector are equivalent. For example, to represent a 2D point, we have (transforming from heterogeneous to homogeneous coordinates),

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} \tag{38.6}$$

for any non-zero scalar, λ. To go from homogeneous coordinates back to the heterogeneous representation of the point, we divide all the entries of the homogeneous coordinate vector by their last dimension:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \end{pmatrix} \tag{38.7}$$

Figure 38.1 illustrates how homogeneous and heterogeneous coordinates relate to each other geometrically. A point in homogeneous coordinates is scale invariant (any point within the line that passes through the origin translates into the same point in heterogeneous coordinates). We can already see that this is closely related to the operation performed by perspective projection.

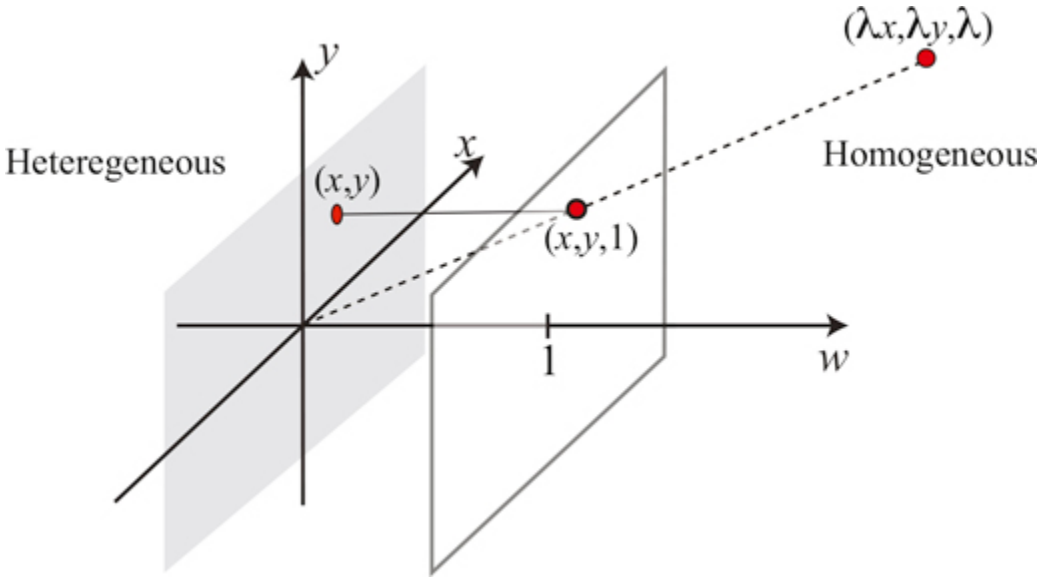


Figure 38.1: Transformation rule between heterogeneous and homogeneous coordinates in 2D. All the points along the dotted line correspond to the same point in heterogeneous coordinates. Note that the points (x, y) and $(x, y, 1)$ live in different spaces.

It is important to mention that while you can add two points in heterogeneous coordinates, you can not do the same in homogeneous coordinates!

> Using homogeneous coordinates, we can place a point at infinity by setting $w = 0$. This is called the **ideal point**.

38.32D Image Transformations

One important operation in computer vision (and in computer graphics) is geometric transformations of shapes. Some of the common transformations we'll want to represent include translation, rotation, scaling, and shearing (see figure 38.2). These transformations can be written as affine transformations of the coordinate system. The mathematical description of these transformations becomes surprisingly simple when using homogeneous coordinates, and they will become the basis for more complex operations such as 3D perspective projection and camera calibration as we will see in later sections.

Figure 38.2:2D geometric transformations. The final transformation is an arbitrary warping with a more complex deformation field.

We'll explore the use of homogeneous coordinates for describing 2D geometric transformations first, then see how they can easily represent 3D perspective projection.

38.3.1Translation

Consider a translation by a 2D vector, (t_x, t_y) as shown in figure 38.3. We'll denote the coordinates after the transformation with a prime. In heterogeneous coordinates, we have

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{38.8}$$

We can write this translation in homogeneous coordinates by the product:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{38.9}$$

What is interesting is that we have transformed an addition into a product by a matrix, **T**, and a vector,

p, in homogeneous coordinates. Rewriting the equation above as:

$$\mathbf{p}' = \mathbf{T}\mathbf{p}, \tag{38.10}$$

with the homogeneous matrix, **T**, defined as

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{38.11}$$

For calculations in homogeneous coordinates, both matrices and vectors are only defined up to an arbitrary (non-zero) multiplicative scaling factor.

To cascade two translation operations **t** and **s**, in heterogeneous coordinates, we have

$$\mathbf{p}' = \mathbf{p} + \mathbf{t} + \mathbf{s} \tag{38.12}$$

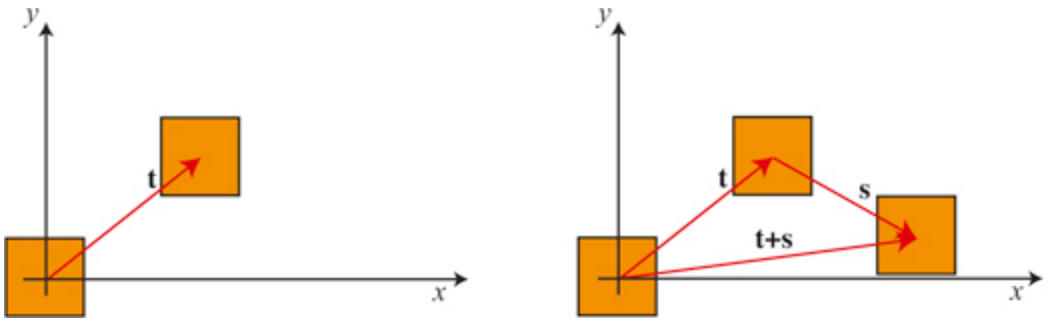


Figure 38.3:(left) Translation. (right) Composition of two consecutive translations.

In homogeneous coordinates, we cascade the corresponding translation matrices:

$$\mathbf{p}' = \mathbf{S}\mathbf{T}\mathbf{p}, \tag{38.13}$$

where the homogeneous translation matrix, **S**, corresponding to the offset **s**, is:

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} \tag{38.14}$$

You can check that:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x + s_x \\ 0 & 1 & t_y + s_y \\ 0 & 0 & 1 \end{bmatrix} \tag{38.15}$$

In summary, in homogeneous coordinates a translation becomes a product with a matrix, and chaining translations can be done by multiplying the matrices together. The benefits of using the

homogeneous coordinates will become more obvious later.

38.3.2Scaling

Scaling the x-axis by s_x and the y-axis by s_y, shown in figure 38.4, yields the transformation matrix in homogeneous coordinates,

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{38.16}$$

The scaling matrix is a diagonal matrix.

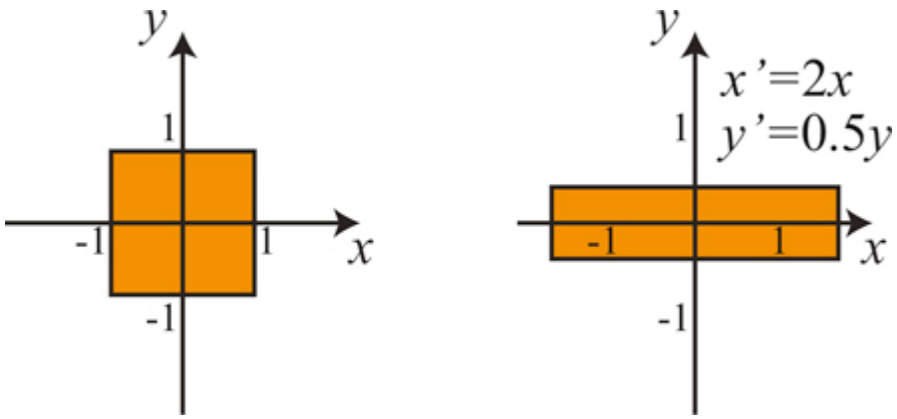


Figure 38.4: Non-uniform or anisotropic scaling. In this example the y-axis is compressed and the x-axis is expanded.

Uniform scaling is obtained when $s_x = s_y$, otherwise the scaling is nonuniform or anisotropic. After uniform scaling, all of the angles are preserved. In all cases, parallel lines remain parallel. Areas are scaled by the determinant of the scaling matrix.

38.3.3Rotation

For a rotation by an angle θ, figure 38.5, we simply have the matrix in homogeneous coordinates:

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{38.17}$$

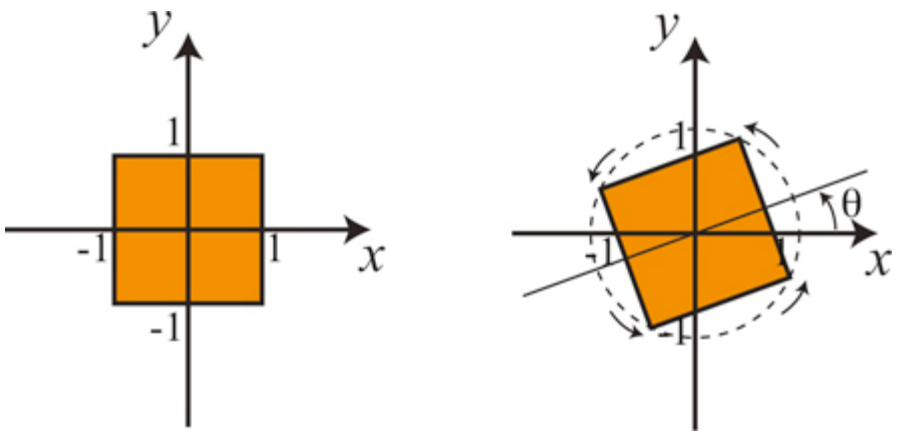


Figure 38.5: Rotation by an angle θ counterclockwise around the origin.

As in the case of the translation, a rotation in homogeneous coordinates is a product

$$\mathbf{p}' = \mathbf{R}\mathbf{p}, \tag{38.18}$$

Also, as we should expect, chaining two rotations with angles α and β, is the same than applying a rotation with an angle $\alpha + \beta$. You can check that multiplying the two rotation matrices you get the right transformation.

The determinant of a rotation matrix is 1 and the matrix is orthogonal, that is the transpose is equal to the inverse: $\mathbf{R}^{\mathsf{T}} = \mathbf{R}^{-1}$. The inverse is also a rotation matrix. The distance between any point and the origin does not change after a rotation.

In heterogeneous coordinates, the transformation can also be written in the same way but using only the upper-left 2×2 matrix. Representing rotations in homogeneous coordinates has no benefit with respect to heterogeneous coordinates. But the advantage is that now both rotation and translation are written in the same way! They are both products of a matrix times a vector, so that they can be combined as we will discuss in section 38.3.5.

If the angle of rotation is very small, then we can approximate the rotation matrix by its Taylor development (for small x, $\sin(x) \approx x$ and $\cos(x) \approx 1$):

$$\mathbf{R} \simeq \begin{bmatrix} 1 & \theta & 0 \\ -\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{38.19}$$

For small angles a rotation becomes a shear, which we will discuss next. In general, for all angles, a rotation can be written as two shears and a scaling.

Rotations in 3D become more complex as there are multiple possible parametrizations.

38.3.4Shearing

Shearing involves scale factors in the off-diagonal matrix locations, as shown in the matrix, **Q**:

$$\mathbf{Q} = \begin{bmatrix} 1 & q_x & 0 \\ q_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{38.20}$$

Figure 38.6 shows examples with an horizontal shear ($qx = 1$, $q_y = 0$), a vertical shear ($q_x = 0$, $q_y = 1$), and an arbitrary shear.

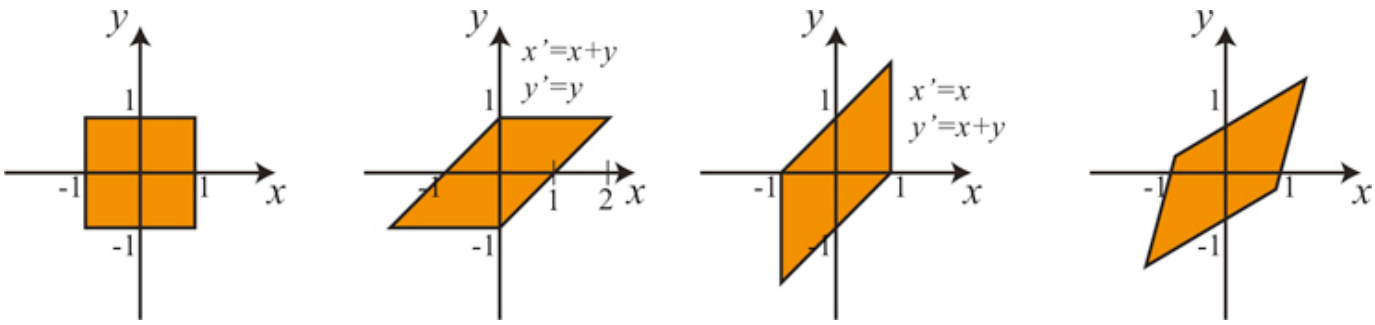


Figure 38.6: Three examples of shear transformation.

In the example of the horizontal shear, the points are displaced along horizontal lines by displacement proportional to the y coordinate, $x' = x + q_y y$.

In a shear, lines are mapped to lines, parallel lines remain parallel, (non-zero) angles between lines change.

38.3.5Chaining Transformations

As we have seen, homogeneous coordinates allows using all the four different transformations as matrix multiplications. We can now build complex transformations by combining these four transformations:

$$\mathbf{p}' = \begin{bmatrix} 1 & q_x & 0 \\ q_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p} \tag{38.21}$$

In heterogeneous coordinates the translation will have to be modeled as an addition breaking the homogeneity of this equation. The transformations described in this section are summarized in figure 38.7.

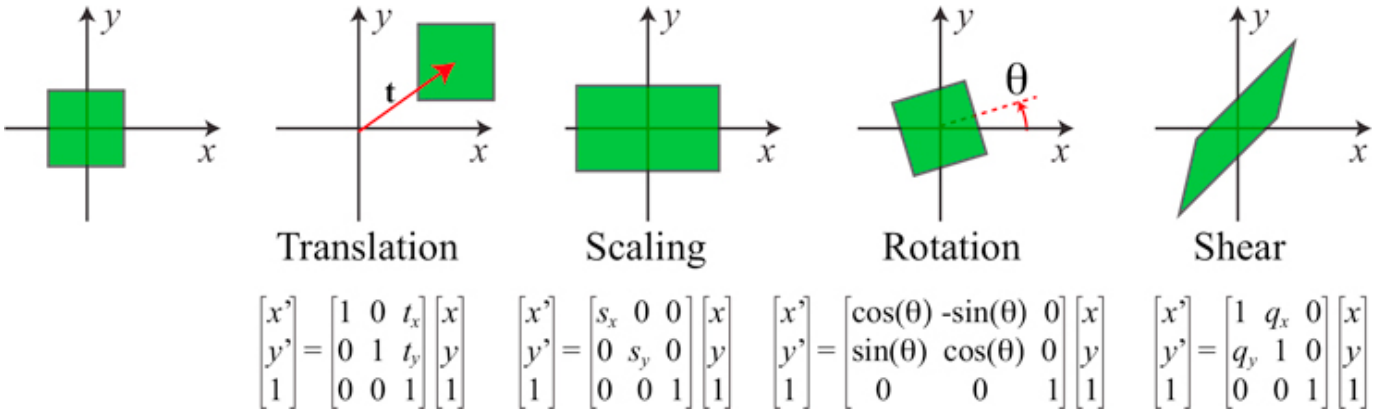


Figure 38.7: Summary of 2D geometric transformations and their formulation in homogeneous coordinates.

As matrix operation are noncommutative, the order in which operations are done is important (i.e., it is not the same to rotate with respect to the origin and then translate, as it is to translate and then rotate). All of the geometric transformations we have described are relative to the origin. If you want to rotate an image around an arbitrary central location, then you need to first translate to put that location at the origin, then rotate and then translate back.

$$\mathbf{p}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p} \tag{38.22}$$

Chaining transformations in such a way is a very important tool in computer graphics and we will also use it extensively as we dive deeper into geometry.

When chaining rotations and translations only we will have a **euclidean transformation** (lengths and angles between lines are preserved). A **similarity transform** is the result of chaining a rotation, translation and scaling (with uniform scaling, $s_x = s_y$). In this case angles are preserved but not lengths. Chaining all transformations results in an **affine transformation**. Each set of transformations forms a group.

38.3.6Generic 2D Transformations

In general, chaining translations, scalings, rotations and shears will result in a generic matrix with the form:

$$\mathbf{p}' = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p} \tag{38.23}$$

Any transformation with that form (6 degrees of freedom) is an affine transformation. An affine transformation has the property that parallel lines will remain parallel after the transformation; however, lengths and non-zero angles might change.

As the last row of the transformation is [0, 0, 1], one could be tempted to drop it and go directly from homogeneous to heterogeneous, using the top 2 × 3 matrix, but this will only work if the input vector has a 1 in the third component.

What happens if we have 9 degrees of freedom?

$$\mathbf{p}' = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \mathbf{p} \tag{38.24}$$

In fact we only have 8 degrees of freedom as a global scaling of the matrix does not change the homogeneous coordinates. The set of transformations described by this full matrix becomes more general than the transformations described in the previous sections. The additional degrees of freedom include **elations** and **projective transformations**.

38.3.7Geometric Transformations as Convolutions

In chapter 15 we showed how certain geometric transformations can be written as convolutions (such as the translation) while others cannot (such as rotations, scalings, shears, etc.). However, things change when adding geometry explicitly to the image representation!

Once geometry is added to the representation, all of the transformations we discussed before can be implemented as one-dimensional (1D) convolutions over the locations as shown in the diagram in figure 38.8.

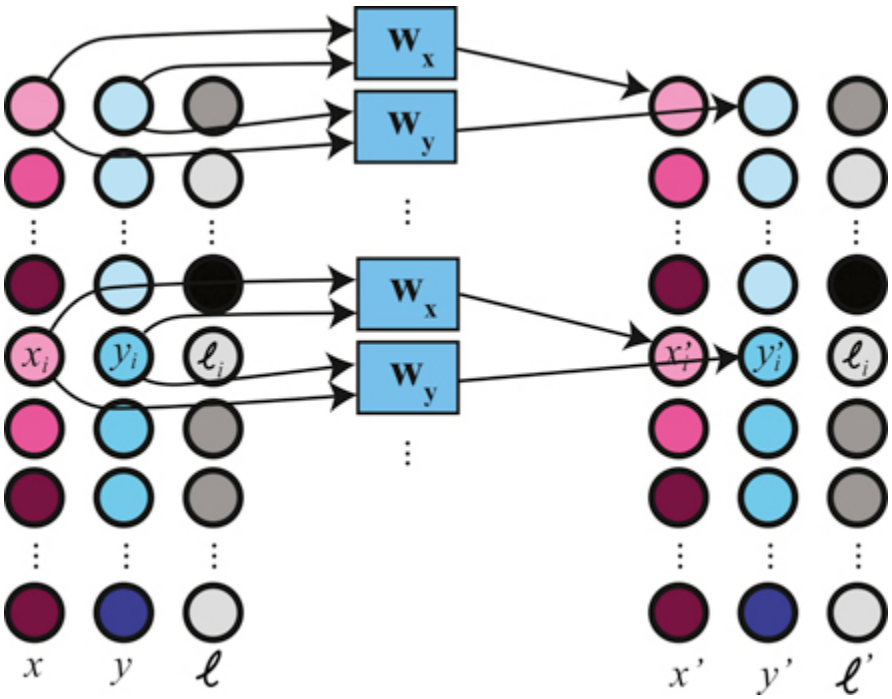


Figure 38.8: Geometric transformation as a convolution. The input and output signals are represented as pixel sets with explicit geometry. The convolution kernels are w_x and w_y, corresponding to one dimensional convolutions as they only mix input features within the same input vector. The output intensity values are the same as the input.

In the diagram (figure 38.8), each input element is a vector (x_i, y_i, ℓ_i) where x_i, y_i are the pixel location, and ℓ_i is the pixel intensity at that location. The convolution kernels are: $w_x = [\cos(\theta), \sin(\theta)]$ and $w_y = [-\sin(\theta), \cos(\theta)]$. The weights are the same for all inputs. The output is also represented using position explicitly: (x'_i, y'_i, ℓ'_i).

However, note that to perform a convolution on the output intensity will require translating the position encoding back into an image on a rectangular grid. For instance, after a rotation, the locations for the intensity values will change and the pixels will not lie on a rectangular grid anymore. The convolution kernels for the intensity channel will have to be transformed too.

38.4Lines and Planes in Homogeneous Coordinates

One interesting application of homogeneous coordinates is to use it to describe lines and planes and perform operations with them. In 2D, the equation of a line is $ax + by + c = 0$, which can be written in homogeneous coordinates as

$$ax + by + c = 0 \rightarrow \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \tag{38.25}$$

In homogeneous coordinates, the equation of a line is the dot product:

$$\boldsymbol{l}^{\mathsf{T}} \mathbf{p} = 0 \tag{38.26}$$

where $\boldsymbol{l}^{\mathsf{T}} = [a, b, c]$. Therefore, a point $\mathbf{p}$, belongs to the line when $\boldsymbol{l}$ and $\mathbf{p}$ are perpendicular.

This representation of the line is in homogeneous coordinates because it is scale invariant. This is, $[a, b, c]$ is the same line as $[a/c, b/c, 1]$. Therefore, it is also useful to describe the equation of the line with $\boldsymbol{l}^{\mathsf{T}} = n_x, n_y, -d$ where (n_x, n_y) is the normal to the line and d is the distance to the origin.

Using homogeneous coordinates makes obtaining geometric properties of points and lines very easy. Given two points $\mathbf{p}_1$ and $\mathbf{p}_2$ in homogeneous coordinates, the line that passes by both points is the cross product (figure 38.9):

$$\boldsymbol{l} = \mathbf{p}_1 \times \mathbf{p}_2 \tag{38.27}$$

This is because if the line passes by both points, it has to verify that $\boldsymbol{l}^{\mathsf{T}}\mathbf{p}_1 = 0$ and $\boldsymbol{l}^{\mathsf{T}}\mathbf{p}_2 = 0$. That is, the vector $\mathbf{l}$ has to be perpendicular to both $\mathbf{p}_1$ and $\mathbf{p}_2$. The cross product between $\mathbf{p}_1$ and $\mathbf{p}_2$ gives a vector that is perpendicular to both.

Following a similar argument, you can show that given two lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ the intersection point between them is the cross product (figure 38.9):

$$\mathbf{p} = \boldsymbol{l}_1 \times \boldsymbol{l}_2 \tag{38.28}$$

The coordinates of **p** computed that way will be given in homogeneous coordinates. So you need to divide by the third component in order to get the actual point coordinates in heterogeneous coordinates.

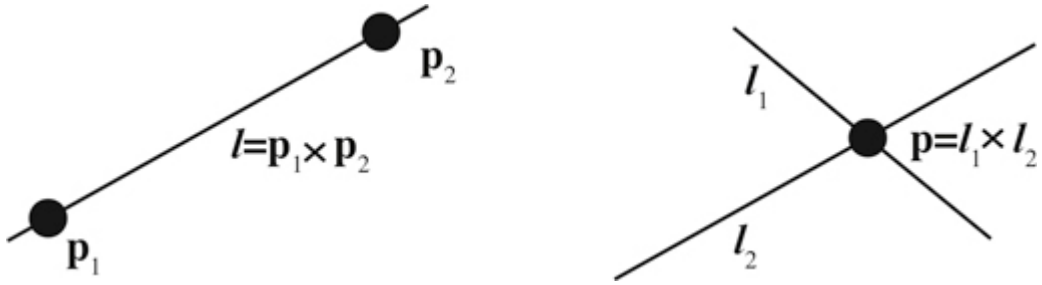


Figure 38.9: Using homogeneous coordinates to get the line that passes by two points, and to obtain the intersection of two lines. Both operations are analogous.

If three 2D points are colinear, then the determinant of the matrix formed by concatenating the three vectors, in homogeneous coordinates, as columns is equal to zero: $\det([\mathbf{p}_1\ \mathbf{p}_2\ \mathbf{p}_3]) = 0$. If three lines intersect in the same point we have a similar relationship: $\det([\boldsymbol{l}_1\ \boldsymbol{l}_2\ \boldsymbol{l}_3]) = 0$.

It is also interesting to point out that a 3D vector can be interpreted as a 2D line or as a 2D point in homogeneous coordinates.

We can also do something analogous to represent planes in 3D. The equation of a 3D plane is $aX + bY + cZ + d = 0$, which can be written in homogeneous coordinates as:

$$\pi^{\mathsf{T}}\mathbf{P} = 0 \tag{38.29}$$

where $\pi = [a, b, c, d]^{\mathsf{T}}$ are the plane parameters and $\mathbf{P} = [X, Y, Z, 1]^{\mathsf{T}}$ are the 3D point homogeneous coordinates.

Representing 3D lines with homogeneous coordinates is not that easy and the reader can consult other sources [187] to learn more about representing geometric objects in homogeneous coordinates.

38.5Image Warping

Now that we have seen how to describe simple geometric transformations to pixel coordinates, we need to go back to the representation of the image as samples on a regular grid. This will require applying image interpolation.

The first algorithm that usually comes to mind when transforming an image is to take every pixel in the original image represented as $[\ell_i, x_i, y_i]$, apply the transformation, **M**, to the coordinates, and record the pixel color into the resulting coordinates in the target image (figure 38.10). As coordinates might result in non-integer values, we can simply round the result to the closest pixel coordinate (i.e.,

nearest neighbor interpolation as we discussed in section 21.4.1). This algorithm is called **forward mapping**. It is an intuitive way of warping an image but it is really not a good idea. We will have all sorts of artifacts such as missing values and aliasing as shown in figures 38.10 and 38.11.

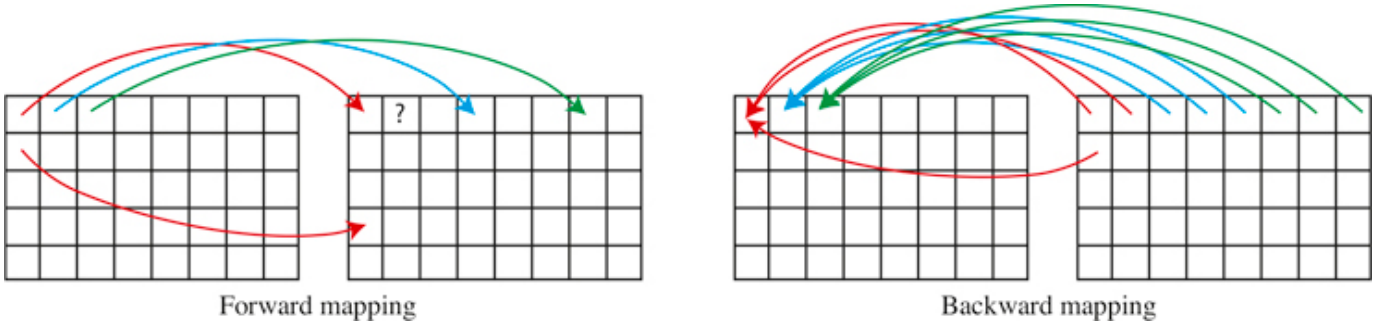


Figure 38.10: Comparison between forward and backward mapping using nearest neighbor interpolation. Forward mapping will produce missing values.

The best approach is to use what is called **backward mapping** which consists of looping over all the pixels of the target image and applying the inverse geometric transform, $\mathbf{M}^{-1}$; we then use interpolation (as described in section 21.4.1 in chapter 21) to get the correct color value (figure 38.10). This process guarantees that there will be no missing values (unless the coordinates go outside the frame of the input image) and there will be no aliasing if the interpolation is done correctly. To avoid aliasing, blurring of the input image might be necessary if the density of pixels in the target image is lower than in the input image. Figures 38.10 and 38.11 compare forward and backward mapping.

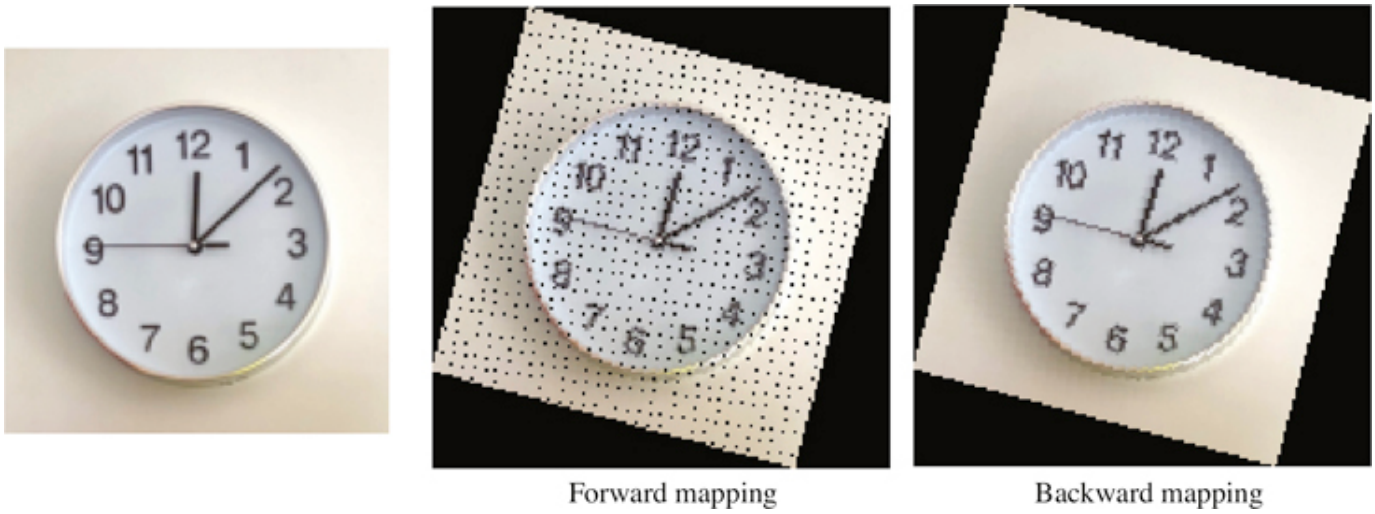


Figure 38.11: Comparison between forward and backward mapping. Forward mapping produces many artifacts. In both cases we use nearest neighbor interpolation.

To achieve high image quality warping it is important to chose a high quality interpolation filter such as bilinear, bicubic, or Lanczos. MIP-mapping [504] is another popular technique for high quality

and efficient interpolation. MIP-mapping relies on a multiscale image pyramid to efficiently compute the best neighborhood structure needed to perform the interpolation at each location, which can be very useful when warping an image onto a curved surface.

Image warping can be applied to arbitrary geometric transformations and not just the ones described in this section.

38.6Implicit Image Representations

An image is an array of values, $\boldsymbol{\ell}^{N\times M\times 3}$, where each value is indexed as $\ell[n, m]$ when n, m take only on discrete values. We can say that interpolation is a way of transforming the discrete image into a continuous signal: $\ell(x, y)$.

An implicit image representation via a function f_θ trained to reproduce the image pixels is a function such that,

$$\ell(x, y) = f_\theta(x, y) \tag{38.30}$$

where now x, y can take on any real value.

For this representation to work better, the input location is usually first transformed using **positional encoding**, and the final network is more complex than the one shown here. We will discuss this type of representations more in depth in chapter 45.

38.6.1Interpolation

In the case of nearest neighbors or bilinear interpolation, the parameters of the interpolation function θ is the input image itself. For example, using a functional form, nearest neighbors interpolation can be written as:

$$\ell(x, y) = \ell\left[\text{round}(x), \text{round}(y)\right] \tag{38.31}$$

What is really interesting about thinking about interpolation in this way is that we can now extend the space of possible functions $f_\theta(x, y)$ to include other functional forms. For instance, this function could be implemented by a neural network that will take as input the two image coordinate values x and y and will output the intensity value at that location. The training set for the neural network is the image itself, and it will consist of the input-output pairs $[(x_i, y_i); \ell(x_i, y_i)]$ (i.e., location as input and intensities/colors as output). During training the neural network will memorize the image. The training will contain only values at discrete positions but in test time we can use any continuous input values. For this formulation to work, the neural network should be able to generalize to non-integer coordinate values, that is, it should be able to interpolate between samples.

38.6.2Image Warping with Implicit Representations

Once the neural network, $f_\theta(x, y)$, has learned to reproduce the image, we can reconstruct the original image or apply transformations to it. Image warping is then implemented by simply applying the inverse geometric transformation to the discrete coordinates of the output grid and use the functional representation of the image to get the interpolated values:

$$\hat{\ell}(x,y) = f_\theta\left(\mathbf{M}^{-1}(x,y,1)^\mathsf{T}\right) \tag{38.32}$$

In this equation the input location is written in homogeneous coordinates, so the function f will first have to divide by the third component to translate the input back to heterogeneous coordinates.

The example in figure 38.12 shows an image encoded by a sinusoidal representation network (SIREN) [445] and then reconstructed with a rotation of 45 degrees and a scaling by 0.5 along both dimensions.

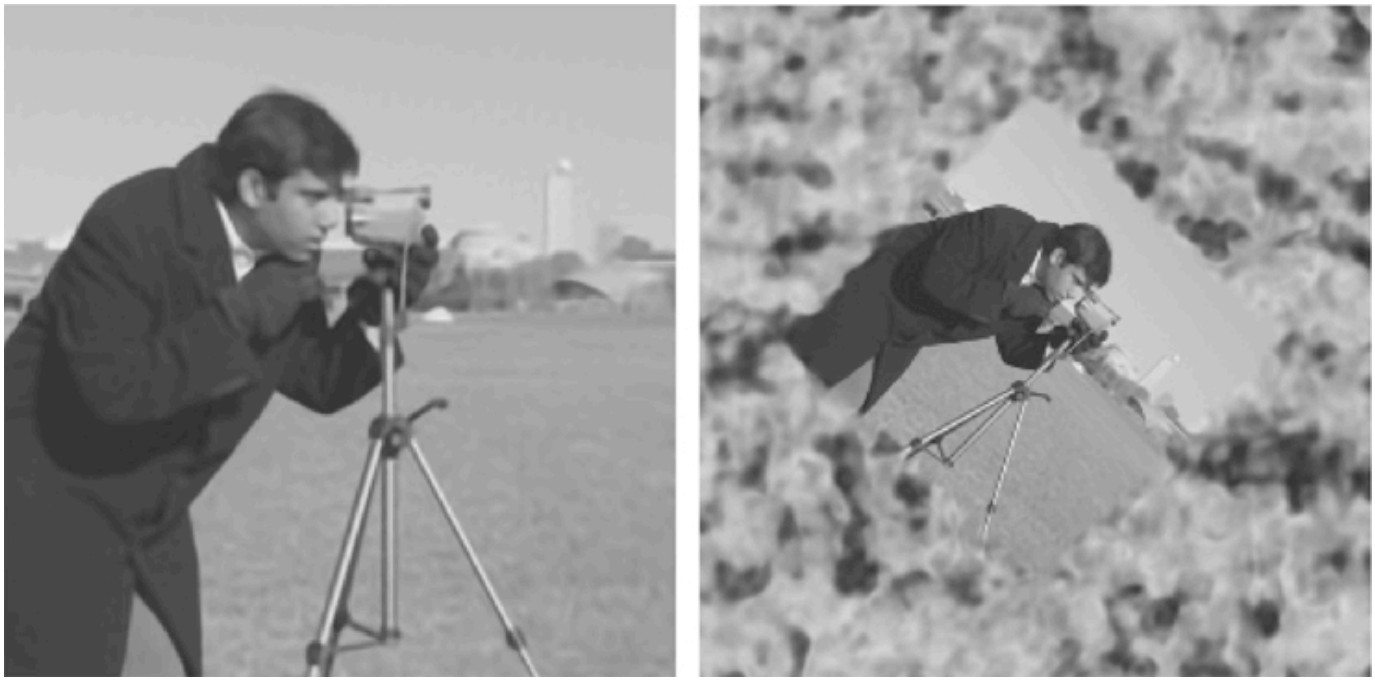


Figure 38.12:(left) Image encoded by SIREN. (right) Image reconstructed with a rotation of 45 degrees and a scaling by 0.5 along both dimensions.

From this result we can make a few observations. First, we can see that, due to the transformation, there is aliasing in the sampled image (aliasing is most visible in the leg of the tripod). One way of avoiding aliasing would be by sampling the output on a finer grid and then blurring and downsampling the result to the desired image size. The second observation is that the way the boundary is extended is not like any of the methods that we studied in chapter 15; instead the image is padded by some form of noise that smoothly extends the image without adding strong image derivatives.

We can also study the reverse problem where we have two images (one is a transformed version of

the other) and the goal is to identify the transformation **M**. This problem is called **image alignment**.

The spatial transformer network [233] is an example of using parametric image transformations inside a neural network. Such transformation can be helpful during learning to align the training examples into a canonical pose.

38.7Concluding Remarks

Homogeneous coordinates are extensively used in computer vision and computer graphics. They allow simplifying the computation of geometric image transformations. Therefore, many libraries in vision and graphics assume that the user is knowledgeable about the different coordinate systems.

One of the most important uses is in the formulation of perspective projection. We will devote the next chapter to describing the image formation process and camera models using homogeneous coordinates.

Representing images as collections of pixels with an explicit representation of geometry has a long history and is at the center of many modern methods for 3D image representation.

39Camera Modeling and Calibration

39.1Introduction

In chapter 5 we described the image formation process assuming that the camera was the central element and we placed the origin of the world coordinates system in the pinhole. But, as we will be interested in interpreting the scene in front of the camera, other systems of reference might be more appropriate. For instance, in the simple world model of chapter 2 we placed the origin of the world coordinates system on the ground, away from the camera. What formulation of the image formation process will allow us changing coordinate systems easily? This is what we will study in this chapter.

In the sketch shown below (figure 39.1), a standing person is holding a camera and taking a picture with it of someone sitting at a table (sorry for our drawing skills but hopefully this description compensates for our poor artistry). We will be answering questions about the scene based on the picture taken by the camera: for instance, how high and large is the table? We will also answer questions about the camera, like how high is the camera above the floor?

Figure 39.1: Camera-centric and world-centric camera coordinate systems.

In settings where we have multiple cameras we will have to be able to transform the coordinates frames between cameras. And finally, to translate the two-dimensional (2D) images into the underlying three-dimensional (3D) scene, we need to know how 2D points relate to 3D world coordinates, which is called calibrating the cameras.

In this chapter we will show how to use homogeneous coordinates to describe a camera projection model that is more general than what we have done until now. This formulation will be key to building most of the approaches we will study from now until the end of this book; therefore it is important to be familiar with it. It is also often used both in classic computer vision approaches and in deep learning architectures.

But before we move into the material, we are sure you would like to see the picture that the standing character from the previous sketch took. Here it is:

Figure 39.2: The picture taken by the standing character from figure 39.1.

39.23D Camera Projections in Homogeneous Coordinates

Let's start this chapter with the most important application of homogeneous coordinates for us: describing perspective projection. For now, we will continue assuming that the origin of the world coordinates system is at the pinhole location and that the Z-axis corresponds to the optical axis (i.e., it is perpendicular to the image plane).

As we have already discussed (see figure 39.3 to refresh your memory), the perspective projection equations are nonlinear and involve a division by the point's Z-location. That is, a 3D point with coordinates $[X, Y, Z]^T$ appears in the image at coordinates $[x, y]^T$ where $x = f X/Z$ and $y = f Y/Z$ (f is the focal length). This division complicates algebraic relations. Projective geometry and homogeneous coordinates are tools that simplify those algebraic relations, and thus simplify many of the equations that involve geometric transformations.

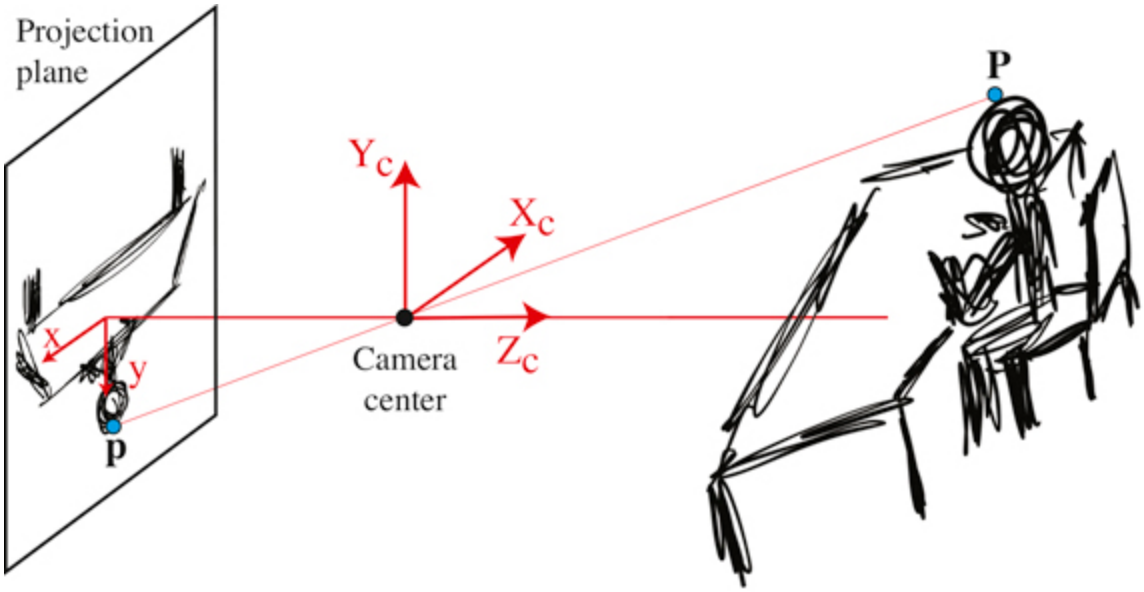


Figure 39.3: Perspective projection. Remember that from similar triangles, we have $x/f = X/Z$ and $y/f = Y/Z$.

To write points in 3D space in homogeneous coordinates, we add a fourth coordinate to the three coordinates of Euclidean space, as described in chapter 38. Using homogeneous coordinates, camera projections can be written in a simple form as a matrix multiplication. Consider a coordinate system with the origin fixed at the center of projection of the camera and the 3D point in homogeneous coordinates, $\mathbf{P} = [X, Y, Z, 1]^{\mathsf{T}}$.

We can verify that the matrix, **K**, shown below, multiplied by the vector describing the 3D point, **P**, returns the position of its perspective projection (equation [5.7]):

$$\mathbf{KP} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{39.1}$$

Transforming the product vector from homogeneous coordinates to heterogeneous coordinates yields

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} \rightarrow \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix} = \begin{pmatrix} cx \\ y \end{pmatrix} = \mathbf{p} \tag{39.2}$$

showing that the matrix **K**, when multiplied by the homogeneous coordinates of a 3D point, **P**, renders the homogeneous coordinates of the perspective projection of that 3D point, **p**. This formulation is one of the biggest benefits of using homogeneous coordinates. It allows writing perspective projection as a linear operator in homogeneous coordinates.

Because homogeneous coordinates, and therefore also the transformation matrices, are only defined up to an arbitrary uniform (non zero) scale factor, the perspective projection transformation

matrix is equivalently written as

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \tag{39.3}$$

Also, in this particular case, it is fine to drop the last column of **K** and use the heterogeneous coordinates form of **P**. However, it is interesting to keep the homogeneous formulation, as it will allow us to work with more general forms of camera models, as we will see next.

39.2.1Parallel Projection

We can use homogeneous coordinates to describe a wide variety of camera models. For instance, the projection matrix for an orthographic projection (equation [5.10]), **K**, is

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{39.4}$$

as can be verified by multiplying by a 3D point in homogeneous coordinates.

39.3Camera-Intrinsic Parameters

Now that we have seen how 3D to 2D projection can be effectively modeled using homogeneous coordinates and simple vector computations, let's apply these tools to model cameras.

We want to construct a camera model that captures the image formation process. Cameras translate the world into pixels, but the relationship between the 3D scene and the 2D image captured by the camera is not always straightforward. To start with, the world is measured in units of meters while points in images are measured in pixels. Factors such as geometric distortions can influence this relationship, and we need to account for these elements in our camera model.

39.3.1From Meters to Pixels

Figure 39.4 illustrates the image formation process and the image sampling by the sensor. We need to account for the perspective projection of the point, **P**, described in camera coordinates, to pixel coordinates in the sensor plane of the camera.

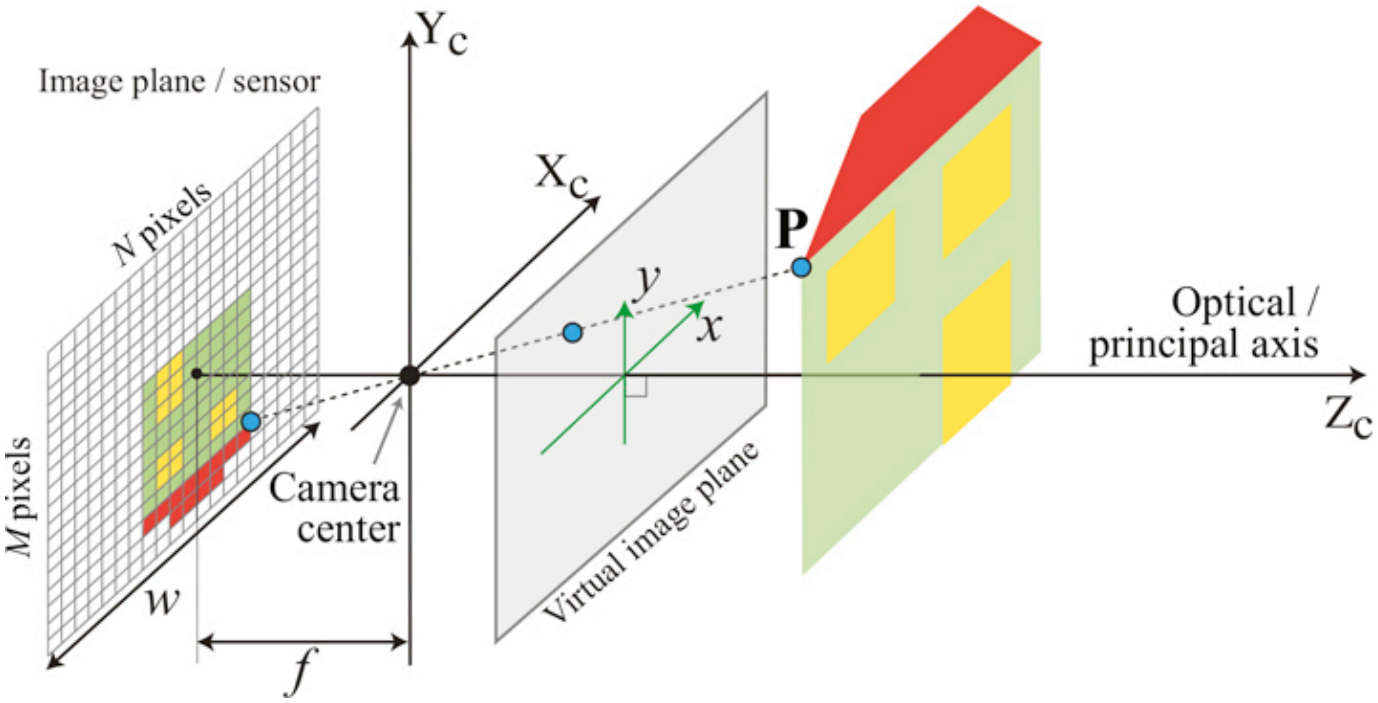


Figure 39.4: An image is projected into the sensor. World coordinates are transformed into pixels at the sensor. The focal length is f, and the physical width of the sensor is w. The sensor has $N \times M$ pixels.

The units of the position in pixel space may be different than those of the world coordinates, which scales the focal length, f, in equation (39.3) to some other value. Let's call that value a:

$$\mathbf{K} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.5}$$

where the constant a is related to the physical camera parameters in the following way:

$$a = f\, N/w \tag{39.6}$$

where f is the focal length (in meters), w is the width of the sensor in meters, and N is the image width in pixels. The ratio N/w is the number of pixels per meter in the sensor.

Also, the image center might not be the pixel $[0, 0]^\mathsf{T}$. If the optical axis coincides with the image center, we need to apply a translation so that a 3D point on the optical axis (i.e., $X = Y = 0$) projects to $[c_x, c_y]^\mathsf{T}$. For example, for an image of size $N \times M$ we will have $c_x = N/2$ and $c_y = M/2$:

$$\mathbf{K} = \begin{bmatrix} a & 0 & c_x & 0 \\ 0 & a & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.7}$$

It is also possible that the pixels are not square, in which case we have to account for different scaling along both axis:

$$\mathbf{K} = \begin{bmatrix} a & 0 & c_x & 0 \\ 0 & b & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.8}$$

Then, the final transformation from 3D coordinates, $[X, Y, Z]^\mathsf{T}$, to image pixels, $[n, m]^\mathsf{T}$, has the form:

$$\begin{bmatrix} a & 0 & c_x & 0 \\ 0 & b & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{pmatrix} aX/Z + c_x \\ bY/Z + c_y \end{pmatrix} = \begin{pmatrix} n \\ m \end{pmatrix} \tag{39.9}$$

The signs of a and b can be positive or negative. With the convention used in this book, where the camera optical axis coincides with the Z-axis and the camera looks in the direction of positive Z, the sign of a has to be negative. If n and m are indices into an array, then the image origin is located at the top-left of the array, which means the signs of a and b will be negative. This is the convention used by Python and OpenCV. Figure 39.5 shows two different common conventions.

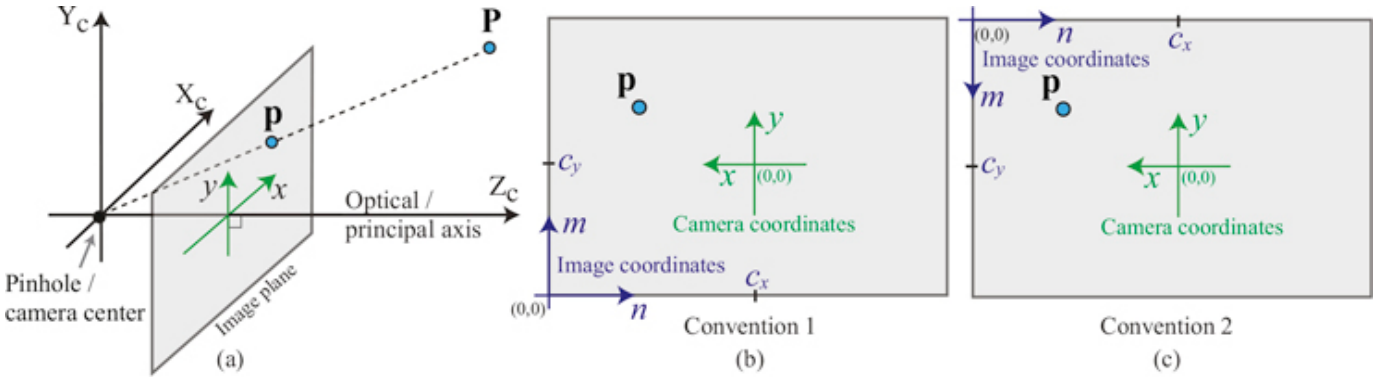


Figure 39.5:(a) 3D representation of the image plane. (b and c) Two different conventions for the image coordinate systems. In this book we have been using (b).

39.3.2From Pixels to Rays

If you have a calibrated camera, can we derive the equation of the ray that leaves the image at one point? Answering this question, will allow us answering the following one: If we know Z for an image pixel with camera-coordinates x, y, can get the point world-coordinates X and Y?

We can relate the point in the image plane, **p**, with the 3D point **P** in world-coordinates. To do this, we will first play a little trick: We will express the 2D point **p** in 3D coordinates. We can do this by realizing that the point **p** is contained in the image plane which is located at a distance f of the origin along the Z-axis (figure 39.6). Therefore, in 3D, the 2D point **p** is a point located at coordinates $Z_c = f$, where f is the focal length, $X_c = x$, and $Y_c = y$, as illustrated in figure 39.6.

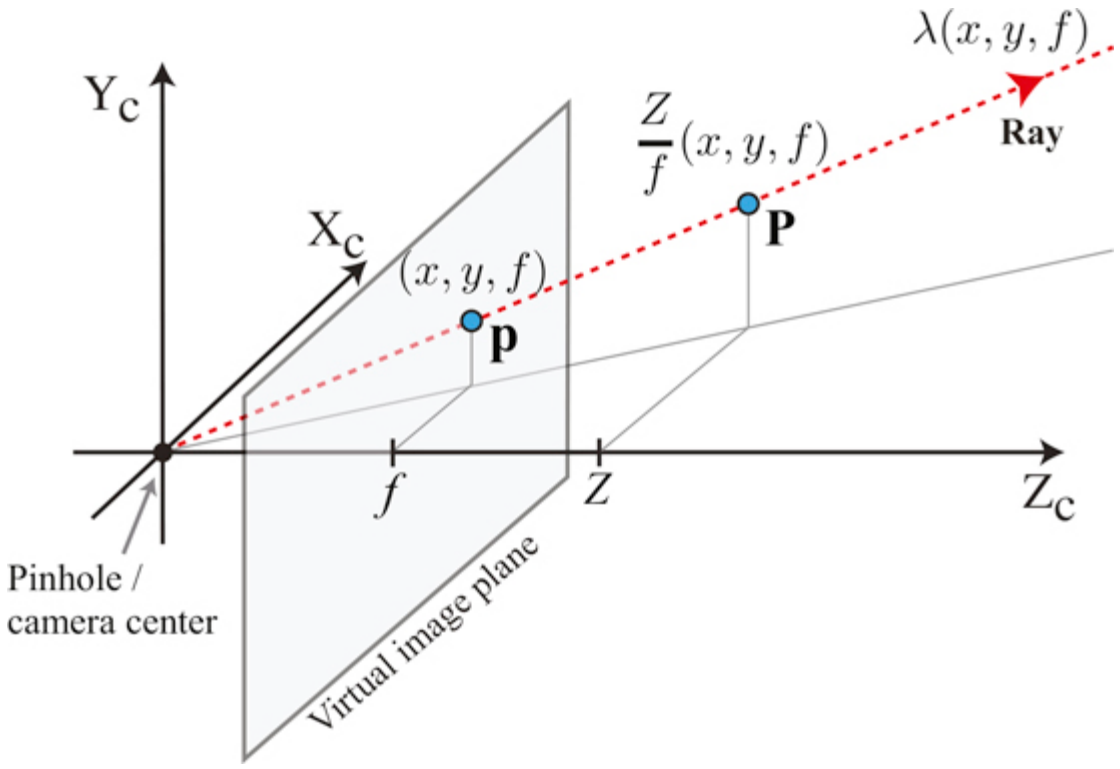


Figure 39.6: The 3D point **P** is obtained by scaling the point $\mathbf{p} = (x, y, f)$ with a scaling factor Z/f. The ray that passes by the point **p** is the line defined by $\lambda(x, y, f)$, with λ being a positive real number.

The ray that connects the camera center and the point **p** is $\lambda(x, y, f)$, with λ being a positive real number. Any 3D point along this line will project into the same image point **p** under perspective projection.

As shown in figure 39.6 the 3D point **P** is obtained by scaling the ray that passes by **p**. The scaling factor is $\lambda = Z/f$:

$$\frac{Z}{f}\begin{pmatrix} x \\ y \\ f \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{39.10}$$

We should use a instead of f if we are using coordinates in pixels. We will revisit this again in chapter 43.

39.3.3A Simple, although Unreliable, Calibration Method

Everything we described up to here, relies on a series of parameters (i.e., focal length, sensor size) that will be unknown in general. Before we continue, let's try to see if we can find out what those parameters are in a real setting.

Let's start by describing a simple and intuitive method for camera calibration illustrated in figure 39.7. The method we will describe is not very accurate, but it provides be a good sanity check before

doing more precise camera calibrations. We will describe a more accurate method in section 39.7.

(a) (b)

Figure 39.7: A simple calibration setting.

To calibrate a camera we need the following three ingredients:

- An object of known size: In this setting, figure 39.7(a), the object is a 8 × 10 chessboard pattern printed so that each square exactly measures 2 × 2 cm. The total width is $W = 20$ cm. This chessboard is a standard calibration target [526].
- The distance between the camera and the object: We use a ruler to measure that the distance between the camera and the chessboard as shown in figure 39.7(a). The distance is approximately $Z = 31$ cm.
- A picture of the object: Figure 39.7(b) shows the picture taken by the camera shown in figure 39.7 (a). The camera plane is parallel to the chessboard. From the picture, we can measure the width of the chessboard in pixels, which is $L = 2{,}002$ pixels.

These three quantities, W, L and Z, are related via the parameter a of the projection matrix by the equation: $a = ZL/W$, which results in:

$$a = \frac{31 \times 2{,}002}{20} = 3{,}103.1 \tag{39.11}$$

The picture size is 4,032 × 3,024 pixels which we can use to get the other two parameters of the camera projection matrix: $c_x = 3{,}024/2$ and $c_y = 4{,}032/2$. Putting all together we get the projection matrix:

$$\mathbf{K} = \begin{bmatrix} 3{,}103.1 & 0 & 1{,}512 & 0 \\ 0 & 3{,}103.1 & 2{,}016 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.12}$$

Would it be possible to infer the intrisic camera matrix from the camera technical specifications alone? Can we directly estimate the parameter a directly from the camera specs? The picture is taken with the wide-angle lens of an iPhone 13 pro. The focal length is 5.7 mm, the sensor size is 7.6 × 5.7 mm, and the image size is 4,032 × 3,024 pixels. This gives as an intrinsic parameter of:

$$a = \frac{4{,}032 \times 5.7}{7.6} = 3{,}024. \tag{39.13}$$

It works! (Does it?) Or at least it is close. But a lot of things happen inside a camera and estimating these values from hardware specs might be difficult.

But is this quality for the calibration enough?

39.3.4Other Camera Parameters

Unfortunately, there are other important aspects of a camera that require precise calibration. In some rare cases, pixels might not be square, or might be arranged along non-perpendicular axes, which requires introducing a skew parameter in **K**. Radial distortion introduced by lenses is often an important and frequent source of issues that requires calibration. When there is radial distortion, straight lines will appear curved. In that case, it is very important to correct for the distortion. Standardized tools for camera calibrations take into account all these different aspects of the camera in order to build an accurate camera model. A more in-depth description of these tools and methods can be found here [526, 525].

39.4Camera-Extrinsic Parameters

In all our previous examples, we have been assuming that the camera origin was located at the origin of the world coordinate system with the optical axis aligned with the world Z-axis. In practice, there will be many situations where placing the camera at a special position will be more convenient. For instance, in the simple vision system we studied in chapter 2 the camera center was placed above the Z -axis. Let's now study a more general setup where the camera is placed at an arbitrary location away from the world coordinate system.

In the examples shown in figure 39.8, we are interested in placing the world coordinate system so that the origin is on the ground and the axes Z_w and X_W are contained on the ground plane, and parallel to the two axis defined by the table. The axis Y_W is perpendicular to the ground. Let's say that our camera has its camera center displaced by vector **T** and the axes are rotated by $\mathbf{R}^\mathsf{T}$ with respect to the world-coordinates system, as shown in figure 39.8. Precisely, this is done by first placing the camera at the origin of the world-coordinates system, then rotating the camera with the rotation matrix $\mathbf{R}^\mathsf{T}$ and finally translating it by **T** (the order of these two operations matters). We use the inverse of **R** (equal to its transpose) to simplify the derivations later.

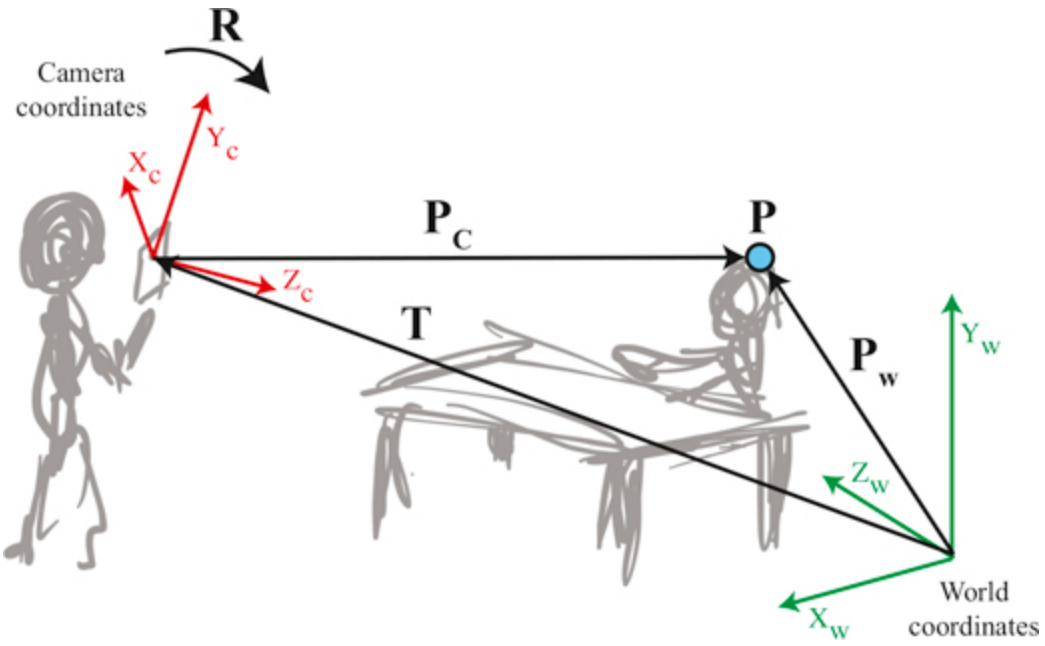


Figure 39.8: World- and camera-coordinate systems. A 3D point expressed in world coordinates, $\mathbf{P}_W$, can be expressed in the camera-coordinate frame, $\mathbf{P}_c$, by applying the translation, **T**, and rotation, **R**, to the point coordinates.

We are now given a point in the world $\mathbf{P}_W$, with its coordinates are described in terms of the world-coordinate system. Our goal is to find how that point projects into the image captured by the camera. To do this, first we need to change the coordinate system and express the point with respect to the camera coordinate system. Once we have done that change of coordinate system, we can use the intrinsic camera model *K* to project the point into the image plane.

We need to initially translate, and subsequently rotate, the camera so that points in the world, $\mathbf{P}_W$, are expressed in a coordinate system that originates at the camera center and is rotated to align with the camera. Note that this order is reversed from how we defined the camera coordinate system earlier in this section. Using heterogeneous coordinates we can describe these two transformations as:

$$\mathbf{P}_C = \mathbf{R}(\mathbf{P}_W - \mathbf{T}) = \mathbf{R}\mathbf{P}_W - \mathbf{R}\mathbf{T} \tag{39.14}$$

where we use the 3 × 3 matrix, **R**, to rotate the camera so that the camera's coordinates are parallel to those of the world. The vector **T** is the position of the camera in world coordinates. The rotation **R** in this equation is the inverse of the rotation matrix of the camera coordinate system with respect to the world coordinates.

Let's use now homogeneous coordinates to describe the transformation from world coordinates to camera coordinates. To do so, we first translate the coordinate system by –**T** using the homogeneous matrix, $\mathbf{M}_1$:

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & -T_X \\ 0 & 1 & 0 & -T_Y \\ 0 & 0 & 1 & -T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.15}$$

Then, we use the 3 × 3 matrix, **R**, to rotate the camera so that the camera's coordinates are parallel to those of the world:

$$\mathbf{M}_2 = \begin{bmatrix} R_1 & R_2 & R_3 & 0 \\ R_4 & R_5 & R_6 & 0 \\ R_7 & R_8 & R_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.16}$$

Using these two matrices we have

$$\mathbf{P}_C = \mathbf{M}_2\mathbf{M}_1\mathbf{P}_W \tag{39.17}$$

where $\mathbf{P}_W$ and $\mathbf{P}_C$ are the 3D coordinates of the same point (figure 39.8), written in world and camera coordinates, respectively. Sometimes, this transformation is written by making the product of the two matrices explicit:

$$\mathbf{M}_2\mathbf{M}_1 = \begin{bmatrix} R_1 & R_2 & R_3 & -\mathbf{r}_1\mathbf{T} \\ R_4 & R_5 & R_6 & -\mathbf{r}_2\mathbf{T} \\ R_7 & R_8 & R_9 & -\mathbf{r}_3\mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{RT} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \tag{39.18}$$

where $\mathbf{r}_i$ represents the row i of the rotation matrix **R**.

Substituting equation (39.18) into equation (39.17) we can now translate a 3D point expressed in world coordinates into camera coordinates:

$$\mathbf{P}_C = \begin{bmatrix} \mathbf{R} & -\mathbf{RT} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \mathbf{P}_W \tag{39.19}$$

The camera parameters, **R** and **T**, that relate the camera to the world are called the **extrinsic camera parameters**. These parameters are external to the camera.

We have seen now how to transform a point described in the world-coordinate system into the camera-coordinates system. We can now use the intrinsic camera parameters to project the point into the image plane. In the next section we will see how to combine both sets of parameters to get a full camera model.

39.5Full Camera Model

The concatenation of the extrinsic and intrinsic transformation matrices yields the desired transformation matrix, which will transform world coordinates to rendered pixel coordinates:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \mathbf{K}\mathbf{M}_2\mathbf{M}_1 \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{39.20}$$

Converting back to heterogeneous from homogeneous coordinates, the 2D pixel indices of a 3D point $[X, Y, Z]^\mathsf{T}$ are $[x, y]^\mathsf{T} = [x'/w, y'/w]^\mathsf{T}$. We can make all the matrices in equation (39.20) explicit and write:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & 0 & c_x & 0 \\ 0 & a & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_1 & R_2 & R_3 & 0 \\ R_4 & R_5 & R_6 & 0 \\ R_7 & R_8 & R_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -T_X \\ 0 & 1 & 0 & -T_Y \\ 0 & 0 & 1 & -T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{39.21}$$

The right-most matrix multiplications reflect the camera's extrinsic parameters, and the left-most matrix reflects the intrinsic parameters and the perspective projection. Note that column of zeros in the left-most matrix means that, without loss of generality, equation (39.21) can be written in slightly simpler form as:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & 0 & c_x \\ 0 & a & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_1 & R_2 & R_3 \\ R_4 & R_5 & R_6 \\ R_7 & R_8 & R_9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -T_X \\ 0 & 1 & 0 & -T_Y \\ 0 & 0 & 1 & -T_Z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{39.22}$$

This is usually written as:

$$\mathbf{p} = \mathbf{K}\left[\mathbf{R} \middle| -\mathbf{R}\mathbf{T}\right]\mathbf{P}_W \tag{39.23}$$

By replacing **K** with different intrinsic parameters we can build models for different camera types such as orthographic cameras, weak-perspective model, affine cameras, and so on. Figure 39.9 summarizes this section.

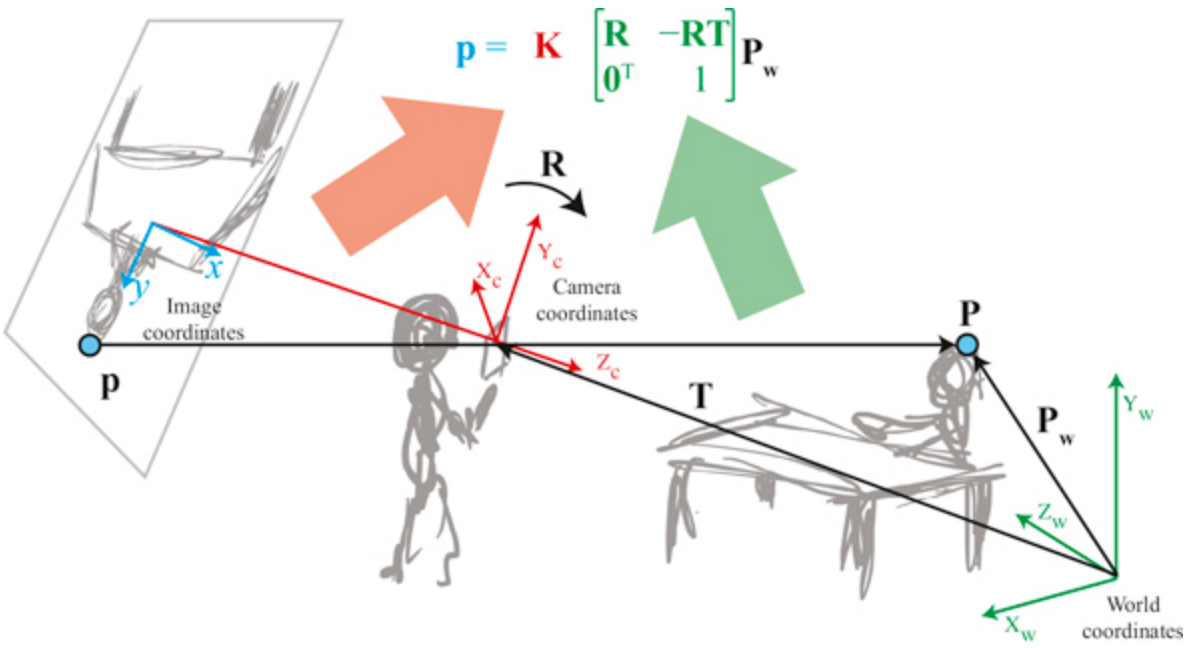


Figure 39.9: Projection of a point into the image plane. This summary puts together figure 39.3 and figure 39.8. First, we change the world-coordinates system, in which the point is expressed, into the camera-coordinates system, using the extrinsic camera model, and then we project it into the camera plane using the intrinsic camera model.

In the rest, we will usually refer to the camera projection matrix, which includes both intrinsic as extrinsic parameters as:

$$\mathbf{M} = \mathbf{K}\begin{bmatrix}\mathbf{R} & -\mathbf{RT}\\ \mathbf{0}^T & 1\end{bmatrix} \tag{39.24}$$

39.6A Few Concrete Examples

Let's write down the full camera model for a few concrete scenarios that are also of practical relevance. The four scenarios we will consider have growing complexity as shown in figure 39.10.

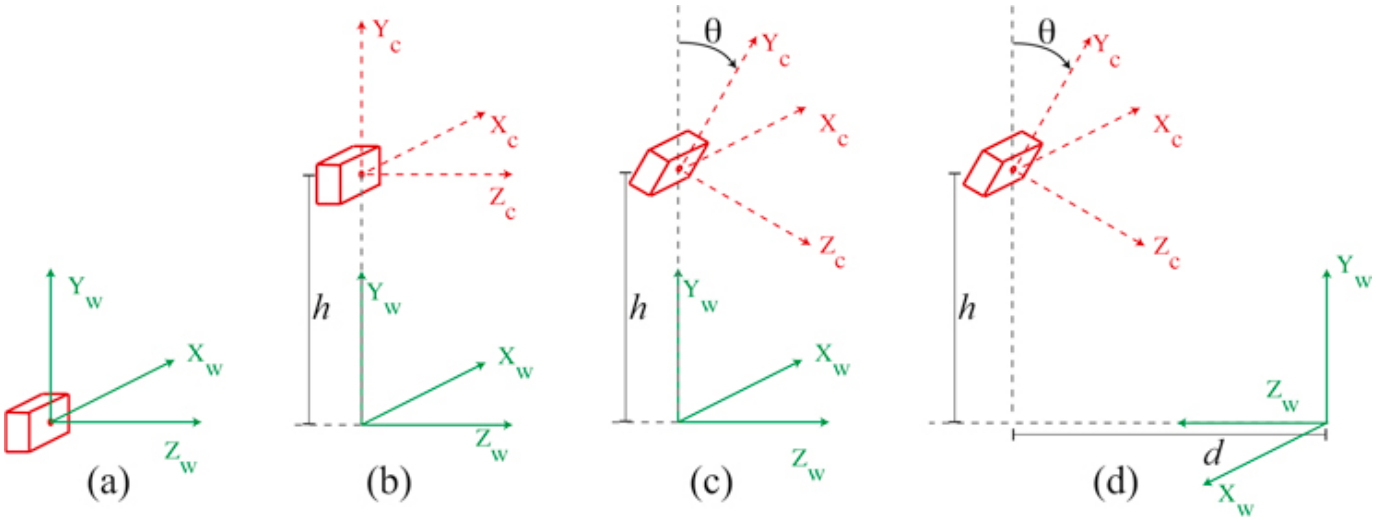


Figure 39.10: Four examples of camera poses respect to the world-coordinates system with increasing complexity. In the text we derive the projection matrix for each scenario.

Example 1. We will start with a camera located at the origin of the world-coordinate systems, as shown in figure 39.10(a). In this case, both coordinate systems are identical and the camera projection matrix is:

$$\mathbf{M} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.25}$$

We have set $c_x = c_y = 0$, which assumes that the center of the image are the coordinates $(x, y) = (0, 0)$. This will simplify the equations for the following examples.

Example 2. The second scenario we will consider is shown in figure 39.10(b). This scenario corresponds to a case where a person is holding a camera, pointing toward the horizon in such a way that the optical axis of the camera is parallel to the ground. In this case we place the world-coordinate frame with its origin on the ground plane right below the camera. The camera is at a height h from the ground, measured in meters.

$$\mathbf{M} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & -ah \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{39.26}$$

Going back to heterogeneous coordinates we have that a 3D point at location $P_W = [X, Y, Z]^\mathsf{T}$ in world coordinates projects to the image coordinates:

$$x = a\frac{X}{Z} \tag{39.27}$$

$$y = a\frac{Y-h}{Z} \tag{39.28}$$

In this scenario, if you hold the camera parallel to the ground at the height of your eyes and take a picture of a person standing in front of you with a similar height, their eyes will project near the middle portion of the vertical axis of the picture (their eyes will be located at $Y \; h$; therefore, they will project to $y \; 0$, which is the center of the picture). And this will be true regardless of how far away they are. This is illustrated in figure 39.11. The horizon line is located at the points where $Z \rightarrow \infty$, which are also at $y = 0$ (we will talk more about the horizon line in the following chapter).

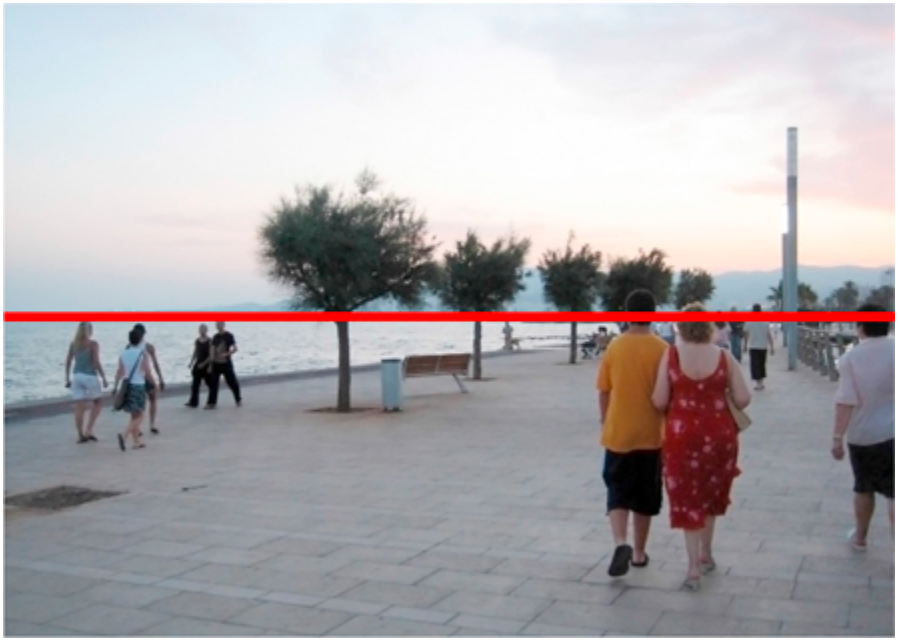


Figure 39.11: If you hold the camera parallel to the ground at the height of your eyes and take a picture of a person standing in front of you with a similar height, their eyes will project near the middle portion of the vertical axis of the picture.

Example 3. Let's now consider that the camera is tilted by looking downward with an angle θ, as shown in figure 39.10(c). The horizontal camera axis remains parallel to the ground, but now the optical axis points toward the ground if $\theta > 0$.

Let's first write down the camera-extrinsic parameters. The angle θ here corresponds to a rotation around the X_c-axis, thus we can write:

$$\begin{bmatrix} \mathbf{R} & -\mathbf{RT} \\ \mathbf{0}^\top & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.29}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & -h\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.30}$$

Putting both intrinsic and extrinsic camera parameters together we get the following projection matrix:

$$\mathrm{M} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a\cos(\theta) & a\sin(\theta) & -ah\cos(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) & h\sin(\theta) \end{bmatrix} \tag{39.31}$$

Going back to heterogeneous coordinates we have:

$$x = a\frac{X}{\sin(\theta)(h-Y)+\cos(\theta)Z} \tag{39.32}$$

$$y = a\frac{\cos(\theta)(Y-h)+\sin(\theta)Z}{\sin(\theta)(h-Y)+\cos(\theta)Z} \tag{39.33}$$

Let's look at three special cases according to the camera angle θ:

- For $\theta = 0$, we recover the projection equations from the previous example. The horizon line is an horizontal line that passes by the center of the image.
- For $\theta = 90$ degrees (this is when the camera is looking downward), we get $x = aX/(h - Y)$ and $y = aZ/(h - Y)$. In this case, the equations are analogous to the standard projection equation with the distance to the camera being $h - Y$ and Z playing the role of Y.
- For arbitrary angles θ, the horizon line corresponds to the y location for a point in infinity, $Z \rightarrow \infty$. Using equation (39.33), the horizon line is located at $y = a\tan(\theta)$.

The sketch in figure 39.12 shows a person taking a picture of two people standing at different distances from the camera. If you take a picture of two people with a similar height to you, h, standing in front of the camera. Then, we can set $Y = h$ in the previous equations, their eyes will be located at the position $y = a\tan(\theta)$ in the picture, which is independent of Z and the same vertical location as the horizon line.

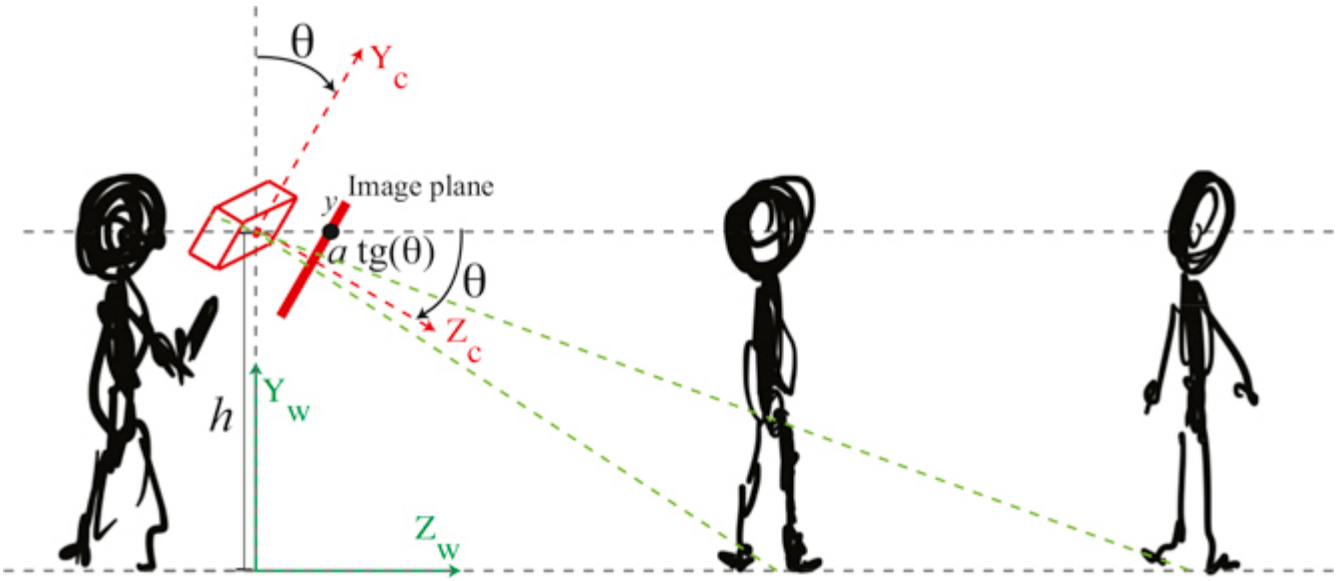


Figure 39.12: Sketch showing a person taking a picture of two people standing at different distances from the camera. Their eyes will project to the same image row regardless of their distance to the camera.

Note that if a is known (i.e., if the camera is calibrated), then we can infer θ by locating the horizon line in the image. Analogously, if the angle of the camera, θ, is known, we could estimate a.

The two pictures in figure 39.13 are taken with two different camera angles. In both pictures, the camera's x-axis is kept parallel to the ground. The right and left pictures were taken at the same location with the camera tilted upward and downward, respectively. The horizontal red lines show the approximate location of the horizon line in each image and coincides also with the vertical location where the heads of the people walking appear on the picture.

Figure 39.13: The two pictures taken with two different camera angles. The red line indicates the position of the horizon line estimated as the horizontal location in which the two vanishing lines (blue) intersect.

These pictures were taken with the iPhone 13 pro, which we calibrated before. The intrinsic camera parameter we got is a 3,103, which we can use to compute the angle θ. On the right, the location of the horizon line is at $y_h = -798$, with the center of the picture being the coordinates (0, 0). The angle is $\theta = \arctan(y_h/a) = 14.6$ degrees. For the picture on the right the horizon line is at $y_h =$ 1,129, resulting in an estimated camera angle of $\theta = 20$ degrees. Although we did not precisely measured the angle of the camera when we were taking these two pictures, both angles seem about right. We leave it to the reader to take two similar pictures to the ones in the example above, and get a precise measurement of the camera angle using a protractor and check if the estimated angle from the horizon line matches your measurement.

> Example 3 can be useful to model typical imaging conditions. It results in a simple model with three parameters (h, a, and θ).

Example 4. The setting shown in shown in figure 39.10(d) is like the one we used in the simple visual system (chapter 2). Let's see if we recover the same equations!

Let's start computing the rotation matrix. Now we have rotation along two axes. We have a rotation around the X_w-axis with an angle $\theta_X = \theta$ and a rotation along the Y_w-axis with a $\theta_Y = 180$ degrees rotation.

$$\begin{aligned}\mathbf{R} &= \begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\theta_X) & \sin(\theta_X)\\ 0 & -\sin(\theta_X) & \cos(\theta_X)\end{bmatrix}\begin{bmatrix}\cos(\theta_Y) & 0 & \sin(\theta_Y)\\ 0 & 1 & 0\\ -\sin(\theta_Y) & 0 & \cos(\theta_Y)\end{bmatrix}\\ &= \begin{bmatrix}-1 & 0 & 0\\ 0 & \cos(\theta_X) & -\sin(\theta_X)\\ 0 & -\sin(\theta_X) & -\cos(\theta_X)\end{bmatrix}\end{aligned} \tag{39.34}$$

It is important to note that this form of dealing with rotations is more complex than it seems. Once we apply a rotation along one axis, the following rotation will be affected. That is, the order in which these rotations are executed matters. In this example, it works if we rotate along the Y-axis first. In general, it is better to use other conventions to write the rotation matrix.

The projection matrix is obtained by using both intrinsic and extrinsic camera parameters together, resulting in:

$$\begin{aligned}\mathbf{M} &= \begin{bmatrix}a & 0 & 0 & 0\\ 0 & a & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}-1 & 0 & 0 & 0\\ 0 & \cos(\theta_X) & -\sin(\theta_X) & 0\\ 0 & -\sin(\theta_X) & -\cos(\theta_X) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & -h\\ 0 & 0 & 1 & -d\\ 0 & 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}-a & 0 & 0 & 0\\ 0 & a\cos(\theta) & -a\sin(\theta) & -ah\cos(\theta)+ad\sin(\theta)\\ 0 & -\sin(\theta) & -\cos(\theta) & h\sin(\theta)+d\cos(\theta)\end{bmatrix}\end{aligned} \tag{39.35}$$

And going back to heterogeneous coordinates we have:

$$x = -a\frac{X}{\sin(\theta)(h-Y)+\cos(\theta)(d-Z)} \tag{39.36}$$

$$y = a\frac{\cos(\theta)(Y-h)+\sin(\theta)(d-Z)}{\sin(\theta)(h-Y)+\cos(\theta)(d-Z)} \tag{39.37}$$

This setting is similar to the one we used when studying the simple visual system from chapter 2. However, you will notice that the equations obtained are different. The reason is that in the simple visual system projection model we made two additional simplifying assumptions. The first simplifying assumption was that the use of parallel projection, while the previous equations use perspective projection. To get the right set of equations we should use the **K** matrix that corresponds to the parallel projection camera model, as shown in equation (39.4), which results in:

$$\mathbf{M} = \begin{bmatrix} -a & 0 & 0 & 0 \\ 0 & a\cos(\theta) & -a\sin(\theta) & -ah\cos(\theta)+ad\sin(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.38}$$

The second constraint we had in the simple world chapter is that the origin of the world coordinates was the point where the camera optical axis intersected the ground plane, this puts a particular restriction between the values of d, h, and θ. Concretely, the constraint is, $\tan(\theta) = h/d$. Therefore, $d\sin(\theta) - h\cos(\theta) = 0$. In heterogeneous coordinates this results in:

$$x = -aX \tag{39.39}$$

$$y = a\cos(\theta)Y - a\sin(\theta)Z \tag{39.40}$$

These equations are similar to equation (2.2) from chapter 2 (with $a = 1$). One remaining difference is that x has the opposite sign. This is because we are using here a different convention for the sign of the x-axis of the image coordinates. This is fine, and in fact, books and conference publications use a wide set of conventions, and you should be ready to adapt the formulation to each setting.

As you can see, working with heterogeneous coordinates can be tedious. However, in most cases we will work directly with the camera matrix **K** and/or the projection matrix **M** without needing to derive their equations. These matrices will be obtained during the camera calibration process.

39.7Camera Calibration

A camera is said to be **calibrated** if we know the transformations relating 3D world coordinates to pixel coordinates within the camera. Those transformations involve two types of parameters: (1) parameters that depend on where the camera is physically located in the world, and (2) parameters that are a function of the camera itself. These two types of parameters are called extrinsic and intrinsic parameters, respectively.

39.7.1Direct Linear Transform

Let's assume that we know the exact locations of several 3D points in our scene in world coordinates and we also know the locations in which those 3D points project into the image. Can we use them to recover the intrinsic and extrinsic camera parameters?

If we have six correspondences, then, under certain conditions, we can recover the projection matrix **M** by solving a linear system of equations.

For each pair of corresponding pixels, $\mathbf{p}_i$, and 3D points, $\mathbf{P}_i$, we have the following relationship:

$$\mathbf{p}_i = \mathbf{M}\mathbf{P}_i = \begin{bmatrix} \mathbf{m}_0^\top \\ \mathbf{m}_1^\top \\ \mathbf{m}_2^\top \end{bmatrix} \mathbf{P}_i \tag{39.41}$$

where **M** is a 3 × 4 projection matrix. To simplify the derivation that comes next, it is convenient to write the matrix **M** in terms of its three rows using the vectors $\mathbf{m}_0^T$, $\mathbf{m}_1^\top$, and $\mathbf{m}_2^\top$. With this notation, the heterogeneous coordinates of **p** are:

$$x_i = \frac{\mathbf{m}_0^\top \mathbf{P}_i}{\mathbf{m}_2^\top \mathbf{P}_i} \tag{39.42}$$

$$y_i = \frac{\mathbf{m}_1^\top \mathbf{P}_i}{\mathbf{m}_2^\top \mathbf{P}_i} \tag{39.43}$$

By rearranging the terms, we get the following two linear equations on the parameters of the matrix **M**:

$$\mathbf{P}_i^\top \mathbf{m}_0 - x_i \mathbf{P}_i^\top \mathbf{m}_2 = 0 \tag{39.44}$$

$$\mathbf{P}_i^\top \mathbf{m}_1 - y_i \mathbf{P}_i^\top \mathbf{m}_2 = 0 \tag{39.45}$$

The same two equations written in matrix form are:

$$\begin{bmatrix} -\mathbf{P}_i^\top & \mathbf{0} & x_i \mathbf{P}_i^\top \\ \mathbf{0} & -\mathbf{P}_i^\top & y_i \mathbf{P}_i^\top \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \mathbf{m}_2 \end{bmatrix} = \mathbf{0} \tag{39.46}$$

Now, if we stack together all the equations derived from *N* correspondences we get the following homogeneous linear system of equations:

$$\begin{bmatrix} -\mathbf{P}_1^\top & \mathbf{0} & x_1\mathbf{P}_1^\top \\ \mathbf{0} & -\mathbf{P}_1^\top & y_1\mathbf{P}_1^\top \\ \vdots & \vdots & \vdots \\ -\mathbf{P}_N^\top & \mathbf{0} & x_N\mathbf{P}_N^\top \\ \mathbf{0} & -\mathbf{P}_N^\top & y_N\mathbf{P}_N^\top \end{bmatrix} \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \mathbf{m}_2 \end{bmatrix} = \mathbf{0} \tag{39.47}$$

This can be summarized as:

$$\mathbf{Am} = \mathbf{0} \tag{39.48}$$

where **A** is a $2N \times 12$ matrix. The vector **m** contains all the elements of the matrix **M** stacked as a column vector of length 12. Note that **m** only has 11 degrees of freedom as the results do not change for a global scaling of all the values.

> Rearranging the vector m into the matrix M is a potential source of bugs. Try first simulating some 3D points and project them with a known M matrix, and check that you recover the same values (up to a scale factor).

As the measurements of the points in correspondence will be noisy, it is generally useful to use more than six correspondences. Therefore, the solution will not be 0. The solution is the vector that minimizes the residual $\mathbf{Am}^2$, which can be obtained as the smallest eigenvector of the matrix $\mathbf{A}^\top\mathbf{A}$. Once we estimate **m** we can rearrange the terms into the projection matrix **M**. This method to obtain the projection matrix is called the **direct linear transform** (DLT) algorithm. This method requires that not all of the N points lie in a single plane [187].

The term $\mathbf{Am}^2$ doesn't have any particular physical meaning, but minimizing it provides a good first guess that can be improved by other methods as we will discuss in section 39.7.4.

39.7.2Recovering Intrinsic and Extrinsic Camera Parameters

Once **M** is estimated, we would like to recover the intrinsic camera parameters, and also camera translation and rotation. Can it be done? It turns out that the answer is yes!

Let's say that you start with a 3×4 matrix **M** obtained by calibrating the camera using DLT (or any other calibration method), and you want to extract the matrices **K**, **R**, and **T**. It is possible to decompose the matrix **M** such that it can be written as:

$$\mathbf{M} = \left[\mathbf{KR} \mid -\mathbf{KRT}\right] \tag{39.49}$$

It turns out that, due to the characteristics of those three matrices, one can find such decomposition. The first step is to recover **T**. This can be done by seeing that the matrix **M** can be written as the concatenation of a 3×3 matrix, **B**, and 1×3 column vector **b**; that is, **M** = [**B b**]. The first matrix is **B** = **KR**, and the vector is **b** = –**KRT**. Therefore, we can estimate the camera translation as:

$$\mathbf{T} = -\mathbf{B}^{-1}\mathbf{b} \tag{39.50}$$

The next step is to decompose **B** into the product of two matrices **KR**. These two matrices are very special. The matrix **K** is an upper triangular matrix by construction (in the case of perspective projection), and **R** is an orthonormal matrix. To estimate **K** and **R** we can use the QR decomposition of the matrix $\mathbf{B}^{-1}$. The QR decomposition decomposes a matrix in the product of a unitary matrix and an upper triangular one, which is the reverse order of what we want. This is why we need to use the inverse of **B**.

The decomposition obtained with the QR decomposition is not unique. In fact, we can change the sign of both matrices and still get the same decomposition. Also, changing the sign of column *i* in **K** and the sign of the column *i* in **R** also results in the same product. As we know that the diagonal elements of the **K** have to be positive we can change the sign of the columns of **K** with a negative entry in the diagonal element and also change the sign of the corresponding row on the rotation matrix **R**.

This method to recover the camera parameters might be quite sensitive to noise, but it can be used to initialize other methods based on minimizing the **reprojection error**.

39.7.3Multiplane Calibration Method

A popular method for camera calibration is to use a planar calibration chart and to take multiple pictures of it by placing the chart in different locations and orientations relative to a fixed camera. This method was introduced by Zhang's in 1999 [526] and it is still commonly used.

39.7.4Nonlinear Optimization by Minimizing Reprojection Error

The reprojection error is illustrated in figure 39.14. The reprojection error is the difference between the estimated image coordinates of a 3D point and its ground-truth image coordinates.

We are given 3D points, $\mathbf{P}_i$, and their corresponding 2D projections, $\mathbf{p}_i$. We start with some estimated camera parameters **K**, **R**, and **T**. We can use these parameters to project the 3D points and get estimated 2D locations, $\mathbf{P}'_i$, of their projections. The reprojection error is the distance between those points, as shown in figure 39.14.

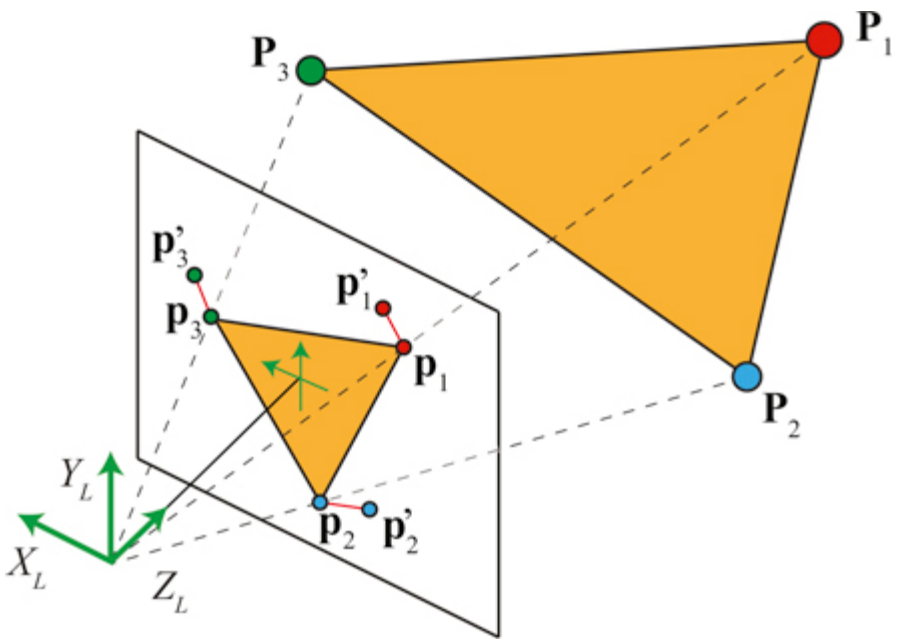


Figure 39.14: Reprojection error indicated by the red lines on the image plane.

The reprojection error can be evaluated with the loss:

$$\sum_{i=1}^{N} \|\mathbf{p}_i - \mathbf{p}'_i\|^2 = \sum_{i=1}^{N} \|\mathbf{p}_i - \pi\left(\mathbf{K}\left[\mathbf{R}\middle|-\mathbf{RT}\right]\mathbf{P}_i\right)\|^2 \tag{39.51}$$

where $\pi()$ is the function that transforms homogeneous coordinates to heterogeneous coordinates. This function can also incorporate other camera parameters to account for geometric distortion introduced by the camera optics.

The reprojection loss (equation [39.51]) can be minimized by gradient descent, optimizing the three matrices, **K**, **R**, and **T**, directly. For the minimization to converge it is usually initialized with the solution obtained using DLT.

39.7.5A Toy Example

The last few pages were full of math, and although everything seems sound, it is good to remain suspicious about what subtleties might be hidden behind the math that might make everything break. Can we run a simple test to see if the theory meets the practice?

We wanted to try it out so here is what we did. We started by taking a picture of one office and measure some distances as shown in figure 39.15. The goal is to get real 3D coordinates measured in the world and use them to calibrate the camera.

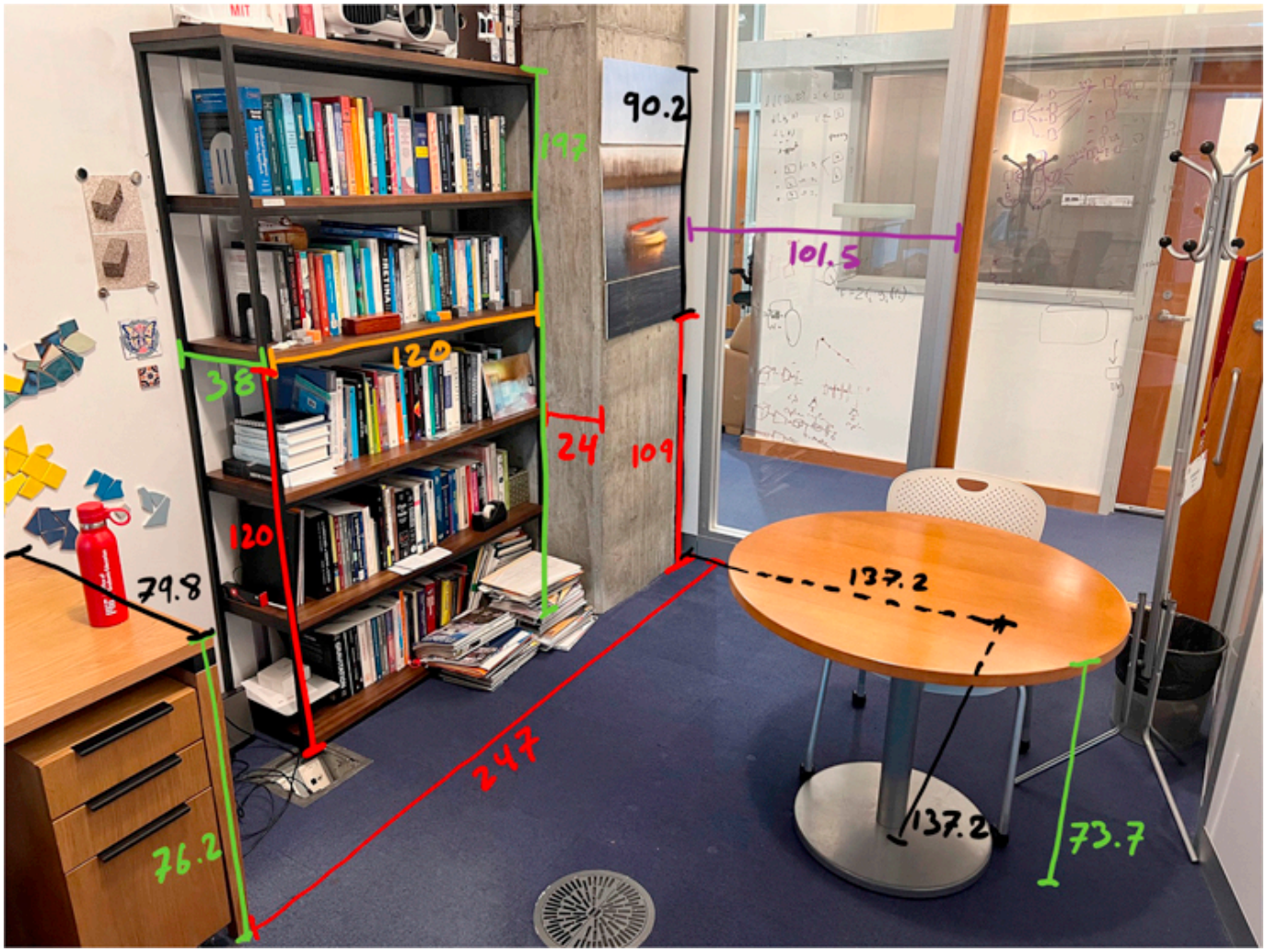
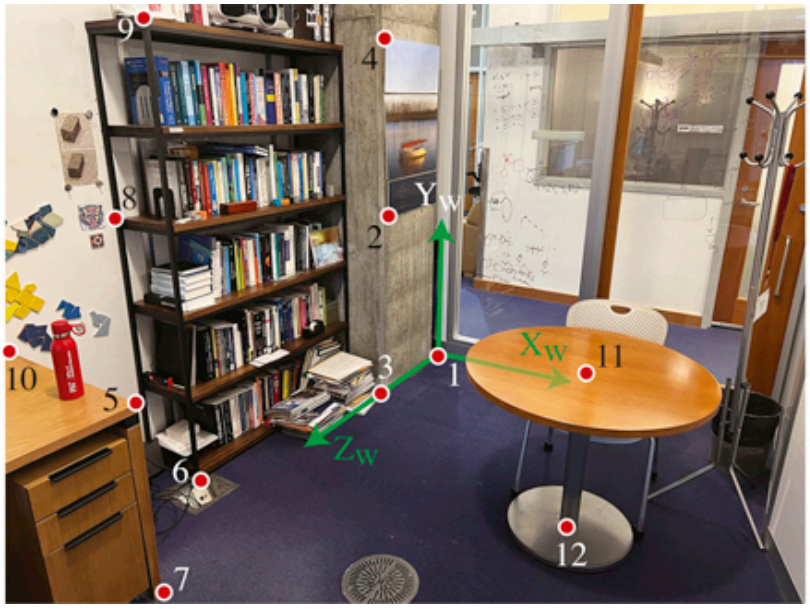


Figure 39.15: Office picture and real distances between a sparse set of points measured in centimeters.

We took a picture, figure 39.15, with the same iPhone we used in section 39.3.3, holding the camera at eye height (which is around 67 in, or 170 cm above the ground). The camera was looking downward at an angle of around 15 degrees with respect to the vertical. Would we be able to recover these extrinsic camera parameters, together with the intrinsic camera parameters, using the theory described in this chapter?

First we need to decide where we will place the origin of the world coordinates and the axis direction. As shown in the following picture, we place the origin on the ground floor, in the corner between the column and the glass wall in the back. Now we can get a list of 3D points from the office measurements and their corresponding 2D coordinates. The following figure shows a list of easily identifiable 3D points and their corresponding image coordinates. The image coordinates are approximated also and are collected manually. We use Adobe Photoshop to read the pixel coordinates.



i	X_i	Y_i	Z_i	x_i	y_i
1	0	0	0	2182	1261
2	0	109	73	1931	1952
3	0	0	73.7	1902	1064
4	0	199.2	73	1917	2855
5	17.8	76.2	247	670	1005
6	-24	0	193.7	979	612
7	17.8	0	247	793	50
8	-62	120	193.7	550	1942
9	-24	197	193.7	692	2963
10	-62	76.2	247	30	1268
11	137.2	73.7	137.2	2938	1137
12	137.2	3.8	137.2	2836	386

Figure 39.16: Office picture and a table with the 3D world coordinates of 12 points, extracted using the measurements from figure 39.15, and their corresponding 2D image coordinates measured in pixels.

Photoshop has the image coordinates origin on the top left. Our convention has been to put the origin on the bottom and the x-axis point from right to left. Therefore, we need to change the coordinates first (or just deal with it later). It is just simple to replace the image coordinates by $y_i \leftarrow 3{,}024 - y_i$ and $x_i \leftarrow 4{,}032 - x_i$.

In order to estimate the projection matrix we need a minimum of eight correspondences. Here we have 12, which will help to reduce the noise (and we assure you there is a lot of noise on those manual measurements). Applying the DLT algorithm we recover the following projection matrix:

$$\mathrm{M} = \begin{bmatrix} -8.1 & -1.8 & -0.2 & 1{,}845.5 \\ -3 & 5.4 & -4.8 & 1{,}262.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{39.52}$$

The matrix has been scaled so that the bottom-right element is a 1. Using this matrix, the mean reprojection error is 12.3 pixels, which is reasonably small for an image of size 4,032 × 3,024 pixels. That is, the reprojection error is < 0.5 percent of the image height.

We now need to recover the intrinsic and extrinsic camera parameters. Using equation (39.50) we recover first the translation vector which gives us **T** = (182.3; 171.8; 347.6). From our definition of the world-coordinates frames, the middle element of **T** is the camera height with respect to the ground plane. The value of 171.8 cm is very close to our initial guess! Given a scene with known measurements and a picture, one can find out details about the camera and also about the observer.

A picture reveals information about the camera and the photographer, even when there is no one on the picture. It is important to keep this in mind when thinking about privacy.

Let's now see if we can recover the camera angle and the intrinsics. Using the QR decomposition as described in section 39.7.2, we get

$$\mathbf{K} = \begin{bmatrix} 2{,}960 & -24.9 & 1{,}979.7 \\ 0 & 3{,}019 & 1{,}433.6 \\ 0 & 0 & 1 \end{bmatrix} \tag{39.53}$$

The values are very close to the ones from equation (39.12) which were obtained using a calibration pattern and also from the camera technical specifications. The off-diagonal elements, which would account for a skew pixel shape in the camera, are small relative to the focal length and probably just due to the annotation noise. Also, both scaling factors are very similar, and their difference might just be due to noise. The camera is likely to have square pixels. The last column is the location of the optical image center, as we defined the image coordinates with the origin in the bottom right instead of that in the image center, and is quite close to what one would expect: [W/2, H/2, 1].

Now let's look at the camera orientation. The rotation matrix obtained with the QR decomposition is

$$\mathbf{R} = \begin{bmatrix} -0.8576 & -0.0162 & 0.5141 \\ 0.1928 & -0.9368 & 0.2921 \\ -0.4769 & -0.3496 & -0.8064 \end{bmatrix} \tag{39.54}$$

The viewing direction of the camera can be obtained as $\mathbf{R}^\mathsf{T}[0, 0, 1]^\mathsf{T}$ (this is the direction of the Z-axis in the camera-coordinates system).

Another potential bug when extracting the camera orientation is to use the rotation matrix instead of its inverse. We defined $\mathbf{R}^\mathsf{T}$ as the rotation needed to move the world coordinate axes to be aligned with the camera-coordinate axes.

The angle of the camera with respect to the vertically can be computed in multiple ways. Here is one:

$$\begin{aligned} \arccos\left([0,1,0]\mathbf{R}^\mathsf{T}[0,1,0]^\mathsf{T}\right) &= \arccos(R[2,2]) \\ &= \arccos(-0.9368) \\ &= 20.5 \text{ degrees} \end{aligned} \tag{39.55}$$

Here we just computed the camera Y-axis and then computed the dot product with a unit vector along the Y-axis of the world-coordinates. This angle is also quite close to the approximate value that we gave at the beginning of the section.

It is always important to visualize everything to make sure that all is correct. Figure 39.17 shows a visualization of the annotated 3D points and the inferred camera location and orientation. The figure shows three different viewpoints.

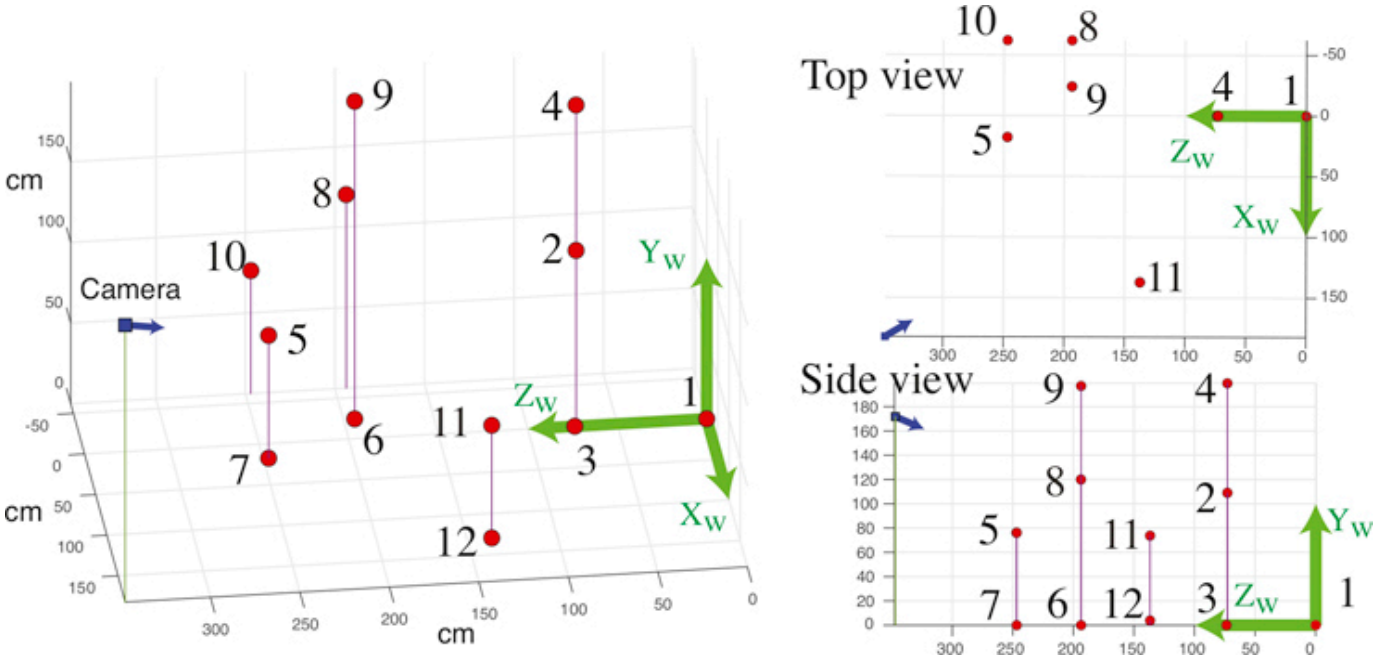


Figure 39.17: Inferred camera location for the office picture. The figure shows three different viewpoints.

39.8 Concluding Remarks

In this chapter we have developed a model for describing how points in the 3D world project into the image. For a more in depth study of the topics described in this chapter, the reader should consult specialized books such as [477] and [187].

There are a number subtleties involved in computing the quantities we have presented in this chapter. Although there are packages that will calibrate a camera and compute the projection matrix for you, it is useful to be familiar with how these methods work as they play an important role when building unsupervised learning methods. Unsupervised learning often demands familiarity with the constraints of the vision problem, which become useful when trying to design the loss functions that do not require human annotated data.

But the goal of vision is to interpret the scene, that is, to recover the 3D scene from images. We will discuss in the next chapters how to do this.

40Stereo Vision

40.1Introduction

Figure 40.1(a) shows a **stereo anaglyph** of the ill-fated ocean liner, the Titanic, from [329]. This was taken with two cameras, one displaced laterally from the other. The right camera's image appears in red in this image, and the left camera image is in cyan. If you view this image with a red filter covering the left eye, and a cyan filter covering the right (figure 40.1[b]) then the right eye will see only the right camera's image (analogously for the left eye), allowing the ship to pop-out into a three-dimensional (3D) stereo image. Note that the relative displacement between each camera's image of the smokestacks changes as a function of their depth, which allows us to form a 3D interpretation of the ship when viewing the image with the glasses (figure 40.1[b]).

Figure 40.1:(a) Stereo anaglyph of the Titanic [329]. The red image shows the right eye's view, and cyan the left eye's view. (b) When viewed through red/cyan stereo glasses the cyan variations appear in the left eye and the red variations appear to the right eye, creating a perception of 3D.

In this chapter, we study how to compute depth from a pair of images from spatially offset cameras, such as those of figure 40.1. There are two parts to the depth computation, usually handled separately: (1) analyzing the geometry of the camera projections, which allows for triangulating depth once image offsets are known; and (2) calculating the offset between matching parts of the objects depicted in each image. Different techniques are used in each part. We first address the geometry, asking where points matching those in a first image can lie in the second image. A manipulation, called **image rectification**, is often used to place the locus of potential matches along a pixel row. Then, assuming a rectified image pair, we will address the second part of the depth computation, computing the offsets between matching image points.

40.2Stereo Cues

How can we estimate depth from two pictures taken by two cameras placed at different locations? Let's first gain some intuition about the cues that are useful to estimate distances from two views captured at two separate observation points.

40.2.1How Far Away Is a Boat?

To give an intuition on how depth from stereo vision works, let's first look at a simple geometry problem of practical use: How do we estimate how far away a boat is from the coast?

> Sailors can use tricks to estimate distances using methods related to the ones presented in this chapter. These techniques can be useful when the electronics on the boat are out.

We can solve this problem in several ways. We will briefly discuss two methods. The first one uses a single observation point but will use the horizon line as a reference. The second method will use two observation points. Both methods are illustrated in figure 40.2.

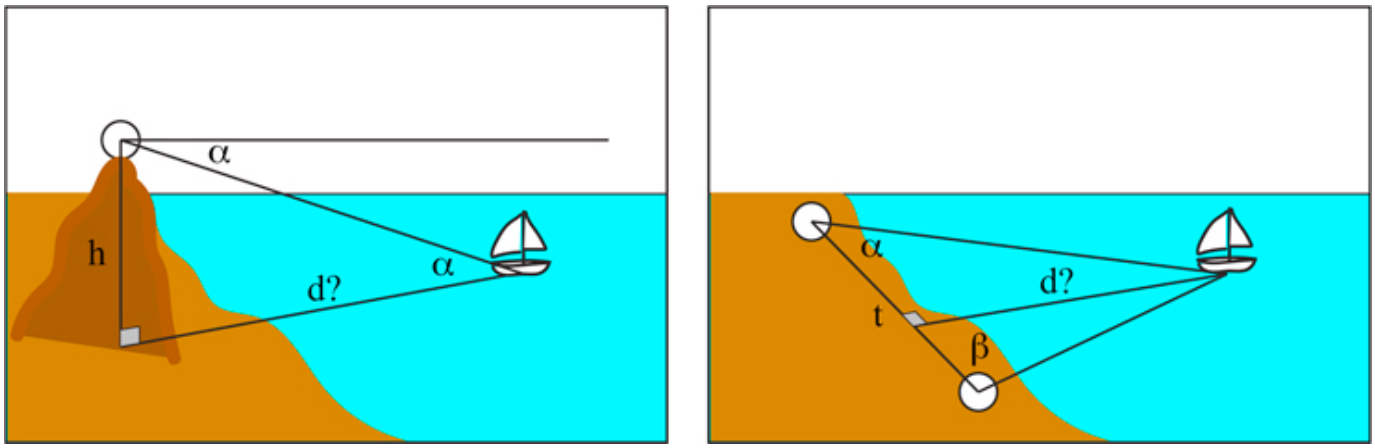


Figure 40.2: Two methods to estimate the distance of a boat from the coast. (left) The first method uses a single observation point, with knowledge of the observer's height above the water. (right) The second method uses two observation points.

The first method measures the location of the boat relative to the horizon line. An observer is standing on top of an elevated point (a mountain, tower, or building) at a known height, h. The observer measures the angle α between the horizontal (obtained by pointing toward the horizon) and the position of the boat. We can the derive the boat distance to the base of the point of observation, d by the simple trigonometric relation:

$$d = \frac{h}{\tan(\alpha)} \tag{40.1}$$

This method is not very accurate and might require high observation points, which might not be available, but it allows as to estimate the boat position with a single observation point, if the observer's height above the water is known. It also requires incorporating the earth curvature when the boat is far away.

The second method requires two different observation points along the coast separated by a distance t. At each point, we measure the angles, α and β, between the direction of the boat and the line connecting the two observation points. Then we can apply the following relation:

$$d = t\frac{\sin(\alpha)\sin(\beta)}{\sin(\alpha+\beta)} \tag{40.2}$$

This method is called **triangulation**. For it to work we need a large distance t so that the angles are significantly different from 90 degrees.

40.2.2Depth from Image Disparities

When observing the world with two eyes we get two different views of the world (figure 40.1). The relative displacement of features across the two images (**parallax effect**) is related to the distance between the observer and the observed scene. Just as in the example of estimating the distance to boats in the sea, our brain uses disparities across the two eye views to measures distances to objects.

> **Free fusion** consists in making the eyes converge or diverge in order to fuse a stereogram without needing a stereoscope. Making it work requires practice. Try it in figure 40.3

Our brain will use many different cues such as the ones presented in the previous chapter. But image disparity provides an independent additional cue. This is illustrated by the **random dot stereograms** (figure 40.3) introduced by Bela Julesz [241]. Random dot stereograms are a beautiful demonstration that humans can perceive depth using the relative displacement of features across two views even in the absence of any other depth cues. The random images contain no recognizable features and are quite different from the statistics of the natural world. But this seems not to affect our ability to compute the displacements between the two images and perceive 3D structure.

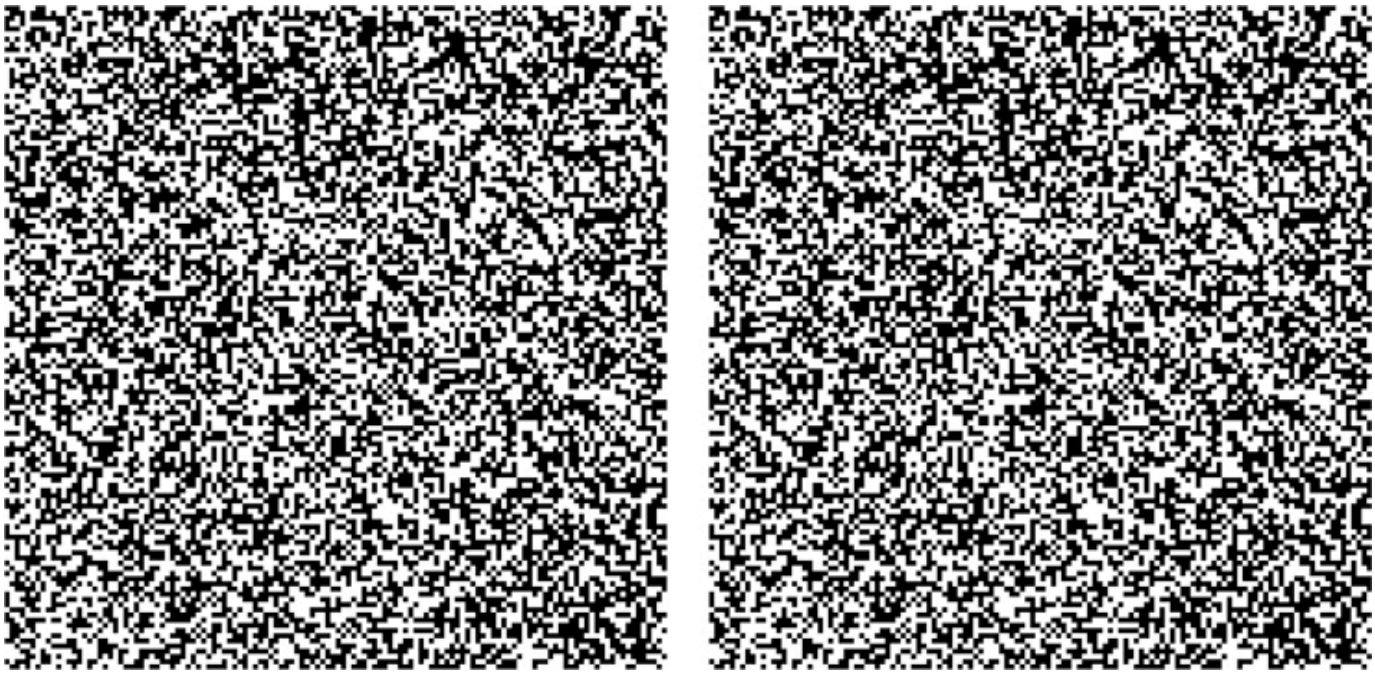


Figure 40.3: Random dot stereogram. Both images are almost identical. The difference is that there is a square in the middle that is displaced by a few pixels left-right between the two images. When each image it seen by one eye you should see a square floating in from of a flat background. You can use a stereoscope or free fusion to see it.

40.2.3Building a Stereo Pinhole Camera

Stereo cameras generally are built by having two cameras mounted on a rigid rig so that the camera parameters do not change over time.

We can also build a **stereo pinhole camera**. As we discussed in chapter 5, a pinhole camera is formed by a dark chamber with a single hole in one extreme that lets light in. The light gets projected into the opposite wall, forming an image. One interesting aspect of pinhole cameras is that you can build different types of cameras by playing with the shape of the pinhole. So, let's make a new kind of pinhole camera! In particular, we can build a camera that produces anaglyph images, like the one in figure 40.1, in a single shot by making two pinholes instead of just one.

We will transform the pinhole camera into an anaglyph pinhole camera by making two holes and placing different color filters on each hole (figure 40.4). This will produce an image that will be the superposition of two images taken from slightly different viewpoints. Each image will have a different color. Then, to see the 3D image we can look at the picture by placing the same color filters in front of our eyes. In order to do this, we used color glasses with blue and red filters.

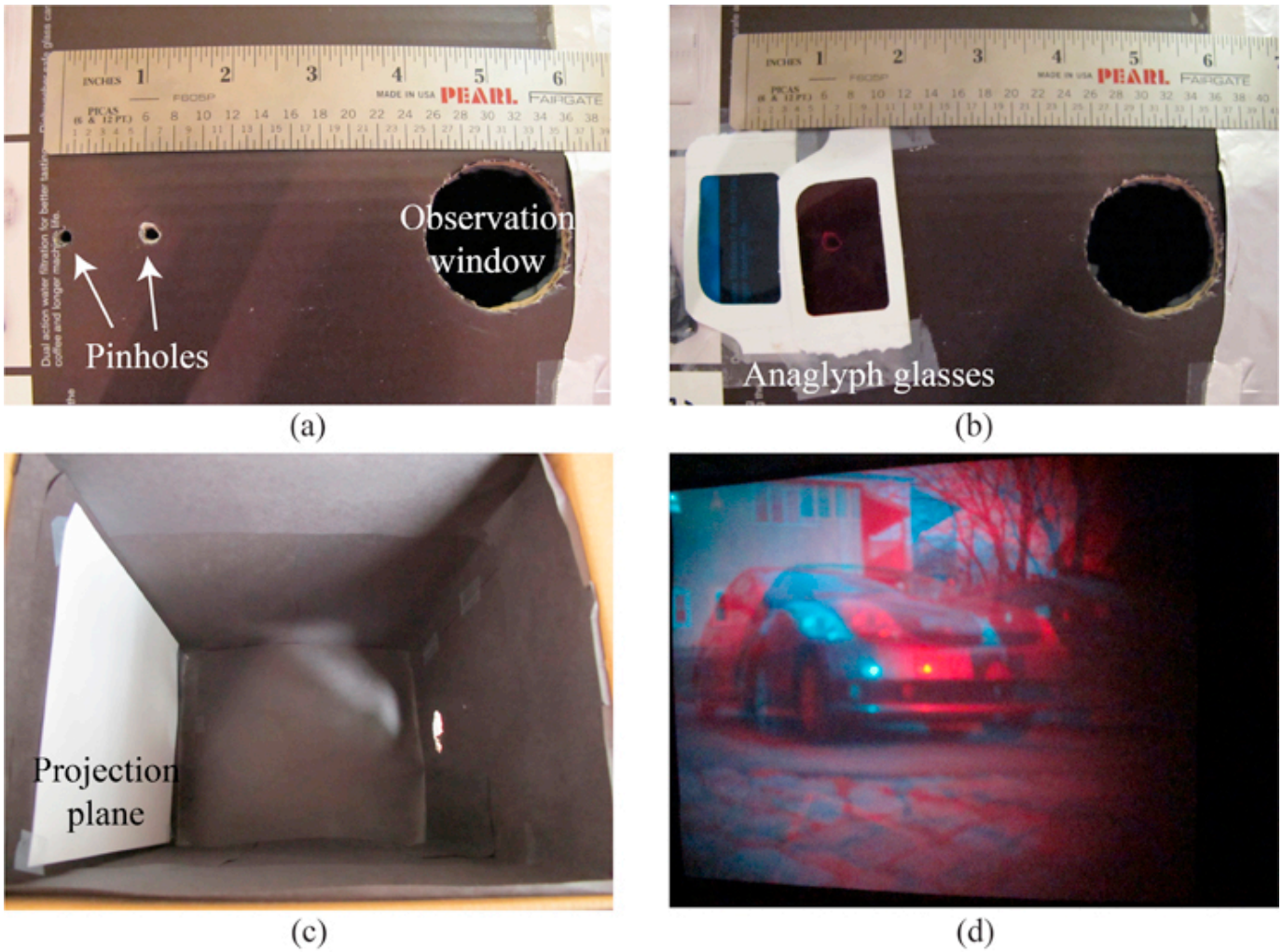


Figure 40.4: One way of playing with pinhole cameras is to build an anaglyph pinhole camera. The anaglyph pinhole camera captures stereo images by projecting an anaglyph image on the projection plane.

We used one of the anaglyph glasses to get the filters and placed them in front of the two pinholes. Figure 40.4 shows two pictures of the two pinholes (figure 40.4[a]) and the filters positioned in front of each pinhole (figure 40.4[b]). Figure 40.4(d) is a resulting anaglyph image that appears on the projection plane (figure 40.4[c]); this picture was taken by placing a camera in the observation hole seen at the right of figure 40.4(c). Figure 40.4(d) should give you a 3D perception of the image when viewed with anaglyph glasses.

Here are experimental details for making the stereo pinhole camera. You will have to experiment with the distance between the two pinholes. We tried two distances: 3 in and 1.5 in. We found it easier to see the 3D for the 1.5 in images. Try to take good quality images. Once you have placed the color filters on each pinhole, you will need longer exposure times in order to get the same image quality than with the regular pinhole camera because the amount of light that enters the camera is smaller. If the images are too blurry, it will be hard to perceive 3D. Also, as you increase the distance between the pinholes, you will increase the range of distances that will provide 3D information because the parallax effect will increase. But if the images are too far apart, the visual system will not be able to fuse them and you will not see a 3D image when looking with the anaglyph glasses.

Let's now look into how to use two views to estimate the 3D structure of the observed scene.

40.3Model-Based Methods

Let's now make more concrete the intuitions we have built in the previous sections. We will start describing some model-based methods to estimate 3D from stereo images. Studying these models will help us understand the sources of 3D information, the constraints, and limitations that exist.

40.3.1Triangulation

The task of stereo is to triangulate the depth of points in the world by comparing the images from two spatially offset cameras.

> If more than two cameras are used, the task is referred to as multiview stereo.

To triangulate depth, we need to describe how image positions in the stereo pair relate to depth in the 3D world.

We will start with a very simple setting where both cameras are identical (same intrinsic parameters) and one is translated horizontally with respect to the other so that their optical axes are parallel, as shown in figure 40.5. We also will assume calibrated cameras. The geometry shown in figure 40.5 reveals the essence of depth estimation from the two cameras. Let's assume that there is one point, **P**, in 3D space and we know the locations in which it projects on each camera, $[x_L, y_L]^\mathsf{T}$ for the left camera and $[x_R, y_R]^\mathsf{T}$ for the right camera, as shown in figure 40.5(a). Because the two cameras are aligned horizontally, the y-coordinates for the projection in both cameras will be equal, $y_L = y_R$. Therefore, we can look at the top view of the setting as shown in figure 40.5(b). We consider the depth estimate within a single image row.

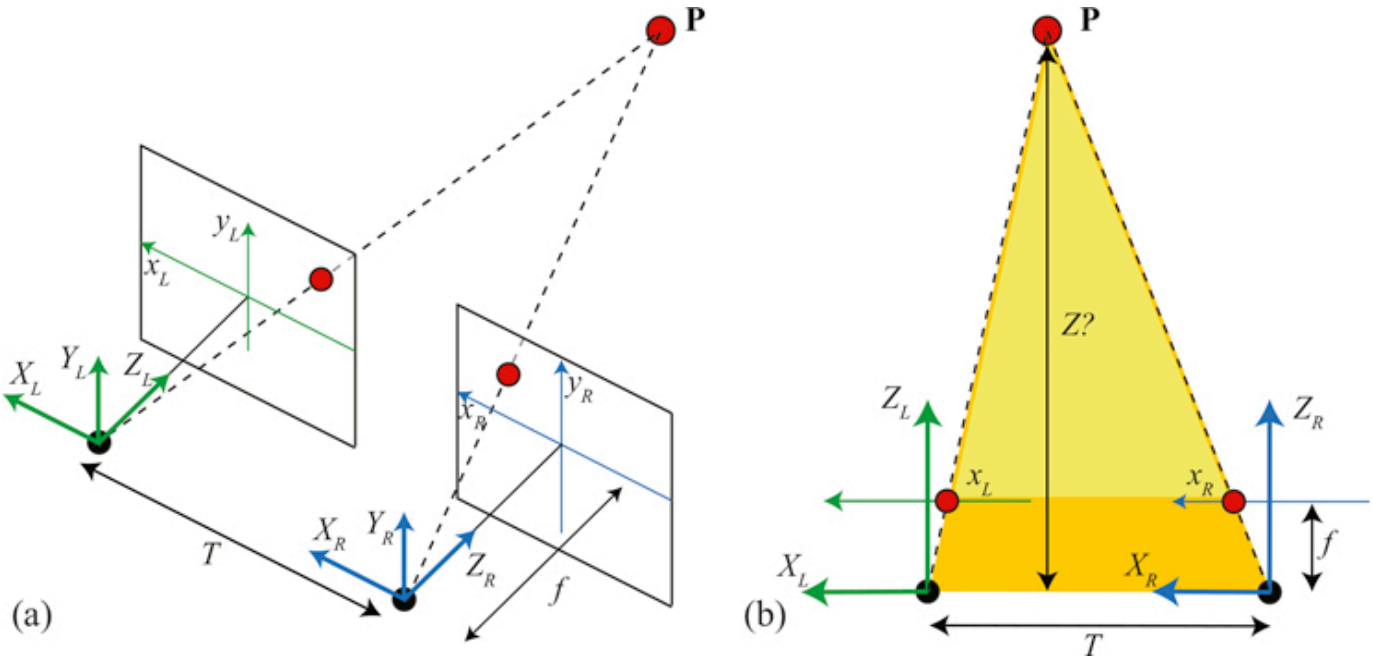


Figure 40.5: Simple stereo. (a) Two cameras, of identical focal length, f, and separated by an offset, T, image the point P onto the two-dimensional (2D) points $[x_L, y_L]^T$ and $[x_R, y_R]^T$ at each camera. (b) The similarity of the two triangles in leads to equation (40.4) for the depth, Z, of the point **P**.

The point **P**, for which we want to estimate the depth, Z, appears in the left camera at position x_L and in the right camera at position x_R.

> To distinguish 2D image coordinates from 3D world coordinates, for this chapter, we will denote 2D image coordinates by lowercase letters, and 3D world coordinates by uppercase letters.

A simple way to triangulate the depth, Z, of a point visible in both images is through similar triangles. The two triangles (yellow triangle and yellow+orange triangle) shown in figure 40.5(b) are similar, and so the ratios of their height to base must be the same. Thus we have:

$$\frac{T + x_R - x_L}{Z - f} = \frac{T}{Z} \tag{40.3}$$

Solving for Z gives:

$$Z = \frac{fT}{x_L - x_R} \tag{40.4}$$

The difference in horizontal position of any given world point, **P**, as seen in the left and right camera images, $x_L - x_R$, is called the **disparity**, and is *inversely proportional to the depth Z* of the point **P** for this configuration of the two cameras–parallel orientation. The task of inferring depth of any point in either image of a stereo pair thus reduces to measuring the disparity everywhere.

However, we are far from having solved the problem of perceiving 3D from two stereo images. One important assumption we have made is that we know the two corresponding projections of the point **P**. But in reality the two images will be complex and we will not know which points in one camera correspond to the same 3D point on the other camera. This requires solving the **stereo matching** problem.

40.3.2Stereo Matching

First, let's examine the task visually. Figure 40.6 shows two images of an office, taken approximately 1 m apart. The locations of several objects in figure 40.6(a) are shown in figure 40.6(b), revealing some of the disparities that we seek to measure automatically. As the reader can appreciate by comparing the stereo images by eye, one can compute the disparity by first localizing a feature point in one image and then finding the corresponding point in the other image, based on the visual similarity of the local image region. To do this automatically is the crux of the stereo problem.

(a) (b)

Figure 40.6: Two cameras, displaced from each other laterally, photograph the same scene, resulting in images (a) and (b). Colored rectangles show some identifiable features in image (a), and where those features appear in image (b). Arrows show the corresponding displacements, which reveal depth through equation (40.4).

A naive algorithm for finding the disparity of any pixel in the left image is to search for a pixel of the same intensity in the corresponding row of the right image. Figure 40.7, from [220], shows why that naive algorithm won't work and why this problem is hard. Figure 40.7(a) shows the intensities of one row of the right-eye image of a stereo pair. The squared difference of the intensity differences from offset versions of the same row in the rectified left-eye image of the stereo pair is shown in figure 40.7(b), for a range of disparity offsets, plotted vertically, in a configuration known as the **disparity space image** [425]. The disparity corresponding to the minimum squared error matches are plotted in red. Note the disparities of the best pixel intensity matches show much more disparity variations than would result from the smooth surfaces depicted in the image; the local intensity matches don't reliably indicate the disparity. Many effects lead to that unreliability, including: lack of texture in the image, which leads to noisy matches among the many nearly identical pixels; image intensity noise;

specularities in the image which change location from one camera view to another; and scene points appearing in one image but not another due to occlusion. Figure 40.7(d) shows an effective approach to remove the matching noise: blurring the disparity space image using an edge-preserving filter [220]. The best-matched disparities are plotted in red, and compare well with the ground truth disparities (figure 40.7[f]), as labeled in the dataset [425].

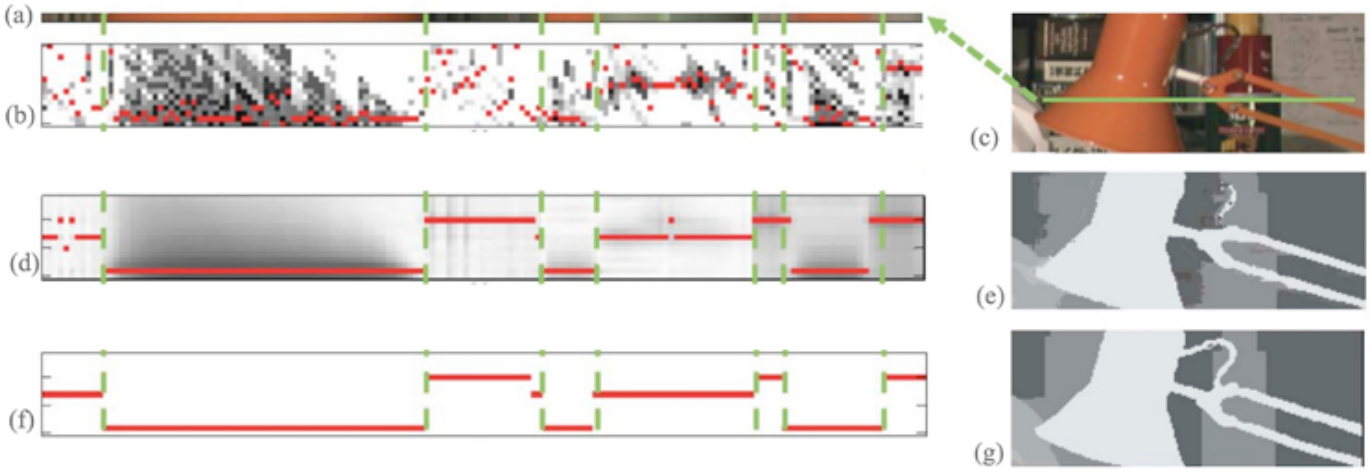


Figure 40.7: Issues with intensity-based stereo matching. Dataset from [425]. Figure modified from [220].

The task of finding disparity at each point is often broken into two parts: (1) finding features, and matching the features across images, and (2) interpolating between, or adjusting, the feature matches to obtain a reliable disparity estimate everywhere. These two steps are often called matching and filtering, because the interpolation can be performed using filtering methods.

We will describe next a classical method to find key points and extract image descriptors to find image correspondences. In chapter 44 we will describe other learning-based approaches.

40.3.2.1Finding image features

Good image features are positions in the image that are easy to localize, given only the image intensities. The **Harris corner detector** [186] identifies image points that are easy to localize by evaluating how the sum of squared intensity differences over an image patch change under a small translation of the image. If the squared intensity changes quickly with patch translation in every direction, then the region contains a good image feature.

To derive the equation of the Harris corner detector we will compute image derivatives. Therefore, we will write an image as a continuous function on x and y. The compute the final quantities in practice we will approximate the derivatives using their discrete approximations as discussed in chapter 18. Let the input image be $\ell(x, y)$, and let the small translation be Δx in x and Δy in y. Then, the squared intensity difference, $E(\Delta x, \Delta y)$, induced by that small translation, summed over a small image patch, P, is

$$E(\Delta x, \Delta y) = \sum_{(x,y)\in P} (\ell(x, y) - \ell(x + \Delta x, y + \Delta y))^2 \tag{40.5}$$

We expand $\ell(x + \Delta x, y + \Delta y)$ about $\ell(x, y)$ in a Taylor series:

$$\ell(x+\Delta x, y+\Delta y) \approx \ell(x,y) + \frac{\partial \ell}{\partial x}\Delta x + \frac{\partial \ell}{\partial y}\Delta y \tag{40.6}$$

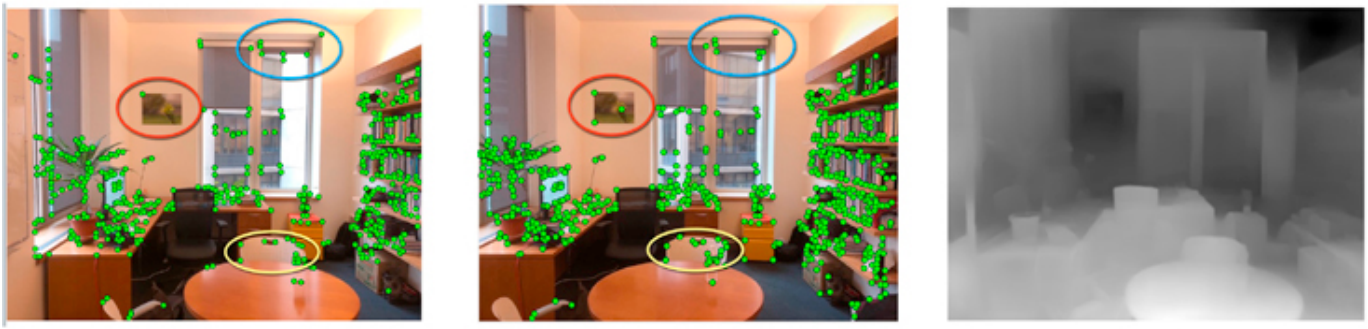
Substituting the above into equation (40.5) and writing the result in matrix form yields

$$E(\Delta x, \Delta y) \approx \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \mathbf{M} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{40.7}$$

where

$$\mathbf{M} = \sum_{(n,m)\in P} \begin{bmatrix} (\frac{\partial \ell}{\partial x})^2 & \frac{\partial \ell}{\partial x}\frac{\partial \ell}{\partial y} \\ \frac{\partial \ell}{\partial x}\frac{\partial \ell}{\partial y} & (\frac{\partial \ell}{\partial y})^2 \end{bmatrix} \tag{40.8}$$

The smallest eigenvalue, λmin of the matrix, **M**, indicates the smallest possible change of image intensities under translation of the patch in any direction. Thus, to detect good corners, one can identify all points for which $\lambda_{\min} > \lambda_c$, for some threshold λ_c. The Harris corner detector has been widely used for identifying features to match across images, although they were later supplanted by the scale-invariant feature transform (SIFT) [309], based on maxima over a scale of Laplacian filter outputs. Figures 40.8(a and b) show the result of the Harris feature detector to the images of figure 40.6.



(a) (b) (c)

Figure 40.8: The feature-based stereo approach, illustrated for the stereo pair of figure 40.6. (a and b) Feature points. (c) Depth image.

Figures 40.8(a and b) reveal the challenges of the feature matching task: (1) not every feature is marked across both images, and (2) we need to decide which pairs of image features go together. Feature points are found, independently, in each of the stereo pair images (figures 40.8[a and b]). Detected Harris feature points [186] are shown as green dots. It is especially visible within the marked ellipses that not the same set of feature points is marked in each image. Figure 40.8(c) shows a depth image resulting from a matched and interpolated set of feature points.

40.3.2.2Local image descriptors

Matching corresponding image features require a local description of the image region to allow matching like regions across two images. SIFT [309] uses histograms of oriented filter responses to create an image descriptor that is sensitive to local image structure, but that is insensitive to minor changes in lighting or location which may occur across the two images of a stereo pair.

Figure 40.9 shows a sketch of the use of oriented edges and SIFT descriptors in image analysis. The top row of figure 40.9(a) shows two images of a hand under different lighting conditions. Despite that both images look quite different in a pixel intensity representation, they look quite similar in a local orientation, depicted in the bottom row by line segment directions. SIFT image descriptors [309] exploit that observation. Taking histograms (figure 40.9[b]) of local orientation over local spatial regions further allow for image descriptors to be robust to small image translations [145, 309].

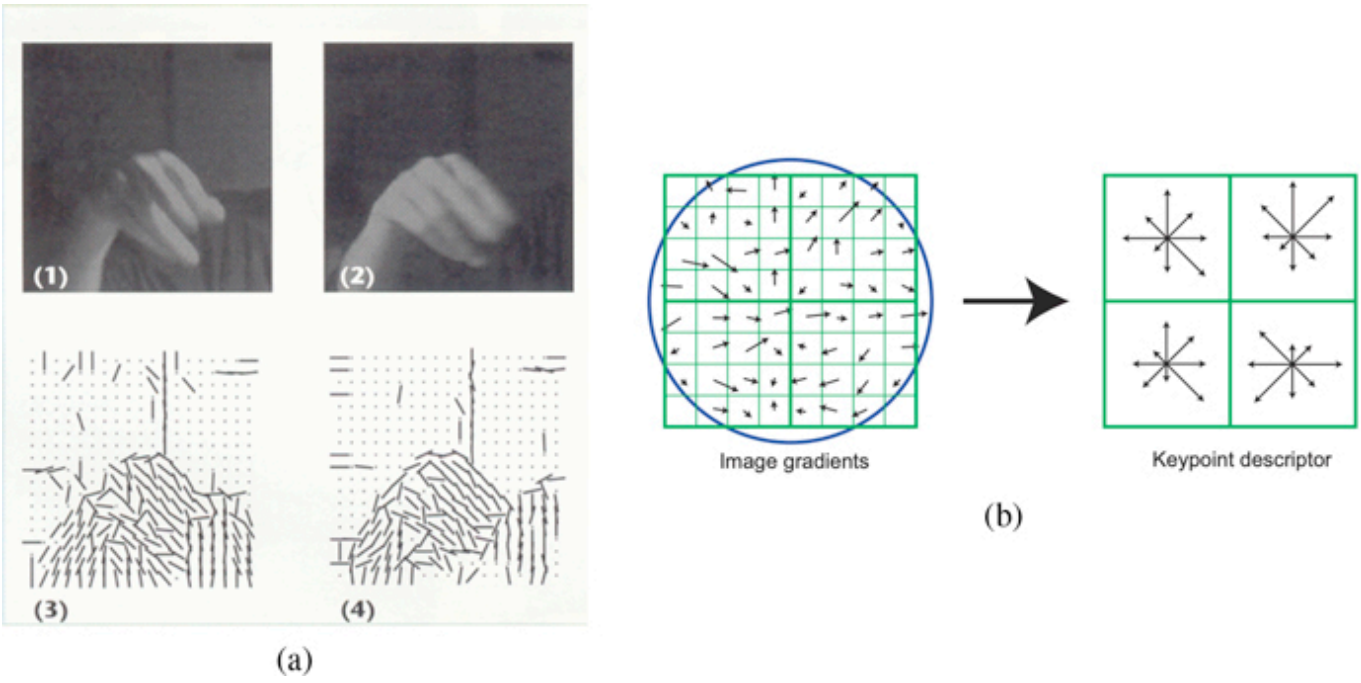


Figure 40.9: Orientation-based image descriptors. (a) Oriented features. Reprinted from [145]. (b) SIFT image descriptors [309].

40.3.2.3Interpolation between feature matches

Methods to interpolate depth between the depths of matched points include probabilistic methods such as belief propagation, discussed in chapter 29. Under assumptions about the local spatial relationships between depth values, optimal shapes can be inferred provided the spatial structure over which inference is performed is a chain or a tree. Note that [207] uses a related inference algorithm, dynamic programming, which has the same constraints on spatial structure for exact inference. Hirschmuller's algorithm, called semiglobal matching (SGM), applies the one-dimensional processing over many different orientations to approximate a global solution to the stereo shape inference problem.

40.3.3Constraints for Arbitrary Cameras

Until now we have considered only the case where the two cameras are parallel, but in general the two images will be produced by cameras that can have arbitrary orientations. Note that disparity of the smokestacks in figure 40.1(a) increases with depth, rather than decreasing as described by equation (40.4). That is because the cameras that took the stereo pair were not in the same orientation, related only by a translation.

Let's assume we have two calibrated cameras: we know the intrinsic and extrinsic parameters for both. They are related by a generic translation, **T**, and a rotation, **R**, such that their optical axis are not parallel. Let's now consider that we see one point in the image produced by camera 1 and we want to identify where is the corresponding point in the image from camera 2. The following sketch (figure 40.10) shows the setting of the problem we are trying to solve.

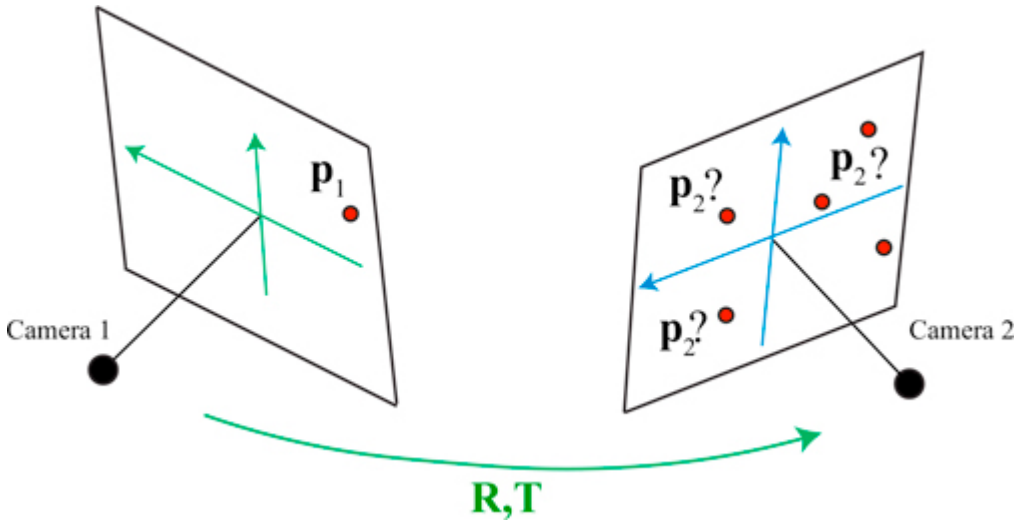


Figure 40.10: How can we identify the specific point $\mathbf{p}_2$ on camera 2 that represents the projection of the same 3D point as $\mathbf{p}_1$ on camera 1?

Are there any geometric constraints that can help us to identify candidate matches? We know that the observed point in camera 1, $\mathbf{p}_1$, corresponds to a 3D point that lies within the ray that connects the camera 1 origin and $\mathbf{p}_1$ (shown in red in figure 40.11):

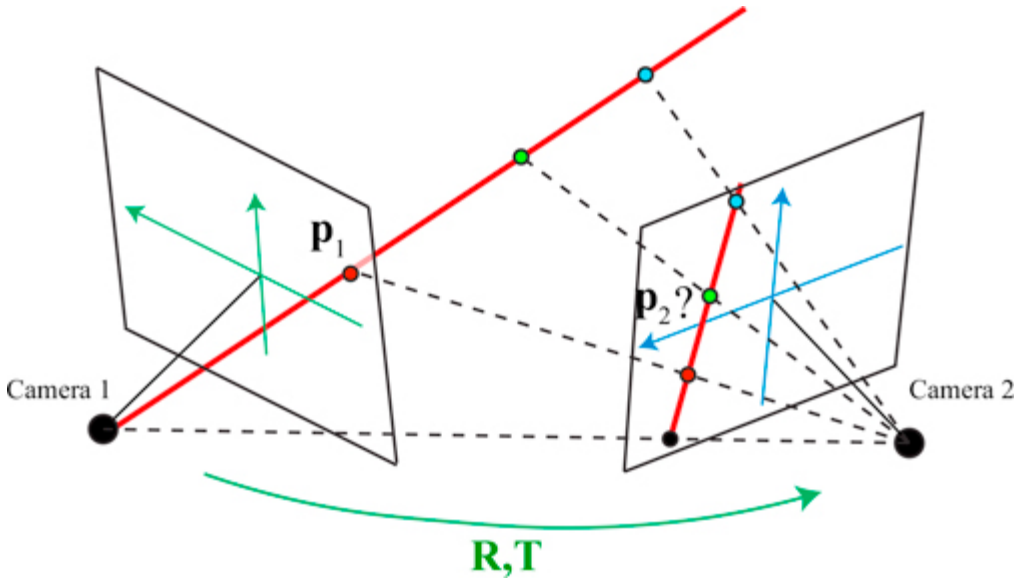


Figure 40.11: The point $\mathbf{p}_1$ corresponds to a 3D point that lies within the ray that connects the camera 1 origin and $\mathbf{p}_1$. The point $\mathbf{p}_2$ is constrained to be within the projection of that line on camera 2.

Therefore, $\mathbf{p}_2$ is constrained to be within the line that results of projecting the ray into camera 2. Remember that a line in 3D projects into a line in 2D. There are two points that are of particular interest in the line projected into camera 2. The first point is the projection of the camera center of camera 1 into the camera plane 2 (the point could lay outside the camera view but in this example it happens to be visible). The second point is the projection of $\mathbf{p}_1$ itself into camera 2. As we know all the intrinsic and extrinsic camera parameters, those two points should be easy to calculate. And these two points will define the equation of the line (there are many other ways to think about this problem).

The following figure shows the geometry of a stereo pair. Given a 3D point, **P**, in the world, we define the following objects:

- The **epipolar plane** is the plane that contains the two camera origins and a 3D point, **P**.
- The **epipolar lines** are the intersection between the epipolar plane and the camera planes. When the point **P** moves in space, the epipolar plane and epipolar lines change.
- The **epipoles** are the intersection points between the line that connects both camera origins and the camera planes. The epipoles' location does not depend on the point **P**. Therefore, all the epipolar lines intersect at the same epipoles.

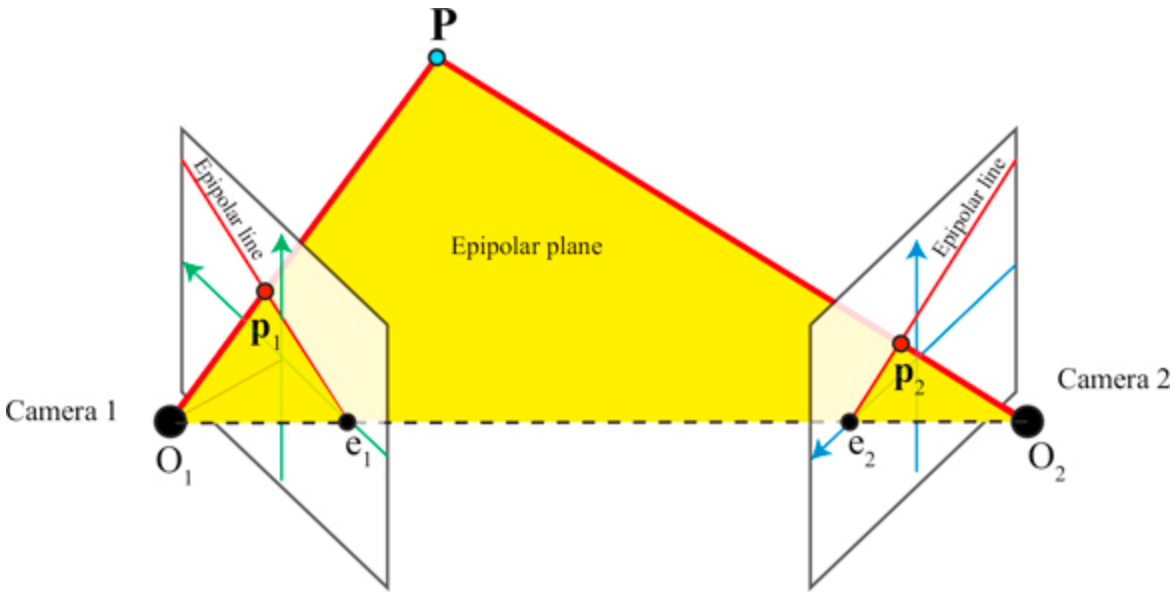


Figure 40.12: Geometry of a stereo pair and terminology. The figure shows the epipolar plane, epipolar lines, and epipoles (e_1 and e_2), for a given point **P**. As we move the point **P**, only the epipoles remain constant.

In the example shown in figure 40.12, as both cameras are only rotated along their vertical axis and there is no vertical displacement between them, the epipoles are contained in the x axes of both camera coordinate systems. The epipolar line in camera 2, corresponds to the projection of the line O_1P into the camera plane. Therefore, given the image point $\mathbf{p}_1$, we know that the match should be somewhere along the epipolar line.

The epipolar line in camera 2 that corresponds to $\mathbf{p}_1$ is uniquely defined by the camera geometries and the location of image point $\mathbf{p}_1$. So, when looking for matches of a point in one camera, we only need to look along the corresponding epipolar line in the other camera. The same applies for finding matches in camera 1 given a point in camera 2.

In the next section we will see how to find the epipolar lines mathematically.

40.3.4 The Essential and Fundamental Matrices

We will follow here the derivation that was proposed by Longuet-Higgins, in 1981 [306]. Let's consider a 3D point, **P**, viewed by both cameras. The coordinates of this 3D point **P** are $\mathbf{P}_1 = [X_1, Y_1, Z_1]^\mathsf{T}$ when expressed in the coordinates system of camera 1, and $\mathbf{P}_2 = [X_2, Y_2, Z_2]^\mathsf{T}$ when expressed in the coordinates system of camera 2. Both systems are related by (using heterogeneous coordinates):

$$\mathbf{P}_2 = \mathbf{R}\mathbf{P}_1 + \mathbf{T} \tag{40.9}$$

The 3D point $\mathbf{P}_1$ projects into the camera 1 at the coordinates $\mathbf{p}_1 = f\,\mathbf{P}_1/Z_1$ where $\mathbf{p}_1$ is expressed also in 3D coordinates $\mathbf{p}_1 = [fX_1/Z_1, fY_1/Z_1, f]^\mathsf{T}$. Without losing generality we will assume $f = 1$.

Taking the cross product of both sides of equation (40.9) with **T** we get:

$$\mathbf{T} \times \mathbf{P}_2 = \mathbf{T} \times \mathbf{R}\mathbf{P}_1 \tag{40.10}$$

as $\mathbf{T} \times \mathbf{T} = 0$. And now, taking the dot product of both sides of equation (40.10) with $\mathbf{P}_2$:

$$\mathbf{P}_2^\top \mathbf{T} \times \mathbf{P}_2 = \mathbf{P}_2^\top \mathbf{T} \times \mathbf{R}\mathbf{P}_1 \tag{40.11}$$

The left side is zero because $\mathbf{T} \times \mathbf{P}_2$ results in a vector that is orthogonal to both $\mathbf{T}$ and $\mathbf{P}_2$, and the dot product with $\mathbf{P}_2$ is zero (dot product of two orthogonal vectors is zero). Therefore, $\mathbf{P}_1$ and $\mathbf{P}_2$ satisfy the following relationship:

$$\mathbf{P}_2^\top (\mathbf{T} \times \mathbf{R}) \mathbf{P}_1 = 0 \tag{40.12}$$

The cross product of two vectors $\mathbf{a} \times \mathbf{b}$ can be written in matrix form as:

$$\mathbf{a} \times \mathbf{b} = \mathbf{a}_\times \mathbf{b}$$

The special matrix $\mathbf{a}_\times$ is defined as:

$$\mathbf{a}_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

This matrix $\mathbf{a}_\times$ is rank 2. In addition, it is a skew-symmetric matrix (i.e., $\mathbf{a}_\times = -\mathbf{a}_\times^\top$). It has two identical singular values and the third one is zero.

The matrix $\mathbf{E}$ is called the **essential matrix** and it corresponds to:

$$\mathbf{E} = \mathbf{T} \times \mathbf{R} = \mathbf{T}_\times \mathbf{R} \tag{40.13}$$

This results in the relationship:

$$\mathbf{P}_2^\top \mathbf{E} \mathbf{P}_1 = 0 \tag{40.14}$$

If we divide equation (40.14) by $Z_1 Z_2$ we will obtain the same relationship but relating the image coordinates:

$$\mathbf{p}_2^\top \mathbf{E} \mathbf{p}_1 = 0 \tag{40.15}$$

This equation relates the coordinates of the two projections of a 3D point into two cameras. If we fix the coordinates from one of the cameras, the equation reduces to the equation of a line for the coordinates of the other camera, which is consistent with the geometric reasoning that we made before.

The essential matrix is a 3×3 matrix with some special properties:

- $\mathbf{E}$ has five degrees of freedom instead of nine. To see this, let's count degrees of freedom: the rotation and translation have three degrees of freedom each, this gives six; then, one degree is lost

in equation (40.15) as a global scaling of the matrix does not change the result. This results in five degrees of freedom.

- The essential matrix has rank 2. This is because **E** is the product of a rank 2 matrix, $\mathbf{T}_\times$, and a rank 3 matrix, **R**.
- The essential matrix has two identical singular values and the third one is zero. This is because **E** is the product of $\mathbf{T}_\times$, which has two identical singular values, and the rotation matrix **R** that does not change the singular values.

As a consequence, not all 3 × 3 matrices correspond to an essential matrix which is important when trying to estimate **E**.

40.3.4.1The fundamental matrix

We have been using camera coordinates, **p**. If we want to use image coordinates, in homogeneous coordinates, $\mathbf{c} = [n, m, 1]^\mathsf{T}$, we need to also incorporate the intrinsic camera parameters (which will also incorporate pixel size and change of origin). Incorporating the transformation from camera coordinates, **p**, to image coordinates, **c**, the equation relating image coordinates in both cameras has the same algebraic form as in equation (40.15):

$$\mathbf{c}_2^\mathsf{T}\mathbf{F}\mathbf{c}_1 = 0 \tag{40.16}$$

where **F** is the **fundamental matrix**.

The fundamental and essential matrices are related via the intrisic camera matrix, namely:

$$\mathbf{F} = \mathbf{K}^{-\mathsf{T}}\mathbf{E}\mathbf{K}^{\mathsf{T}} \tag{40.17}$$

The fundamental matrix has rank 2 (like the essential matrix) and 7 degrees of freedom (out of the 9 degrees of freedom, one is lost due to the scale ambiguity and another is lost because the determinant has to be zero).

40.3.4.2Estimation of the essential/fundamental matrix

Given a set of matches between the two views we can estimate either the essential or the fundamental matrix by writing a linear set of equations. Any two matching points will satisfy $\mathbf{p}_2^\mathsf{T}\mathbf{E}\mathbf{p}_1 = 0$ and we can use this a linear set of equations for the coefficients of **E**. However, the matrix **E** has a very special structure that allows more efficient and robust methods to estimate the coefficients. The same applies for the fundamental matrix.

As both matrices are rank 2, the minimization the linear set of equations can be done using a constrained minimization or directly optimizing a rank 2 matrix. You can also delete the third singular value **F**, and, for **E**, you can delete the third singular value and set the first and second singular values to be equal.

Once **E** is estimated, it can be decomposed to recover the translation (skew-symmetric matrix) and rotation (orthonormal matrix) between the two cameras using the singular value decomposition

methods.

40.3.4.3Finding the epipoles

As the epipoles, $\mathbf{e}_1$ and $\mathbf{e}_2$, belong to the epipolar lines, they satisfy

$$\mathbf{p}_2^T \mathbf{E} \mathbf{e}_1 = 0 \tag{40.18}$$

$$\mathbf{e}_2^T \mathbf{E} \mathbf{p}_1 = 0 \tag{40.19}$$

and, as the epipoles are the intersection of all the epipolar lines, both relationships shown previously have to be true for any points $\mathbf{p}_1$ and $\mathbf{p}_2$. Therefore,

$$\mathbf{E} \mathbf{e}_1 = 0 \tag{40.20}$$

$$\mathbf{e}_2^T \mathbf{E} = 0 \tag{40.21}$$

As the matrix $\mathbf{E}$ is likely to be noisy, we can compute the epipoles as the eigenvectors with the smallest eigenvalues of $\mathbf{E}$ and $\mathbf{E}^T$.

40.3.4.4Epipolar lines: The game

In order to build an intuition of how epipolar lines work we propose to play a game. Figure 40.13 shows a set of camera pairs and a set of epipolar lines. Can you guess what camera pair goes with what set of epipolar lines?[20]

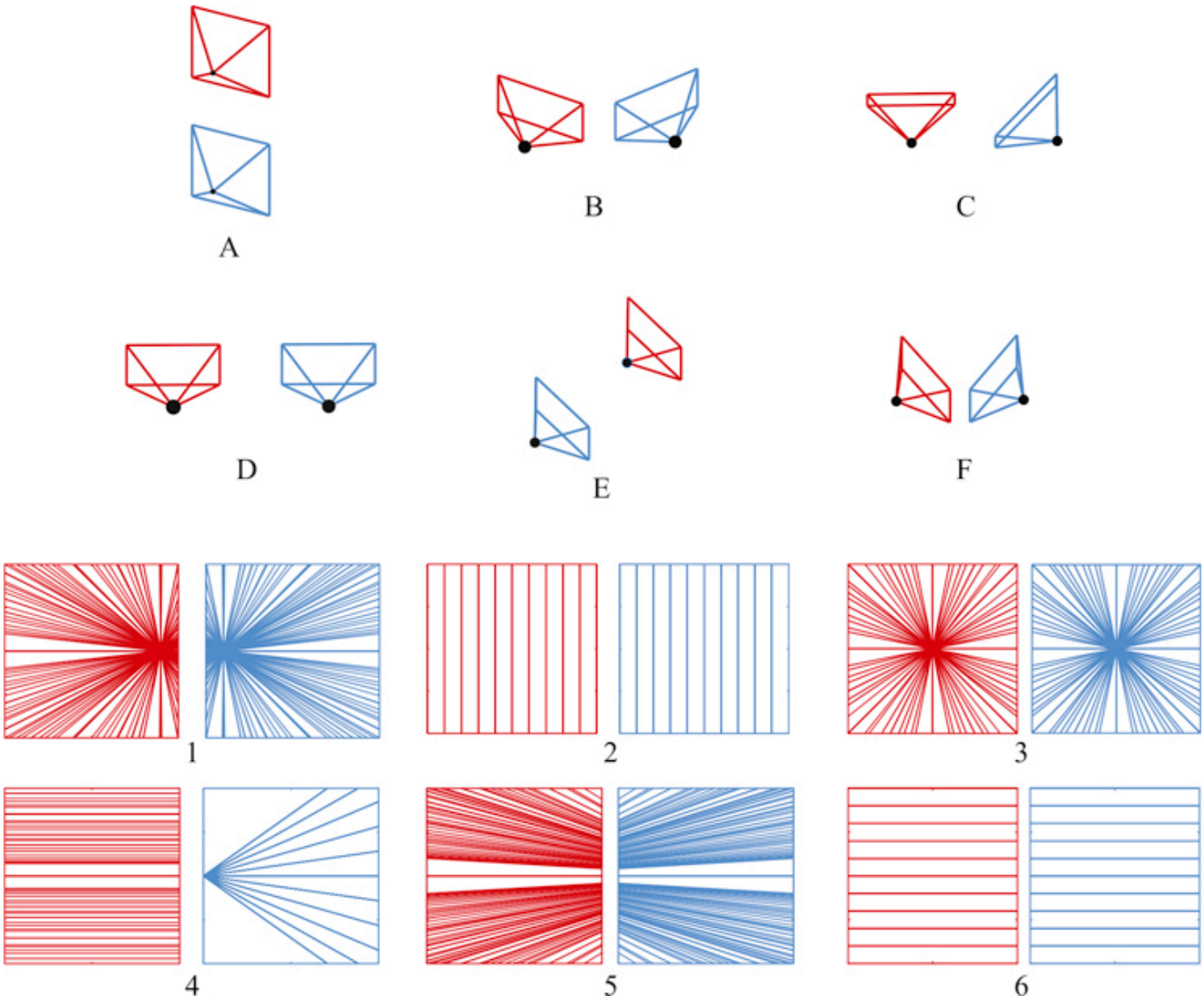


Figure 40.13: Epipolar game. The figure shows six camera pair arrangements, and six sets of epipolar lines. Can you identify which camera pairs correspond to each set of epipolar lines?

To solve the game, visualize mentally the cameras and think about how rays from the origin of one camera will be seen from the other camera. The epipole corresponds to the location where one camera sees the origin of the other camera.

Observe that most of the sets of epipolar lines shown in the figure are symmetric across the two cameras, but it is not always true. What conditions are needed to have symmetric epipolar lines across both cameras? What camera arrangements will break the symmetry? If you were to place two cameras in random locations, do you expect to see symmetric epipolar lines?

It is also useful to use the equations defining the epipolar lines and deduce the equations for the epipolar lines for some of the special cases shown in figure 40.13.

When building systems, it is crucial to visualize intermediate results in order to find bugs in code. This requires knowing what the results should look like. The game from figure 40.13 helps building that understanding.

40.3.5Image Rectification

As described previously, one only needs to search along epipolar lines to find matching points in the second image. If the sensor planes of each camera are coplanar, and if the pixel rows are colinear across the two cameras, then the epipolar lines will be along image scanlines, simplifying the stereo search.

Using the homography transformations (chapter 41), we can warp observed stereo camera images taken under general conditions to synthesize the images that would have been recorded had the cameras been aligned to yield epipolar lines along scan rows.

Many configurations of the two cameras satisfy the desired image rectification constraints [521]: (1) that epipolar lines are along image scanlines, and (2) that matching points are within the same rows in each camera image. One procedure that will satisfy those constraints is as follows [503]. First, rotate each camera to be parallel with each other and with optical axes perpendicular to the line joining the two camera centers. Second, rotate each of the cameras about its optical axis to align the rows of each camera with those of the other. Third, uniformly scale one camera's image to be the same size as to the other, if necessary. The resulting image pair will thus satisfy the two requirements above for image rectification. Image rectification uses image warping with bilinear (or other) interpolation as described in section 21.4.2.

Other algorithms are optimized to minimize image distortions [521] and may give better performance for stereo algorithms. We assume camera calibrations are known, but other rectification algorithms can use weaker conditions, such as knowing the fundamental matrix [187, 390]. Many software packages provide image rectification code [307].

40.4Learning-Based Methods

As is the case for most other vision tasks, neural network approaches now dominate stereo methods. The two stages of a stereo algorithm, matching and filtering, can be implemented separately by neural networks [516], or together in a single, end-to-end optimization ([519, 75, 327, 255].

40.4.1Output Representation

One important question is how we want to represent the output. We are ultimately interested in estimating the 3D structure of the scene. We can estimate the disparity or the depth. Disparity might be easier to estimate as it is independent of the camera parameters. It is also easier to model noise, as we can assume that noise is independent of disparity. However, when using depth, estimation error will be larger for distant points.

Another interesting property of disparity is that for flat surfaces, the second-order derivative along any spatial direction, **x**, is zero:

$$\frac{\partial^2 d}{\partial \mathbf{x}^2} = 0. \tag{40.22}$$

This equality does not hold for depth. This property allows introducing a simple regularization term that favors planarity.

To translate the returned disparity for a given pixel $d[n, m]$ into depth values $z[n, m]$, we need to invert the disparity, that is

$$z[n,m] = \frac{1}{d[n,m]} \tag{40.23}$$

Now we need to recover the world coordinates for every pixel. Assuming a pinhole camera and using the depth, $z[n, m]$, and the intrinsic camera parameters, **K**, we obtain the 3D coordinates for a pixel with image coordinates $[n, m]$ in the camera reference frame as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = z[n,m] \times \mathbf{K}^{-1} \times \begin{bmatrix} n \\ m \\ 1 \end{bmatrix} \tag{40.24}$$

Note that **K** are the intrinsic camera parameters for the reference image.

40.4.2Two-Stage Networks

Given two images, $\mathbf{v}_1$ and $\mathbf{v}_2$, the disparity at each location can be computed by a function, that is:

$$\hat{\mathbf{d}} = f_\theta(\mathbf{v}_1, \mathbf{v}_2) \tag{40.25}$$

The objective is that the estimated disparities, $\hat{\mathbf{d}}$, should be close to ground truth disparities, **d**. One example of training set are the KITTI dataset [157] and the scene flow dataset [327], which contain ground truth disparities from calibrated stereo cameras. One typical loss is:

$$J(\theta) = \sum_t \left| \mathbf{d} - f_\theta(\mathbf{v}_1, \mathbf{v}_2) \right|^2 \tag{40.26}$$

This is optimized by gradient descent as usual.

When matching and filtering are implemented separately, one common formulation is to use one neural network to extract features from both images, $g(\mathbf{i}_1)$ and $g(\mathbf{i}_2)$, and one neural network to compute the matching cost and estimate the disparities at every pixel.

$$f(\mathbf{v}_1, \mathbf{v}_2) = c(g(\mathbf{v}_1), g(\mathbf{v}_2)) \tag{40.27}$$

A common approach, depicted in block diagram form in figure 40.14, is the following:

- Rectify the stereo pair so that scanlines in each image depict the same points in the world.
- Extract features from each image, using a pair of networks with shared weights.
- Form a 3D cost volume indicating the local visual evidence of a match between the two images for each possible pixel position and each possible stereo disparity. (This 3D cost volume can be referenced to the *H* and *V* positions of one of the input images.)
- Train and apply a 3D convolutional neural net (CNN) to aggregate (process) the costs over the 3D cost volume in order to estimate a single, best disparity estimate for each pixel position. This performs the function of the regularization step of nonneural-network approaches, such as SGM [207], but the performance is better for the neural network approaches.

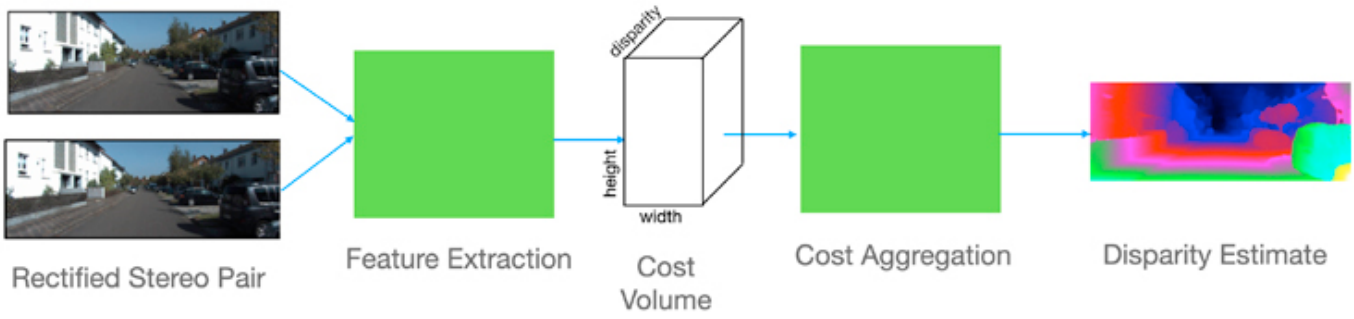


Figure 40.14: Block diagram of CNN stereo processing, abstracted from the block diagrams of [75, 519, 255].

The performance of methods following the block diagram of figure 40.14 surpassed that of methods with handcrafted features. More recent work has addressed the problem of poor generalization to out-of-training-domain test examples [520]. Training and computing the 3D neural network to process the cost volume can be expensive in time and memory and recent work has obtained both speed and performance improvement through developing an iterative algorithm to avoid the 3D filtering [301].

40.5Evaluation

The Middlebury stereo web page [425] provides a comprehensive comparison of many stereo algorithms, compared on a common dataset, with pointers to algorithm descriptions and code. One can use the web page to evaluate the improvement of stereo reconstruction algorithms over time. The advancement of the field over time is very impressive.

40.6Concluding Remarks

Accurate depth estimation from stereo pairs remains an unsolved problem. The results are still not reliable. In order to get accurate depth estimates, most methods require many images and the use of multiview geometry methods to reconstruct the 3D scene.

Stereo matching is often also formulated as a problem of estimating the image motion between two

views captured at different locations. In chapter 46 we will discuss motion estimation algorithms.

We have only provided a short introduction to stereo vision, for an in depth study of the geometry of stereo pairs the reader can consult specialized books such as [187] and [122].

---

20. Answer for the game in figure 40.13: A-2, B-5, C-4, D-6, E-3, F-1.

41Homographies

41.1Introduction

In chapter 38 we discussed several types of geometric transformations of two-dimensional (2D) images: translations, rotations, skewing, scalings, and we saw that all could be modeled as the product between the coordinates of a point in the input image described using homogeneous coordinates and a 3×3 matrix. Now that we have discussed camera models, we are well equipped to present a more general geometric image transformation that opens the door to many applications: the homography.

As we discussed in the previous chapter 40, if we capture a scene with two cameras from different viewpoints, corresponding points across both images are constrained by the epipolar constrain. In general, there is not a simple geometric transformation that maps pixels from one image into the pixels of the other image. To find that mapping we will need to know the three-dimensional (3D) location of each point as well as the relative camera translation and rotation between both images.

However, there are a few scenarios where we can find a simple transformation that allows warping one image into another image corresponding to a new camera location without requiring knowing the 3D scene structure: this will happen when the scene is planar (as in figure 41.1) or when the two cameras are related just by a rotation (figure 41.4). In those two cases the coordinates of corresponding points across the two images are related by a **homography**.

Figure 41.1: These two images are (approximately) related by an homography.

Homographies can be used in applications such as perspective correction, augmented reality, image stitching, creating bird's-eye views, correcting keystone distortions for projected images, and many others.

One popular tool available in many cameras and photography editing tools is the option to combine multiple overlapping images into a large panorama. The **homography** is what is behind those tools.

41.2Homography

Let's start with a formal definition of the homography. A homography (or projective transformation) is a geometric transformation that preserves straight lines. The homography is a function h that maps points into points, $\mathbf{p}' = h(\mathbf{p})$, with the property that if points $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are colinear, then the transformed points $h(\mathbf{p}_1)$, $h(\mathbf{p}_2)$, and $h(\mathbf{p}_3)$ are also colinear. Such a function, $\mathbf{p}' = h(\mathbf{p})$, can be written in homogeneous coordinates as a product with a matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{41.1}$$

This can be written in the short form:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{41.2}$$

where $\mathbf{p}$ and $\mathbf{p}'$ correspond to 2D points in homogeneous coordinates, and $\mathbf{H}$ is a 3×3 matrix.

To prove that colinear points remain colinear after the homography we can use the equation of a line in homogeneous coordinates, $\boldsymbol{l}^{\mathsf{T}}\mathbf{p} = 0$. From equation (41.2), we have that $\mathbf{p} = \mathbf{H}^{-1}\mathbf{p}'$. Replacing the last equality into the equation of the line gives $(\mathbf{H}^{-\mathsf{T}}\boldsymbol{l})^{\mathsf{T}}\mathbf{p}' = 0$, which corresponds to the equation of the line for the projected points $\mathbf{p}'$.

In a general homography, angles and lengths are not preserved. Therefore, parallel lines might not remain parallel, and lines of identical length might have different lengths after a homography. Figure 41.2 shows how a planar grid gets transformed after applying a homography.

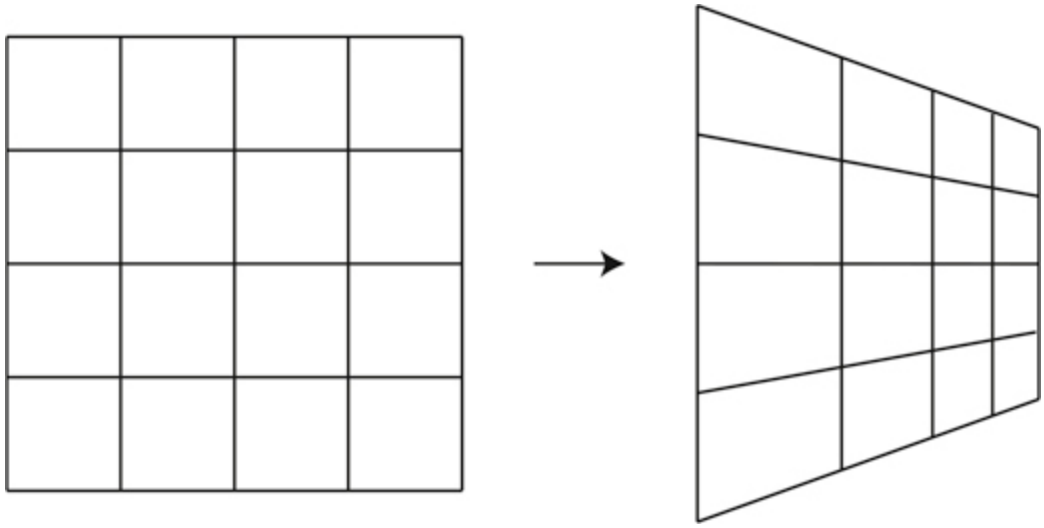


Figure 41.2: Homography. Colinear points remain colinear after transformation. Angles between lines are not preserved.

Let's start discussing in which scenarios the coordinates of points between to images are related by a homography.

41.2.1 Camera Rotation

Consider the following three images shown in figure 41.3.

Figure 41.3: Three pictures taken by rotating the camera while trying to keep the camera center fixed.

The three images are taken from the same point by rotating the camera. There is no translation between the three camera positions as the photographer took the three images by keeping the camera origin static when moving the camera. Under this condition, the coordinates of the corresponding points across each pair of images are related by a homography independently of how far they are from the camera.

Here, we show that the mapping between feature point locations in two cameras differing only in their 3D orientation, as shown in figure 41.4, is a 3 × 3 matrix transformation of their 2D positions written in homogeneous coordinates, that is, a homography.

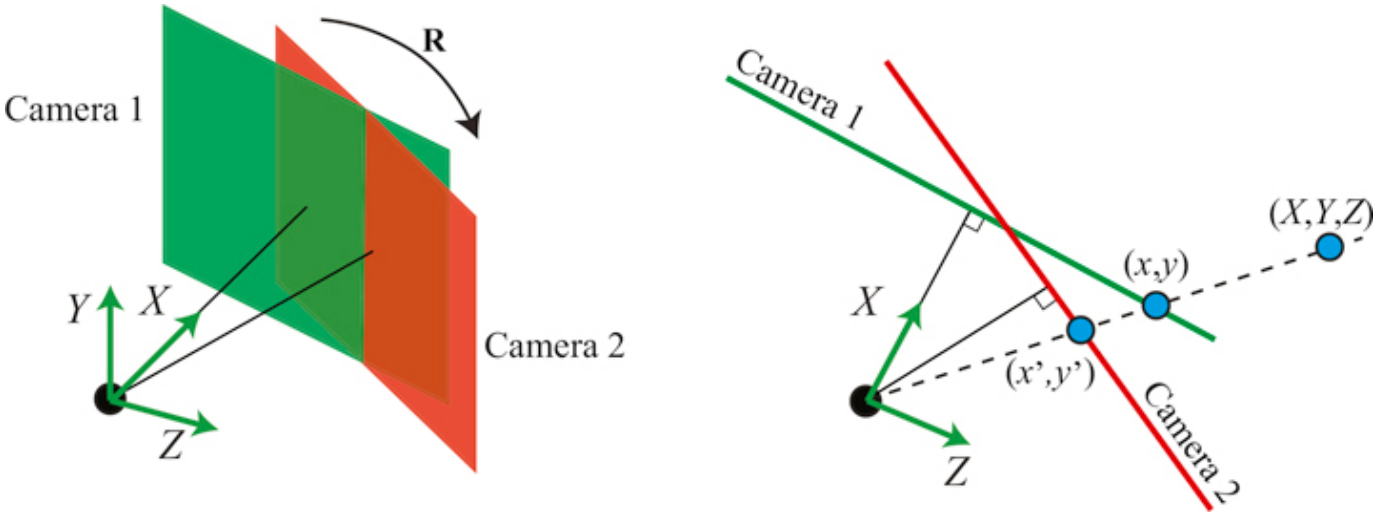


Figure 41.4: Camera rotation generates two images related by a homography.

We start by setting the world coordinate system aligned with the coordinate system of camera 1, as shown in figure 41.4. Therefore, the projection of a 3D point, $\mathbf{P} = [X, Y, Z]^{\mathsf{T}}$, into the camera 1 plane gives, setting the translation vector to be zero:

$$\lambda_1 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{41.3}$$

We can now project the same 3D point into camera 2. Setting the translation vector to be zero in equation (39.22), and writing the 3 × 3 rotation matrices for camera 2 as **R**, we have for the observed position of a common 3D point observed from camera 2:

$$\lambda_2 \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{41.4}$$

As both cameras see the same 3D point, $\mathbf{P} = [X, Y, Z]^{\mathsf{T}}$, we can invert the two projection equations (41.10) and (41.4) to get the following relationship:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \lambda_1 \mathbf{K}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \lambda_2 \mathbf{R}^{-1} \mathbf{K}^{-1} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{41.5}$$

Observe that equation (41.5) does not mean that we can recover the 3D coordinates of a point from the image coordinates. We know that this is impossible, so what is the issue? The subtlety is that the equations are written as functions of λ_1 and λ_2, which we do not know when only given image coordinates. Therefore, the equations only specify the equation of a ray when looking at all possible values of λ_1/λ_2.

Using the last equality, we can establish the following relationship between the corresponding image points, in homogeneous coordinates:

$$\lambda_2/\lambda_1 \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}\mathbf{K}^{-1} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{41.6}$$

We can write the relationship between corresponding points as

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{41.7}$$

where **H** is a 3 × 3 matrix. Thus, camera rotation modifies the locations of the image points by a homography. The relationship also holds if the two cameras have different intrinsic camera parameters. This concludes the proof.

Under certain conditions, a homography predicts the position of points when viewed with another camera from another position. We just discussed one condition, when the two cameras differ in their position by a rotation about a common center of projection. We will now consider another case.

41.2.2Planar Surface

A second case is when two cameras observe a 2D plane [187]. In that case, the coordinates of corresponding points across the two camera views, for points inside the plane, are related by a homography (figure 41.5).

Figure 41.5: Two pictures of a building facade taken from different positions. The coordinates of corresponding points on the planar facade are related by a homography.

To prove this, we can first show that the coordinates inside a plane relate to the coordinates in the image plane by a homography independently of relative position between the plane and the camera. Therefore, the relationship between points across two images will also be related by a homography.

We start by placing the world-coordinate system on the plane so that the plane is defined by $Z = 0$ as shown in the figure 41.6. Note that the origin can be placed at any arbitrary location in the world.

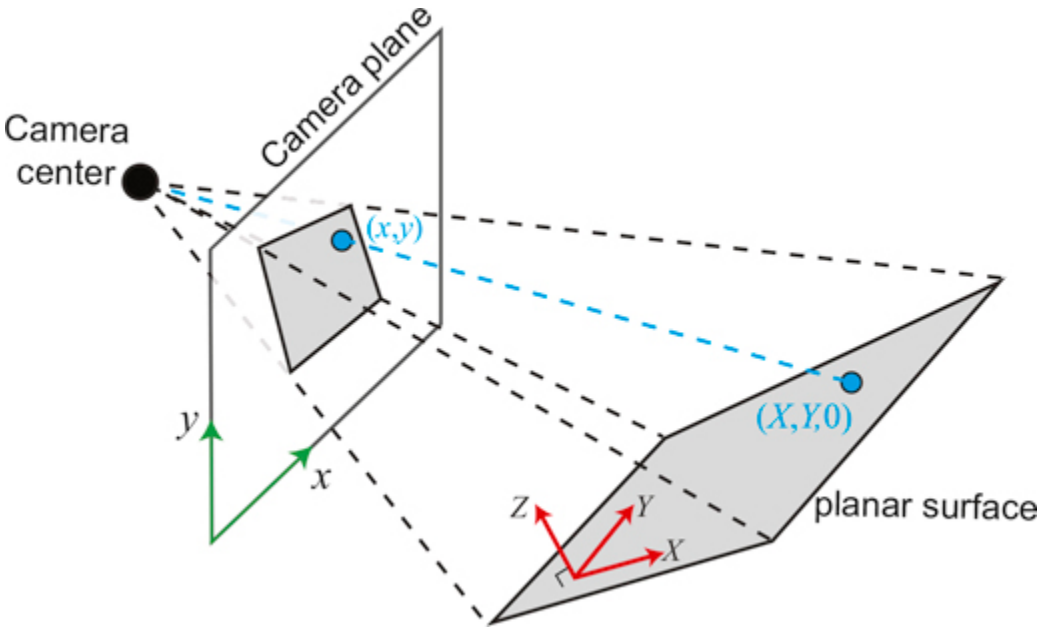


Figure 41.6: Geometry of a the projection of a planar scene. The origin of the world-coordinates system is placed inside the plane and the Z-axis is perpendicular to it.

The proof is similar to what we did in the previous section. We start writing the projection equation

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{41.8}$$

Note that we are not using the same notation for the translation vector as we used in equation (39.17). You can obtain the same by setting $\mathbf{t} = -\mathbf{RT}$, where **R** and **T** are the translation and rotation of the camera with respect to the world-coordinate system. As we can put the world-coordinate system in any arbitrary location, we chose the world-coordinates system to be so that points in the plane have coordinates $Z = 0$ as shown in figure 41.6. Therefore, we can write:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix} \tag{41.9}$$

Where c_i is the column i of the rotation matrix **R**. As the last expression contains the product of two 3 × 3 matrices, we can write

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{41.10}$$

This last result shows that the coordinates of points on the plane are related to the corresponding points in the image plane by a homography.

As equation (41.10) is true for any camera, the relationship between corresponding points in the two cameras will also be related by a homography when the points correspond to points in a plane in the 3D space.

> A homography is a stronger constraint than the epipolar constraint between corresponding points across two camera views. However, the homography only applies under certain conditions as we have seen, while the epipolar constraint always holds.

41.3Creating Image Panoramas

Let's now study a popular application of the homography: stitching images to create large panoramas. To create a large panorama from multiple pictures, the images need to be taken by a rotating camera without translation.

To stitch together images, one needs to estimate the homography relating the images to be stitched together. While one could estimate the rotation and intrinsic camera matrices within equation (41.6) to compute the homography, it is usually simpler to follow the procedure below.

We will start assuming we have some initial correspondences between each pair of images as shown in figure 41.7. The correspondences in this example are computer using **speeded up robust features** (SURF) [41].

Figure 41.7: Two overlapping images and some found correspondences using SURF descriptors. A random set of four correspondences are highlighted.

41.3.1Direct Linear Transform Algorithm

Given a set of point locations, and their correspondences across the two cameras, we can compute the homography relating the locations of imaged points within the two cameras. To estimate the homography we can use the direct linear transform (DLT) algorithm, as we did for the camera calibration problem in section 39.7. This will give us a linear set of equations for the parameters of the homography matrix with the form:

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{41.11}$$

where $\mathbf{A}$ is a $2N \times 9$ matrix, when given N corresponding pairs. The vector $\mathbf{h}$ contains all the elements of the matrix $\mathbf{H}$ stacked as a column vector of length 9. Note that $\mathbf{h}$ only has 8 degrees of freedom as the results do not change for a global scaling of all the values. Therefore we will need at least four correspondences to estimate the homography between two sets of corresponding points, as the ones shown in figure 41.7. As before, the solution is the vector that minimizes the residual $\mathbf{A}\mathbf{h}^2$. The result is given by the eigenvector of the matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ with the smallest eigenvalue.

In practice, to get accurate results, it is useful to also minimize the reprojection error after you initialize with the DLT solution since $\mathbf{A}\mathbf{h}^2$ is meaningless geometrically.

41.3.2Robust Model Fitting: Random Sampling with Consensus

To perform the image stitching in the section above, we had to find the homography parameters that best described the transformation from one image to another. Because the detection of the feature points can be noisy, the homography fitting needs to be robust against inevitable feature mismatches between images. A good algorithm to fit models to data robustly is RANSAC [128], which stands for **random sampling with concensus**.

> RANSAC, introduced in 1981 by Fischler and Bolles [128], has been a workhorse in computer vision ever since [532].

The procedure for RANSAC is simple, and the name contains most of the steps of the algorithm. First, randomly select a sufficient set of datapoints to fit the parameters of some model. The could be the parameters that define a line, some other structure, or a homography. Then, compute the model parameters from the randomly sampled set of points. Compute the inliers in the dataset, that is, the datapoints that fit the model (or achieve consensus) to within some tolerance. Repeat that procedure some number of times, N. Compute a final model from the set of inlier points corresponding to the largest number of inlier datapoints.

Figure 41.8 shows a simple instantiation of this algorithm for the case of robustly fitting a straight line model to a collection of datapoints (figure 41.8[a]). First, a number of datapoints, sufficient to uniquely specify the model parameters, are drawn from the dataset. For this problem, that is two points. Two randomly selected points are marked in green in figure 41.8(b). After finding the line that passes through those two points, the number of **inliers** (datapoints within of the model) are determined. For the line in figure 41.8(b), the number of inliers is three. This process is repeated until some desired number of samplings, S, have been drawn. The output model is fitted from all the inlier datapoints corresponding to the model that led to the most inlier datapoints.

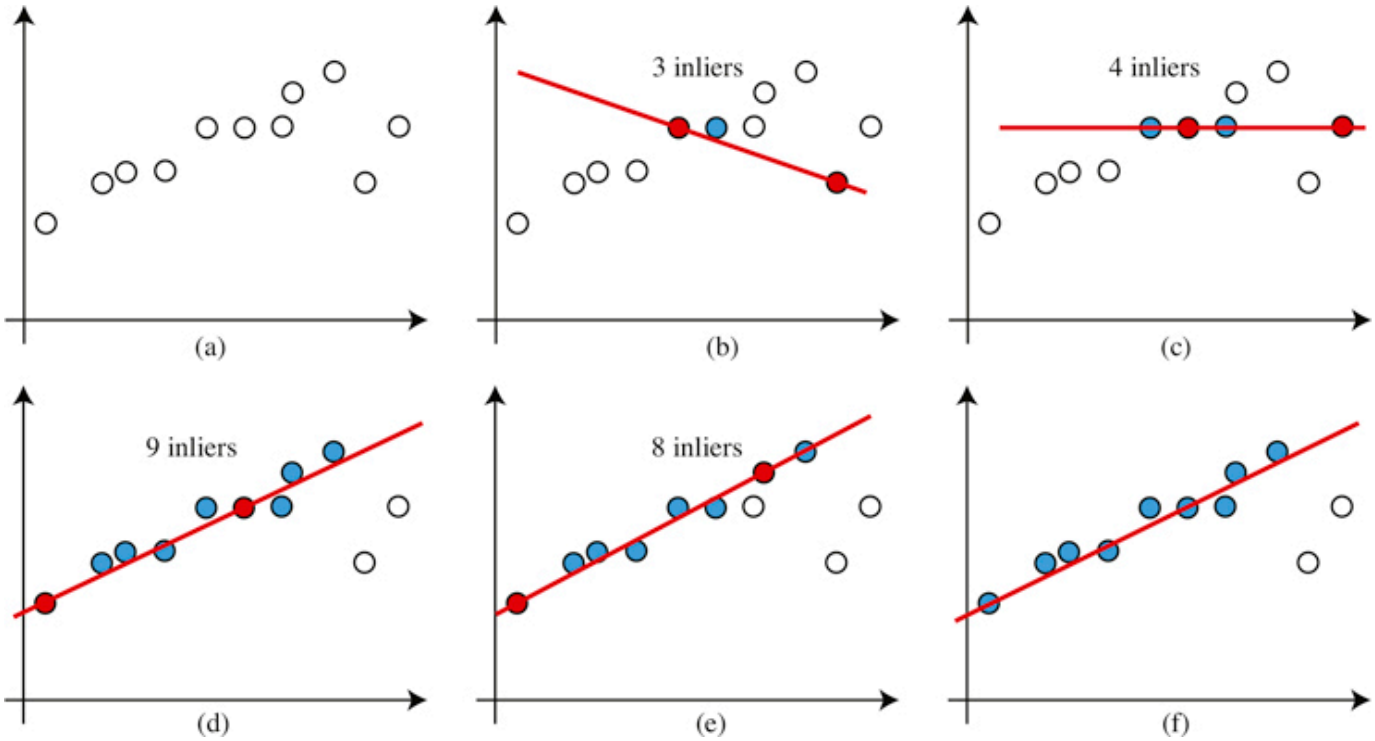


Figure 41.8: RANSAC applied to robust estimation of line parameters. (a) Datapoints for which robust line estimation is sought. (b) Two points (in blue) are selected at random, sufficient to estimate line model parameters. The number of inliers is computed for this model. (c - e) Repeat k times. (f) Finally, select the model with the maximum number of inliers.

The paper [128] derives a heuristic for the number of samples, S, required to be assured a good model fit. Let n be the number of datapoints required to specify the model, and let w be (an estimate of) the probability that any selected datapoint is within the error tolerance of the model. For figure 41.8(f), there are 2 outliers out of 11 points, so $w = 0.18$ (in general, this value needs to be estimated). The w^n is the probability that all the selected points are inliers, and $1 - w^n$ is the probability that at least one selected point is an outlier. The probability that in k trials only bad models are selected is then $(1 - w^n)^k$. If p is the probability of selecting the correct model, we have

$$1 - p = (1 - w^n)^k \tag{41.12}$$

Taking the logarithm of both sides lets us solve for k, the number of RANSAC iterations required to find a good fit, with probability p:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{41.13}$$

For the example of figure 41.8, for 95 percent accuracy, $k = \frac{\log(-05)}{\log(1-0.18^2)} = \frac{-1.3}{-0.0143} = 91$ So with 91 trials, we would have 95 percent confidence of finding the correct model through RANSAC. Note that in

practice you should select points without replacement to avoid degenerate fits, but the previous analysis assumed that points are selected with replacement. This is all legitimate when the number of points selected is small relative to the set of available points.

In the case of the homography estimation, we need eight equations to use the DLT algorithm to estimate **H**. As each point contributes two equations, we need four corresponding points. We can sample multiple randomly selected sets of four points and use RANSAC to estimate the homography between two images.

41.3.3Image Stitching

Using DLT and RANSAC with can compute the homography between each pair of images in figure and align them with respect to a reference image (in this example we pick the middle image as a reference). We can use the estimated homographies to warp all of the images into a single camera view as shown in figure 41.9.

Figure 41.9: Panoramic image composed by the three overlapping images from figure 41.3. Only the middle picture is unmodified. The other two are warped using the estimated homographies.

41.4Concluding Remarks

Homographies are an important class of geometric transforms between images and they have lots of applications. Homographies can be used to take measures on a plane given a reference measure. Although we have focused on computing homographies between images using correspondences, one can compute homographies in other ways. For instance, to produce a bird's-eye view of a plane one can compute the needed homography using the horizon line to get the camera rotation and, for an uncalibrated camera, we can use vanishing points to get the intrinsic camera parameters. The homography can also be extracted using a neural networks trained to regress the homography given two input images [2].

42Single View Metrology

42.1Introduction

The goal of this chapter is to introduce a set of techniques that allows recovering three-dimensional (3D) information about the scene from a single image like the picture of the office we used before (figure 42.1).

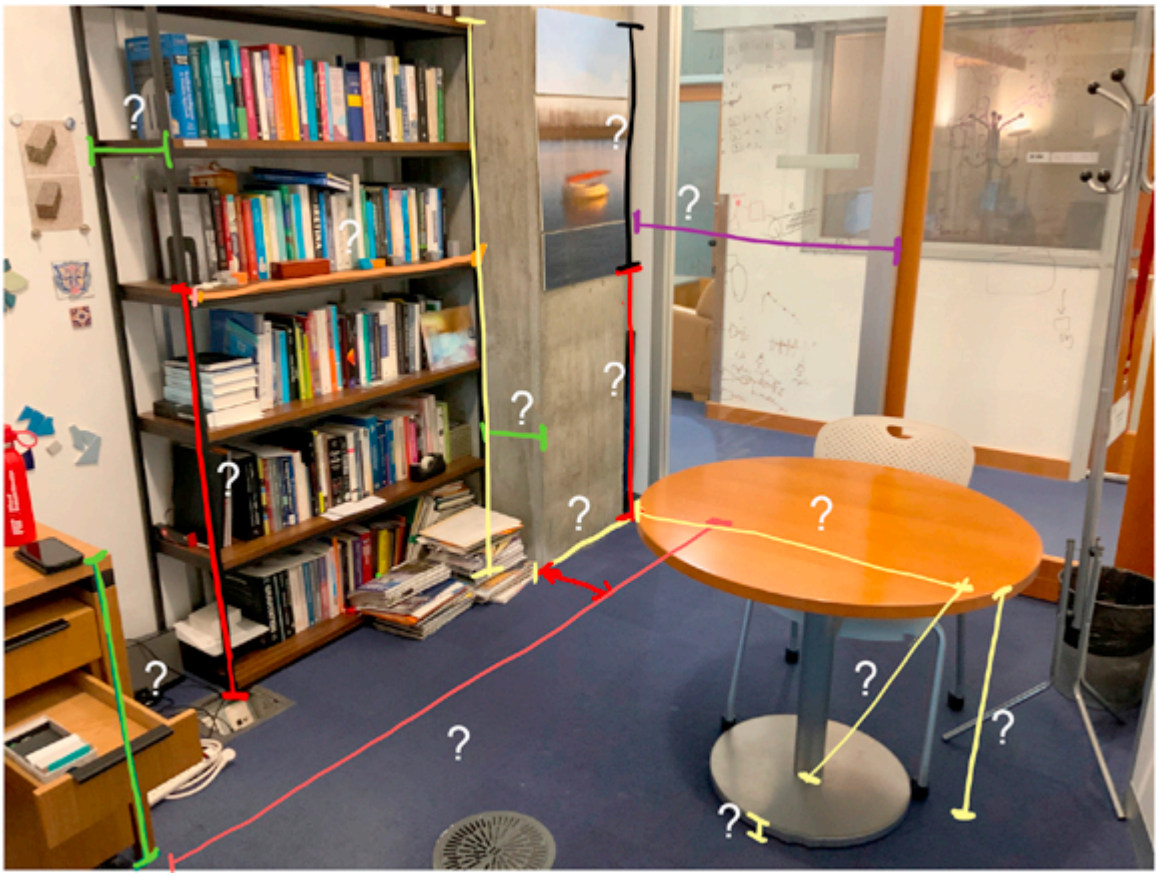


Figure 42.1: Is it possible to recover three-dimensional (3D) scene information from a single two-dimensional (2D) image? How tall is the desk? How wide is the bookshelf? What do we need to know about the scene in order to be able to answer those questions?

Single view metrology relies on understanding the geometry of the image formation process, the rules of perspective, and other visual cues present in the image to answer questions about the 3D structure of the scene. This is in contrast to other methods, like stereo vision, structure from motion or multiview geometry, which utilize multiple images to reconstruct 3D information. The reader should consult specialized monographs for a deeper analysis such as the book on multiview geometry by Hartley and Zisserman [187].

There are many scenarios where recovering 3D information from a single view is important. There

are applications from which only one view is available, such as when looking at paintings or TV, when analyzing photographs, and so on. Even when we have stereo vision, stereo is only effective when the disparity is large enough which limits the range of distances for which stereo can be used reliably.

We could just start by diving into learning-based methods and train a huge neural network to solve the task, but that would be no fun (we will do it later, no worries). Instead, we will start by using a model based single-view metrology. There will be no learning involved. Instead we will derive cues for 3D estimation from first principles.

42.2A Few Notes About Perception of Depth by Humans

The image in figure 42.2, recreated from Mezzanotte and Biederman [49], displays a flat set of lines that elicit a powerful 3D perception. Even more striking is that we perceive far more than mere 3D geometric figures (which is already intriguing, considering we are looking at a flat piece of paper). We discern buildings, people, a car in the street, the sky between the structures, and even what appears to be the entrance to a movie theater on the left sidewalk. Yet, the sidewalk is simply one line!

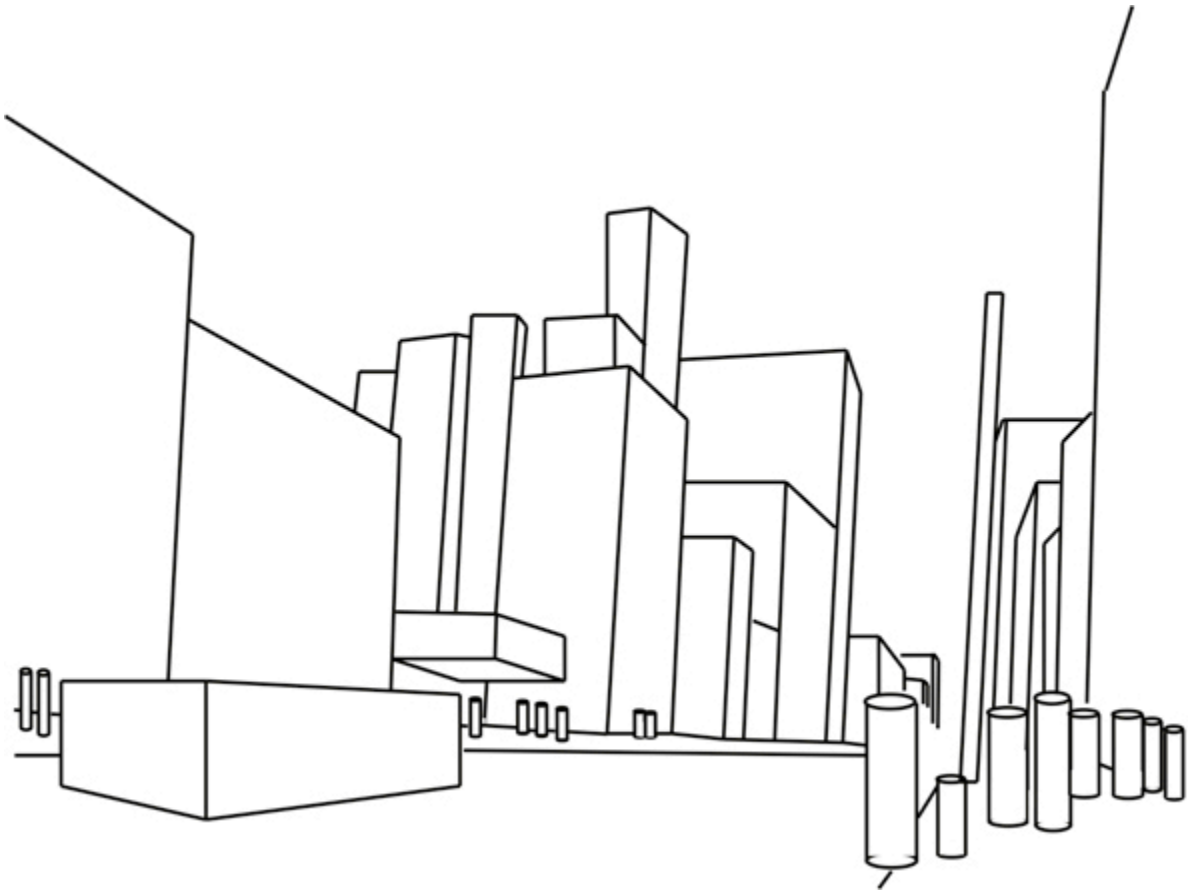


Figure 42.2: The image shows how scene understanding is possible from simple 3D cues and context. This sketch is a recreation of a real photograph shown in [49]. *Source*: Mezzanotte and Biederman.

We perceive everything we see in the sketch as 3D, even when we know that it's merely a 2D image. Our 3D processing is ingrained that we cannot easily switch off. That said, we are aware that it

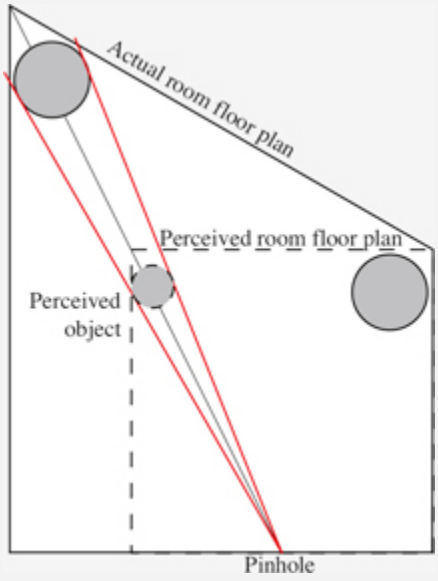
is just a collection of lines and geometric figures drawn on a flat piece of paper, but the 3D interpretation transforms the way in which we perceive the sketch. The influence of the 3D perceptual mechanisms is better appreciated when looking at visual illusions.

3D visual illusions are remarkably powerful. One captivating 3D illusion is the **Ames room**.

The Ames room was invented by Adelbert Ames, director of research at the Dartmouth Eye Institute, in 1946. Since then, illusions based on Ames' ideas have been utilized by comedians to create amusing effects.

The Ames room is an irregular room constructed in such a way that when you look at it with one eye from a particular vantage point, the room appears to be perfectly rectangular. This is achieved by making the part of the wall that is farther away from the viewer larger so that it appears fronto-parallel.

Geometric description of the Ames room visual illusion.



The best way to understand this illusion is by building it and experiencing it by yourself. You can find the instructions in figure 42.3. When you look thought the aperture on the wall indicated by "cut" the room will appear square and smaller than it actually is, and it will seem as if you are looking inside the room from the middle of it despite that the opening is closer to the right wall.

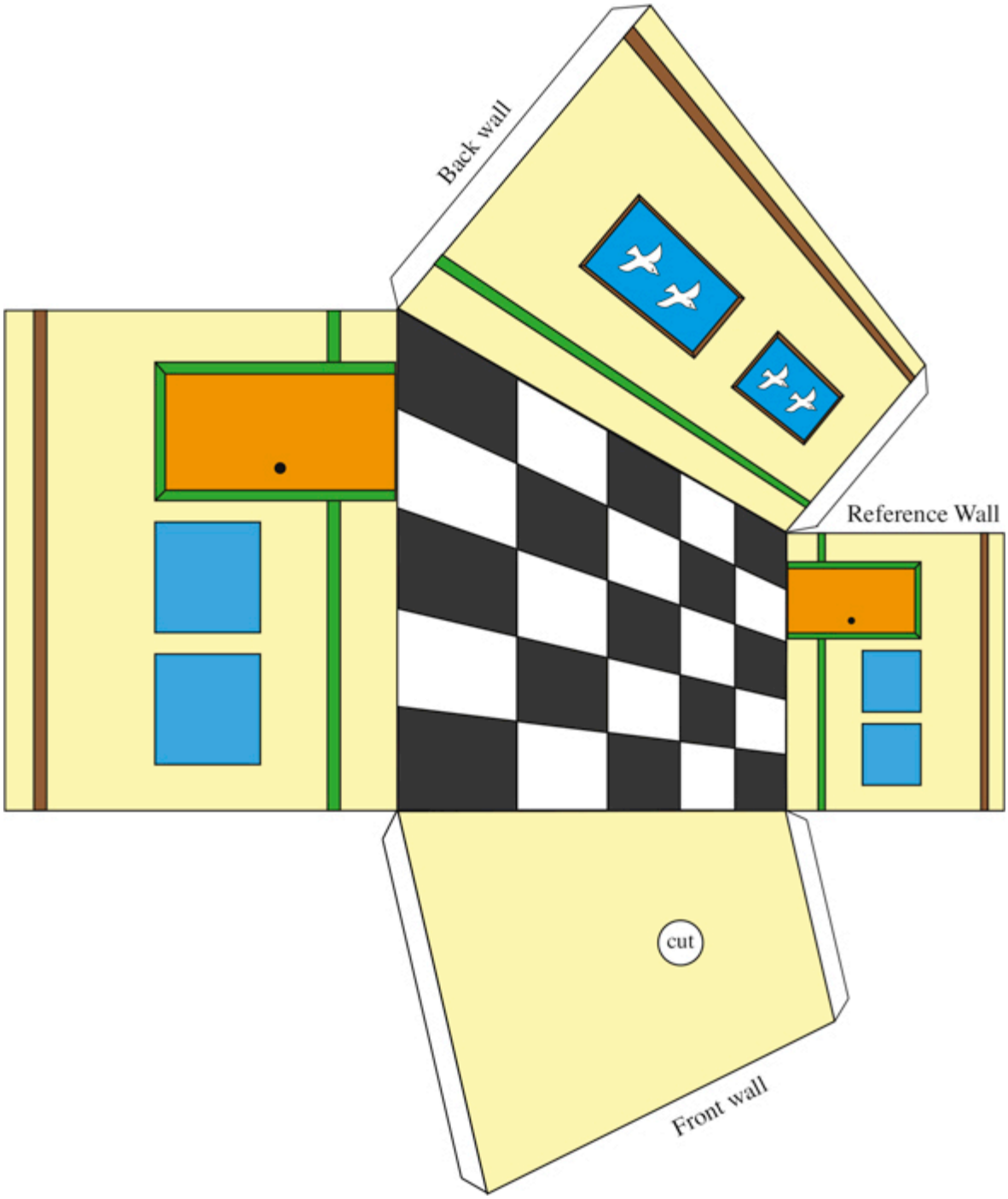


Figure 42.3: Make your own Ames room. Take a picture of this page and print it as large as you can (A4 or letter size will be sufficient). When you look through the aperture on the *front wall*, the room will appear square and smaller than it actually is. The shape of the *reference wall* is the only one that will appear unchanged, while all other planes will appear smaller than their actual size. You can change the room's content by drawing on the *reference wall* and then warping the image onto the other walls using homography. The floor pattern results from a homography with a square floor plan.

This illusion is especially remarkable when the room is large enough for people to walk in. You will notice that as people walk along the back wall, they appear to change in height. Your perception will disregard all of your (very strong) prior expectations about how that person should appear. Figure

42.4 illustrates this phenomenon with our cut-out version of the Ames room.

Figure 42.4: If you place an object along the back wall of the Ames room shown in figure 42.3, its perceived size will change. Also note that both doors appear to be similar en size despite being quite different in reality.

Clearly, a single image contains very strong cues about the 3D scene structure. In the following sections we will study linear perspective, arguably the most powerful among all pictorial cues used for 3D interpretation from a 2D image.

42.3Linear Perspective

Linear perspective uses parallel and orthogonal lines, vanishing points, the ground plane and the horizon line to infer the 3D scene structure from a 2D image.

> The discovery of linear perspective is attributed to the Italian architect Fillipo Brunelleschi in 1415.

As we will see in the following sections, linear perspective relies on a few assumptions about the scene and, as a consequence, linear perspective is not a general cue that is useful in all scenes, but it is very powerful in most human-made environments.

42.3.1Projection of 3D Lines

In perspective projection, 3D lines project into 2D lines in the camera plane. Let's start writing the parametric vector form of a 3D line using heterogeneous coordinates. We can parameterize a 3D line with a point, $\mathbf{P}_0 = [X_0, Y_0, Z_0]^\mathsf{T}$, and a direction vector, $\mathbf{D} = [D_X, D_Y, D_Z]^\mathsf{T}$, as:

$$\mathbf{P}_t = \begin{bmatrix} X_0 + tD_x \\ Y_0 + tD_y \\ Z_0 + tD_z \end{bmatrix} \tag{42.1}$$

The 3D point $\mathbf{P}_t$ moves along a line when the independent variable t goes from $-\infty$ to ∞. A camera located in the world coordinates origin will observe a 2D line resulting from projecting the 3D line into the camera plane as shown in the sketch shown in figure 42.5.

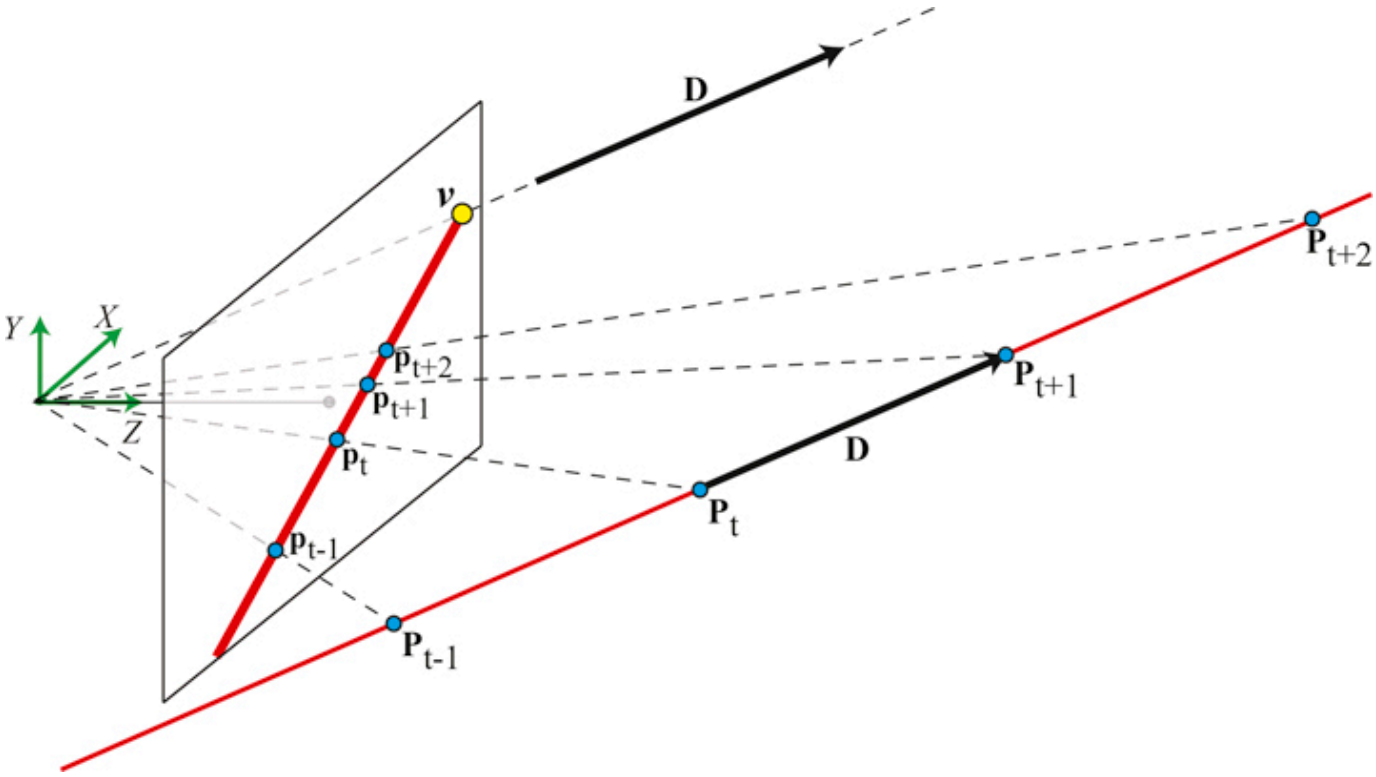


Figure 42.5: Projection of a 3D line onto the camera plane. The 3D line and the camera center form a plane (the camera center is a the world-coordinates origin). The intersection of this plane with the camera plane is the 2D line that appears in the image. The image point **v** is the vanishing point of the line.

We can get the equation of the 2D line that appears in the image by using the perspective projection equations, resulting in:

$$\mathbf{p}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} f\dfrac{X_0 + tD_X}{Z_0 + tD_Z} \\ f\dfrac{Y_0 + tD_Y}{Z_0 + tD_Z} \end{bmatrix} \tag{42.2}$$

> One subtle point about what is visible by the camera: Remember that when we plot the image plane we are using the virtual camera plane, but the real "image" is being formed behind the pinhole. So, in the example shown in figure 42.5, the points along the line with positive Z-value might be visible even when they are behind the virtual camera plane.

42.3.2Vanishing Points

Even though a line in 3D space extends infinitely, its 2D projection doesn't. According to equation (42.2), in the limit when t goes to infinity, the line converges to a finite point called the vanishing point:

$$\mathbf{v} = \lim_{t \to \infty} \mathbf{p}_t = \begin{bmatrix} f\frac{D_X}{D_Z} \\ f\frac{D_Y}{D_Z} \end{bmatrix} \tag{42.3}$$

The vanishing point, $\mathbf{v} = [v_x, v_y]^\mathsf{T}$, only depends on the direction vector, **D**, thus all parallel lines in 3D project into non-parallel 2D images that converge to the same vanishing point in the image. If $D_Z = 0$ then the 3D line is parallel to the camera plane and the vanishing point is in infinite.

Figure 42.6 shows a set of parallel 3D lines and their 2D image projections. Each group of parallel lines converges to a different vanishing point. Lines that are parallel to the camera plane remain parallel as they converge to a vanishing point in infinity.

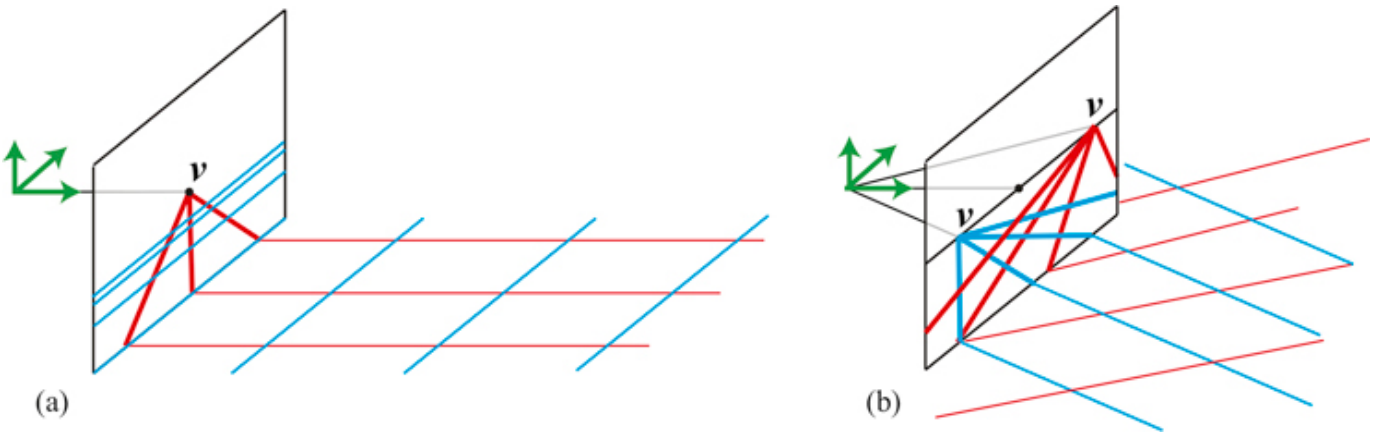


Figure 42.6: Parallel 3D lines converge to the same vanishing point. (a) The red lines are parallel to the optical axis and converge to a vanishing point in the center. (b) Two sets of parallel lines contained in an horizontal plane, parallel to the optical axis. Their vanishing points are on an horizontal image line.

All the sets of parallel lines inside a plane converge to a set of co-linear vanishing points. These points lie along the plane's horizon line.

Vanishing points are very useful to recover the missing depth of the scene as we will see later. The human visual system also makes use of vanishing points for 3D perception. Take, for example, the *leaning tower visual illusion* [258] as depicted in figure 42.7. Here, a pair of identical images of the Leaning Tower of Pisa are placed side by side. Interestingly, despite being identical, they create an illusion of leaning with different angles. The image on the right appears to have a greater tilt than the one on the left. One explanation of this illusion is that the lines of each tower converge at a different vanishing point. If both towers were parallel in 3D, all the lines would converge to the same 2D vanishing point. As they do not, our brain interprets that the towers should have different 3D

orientations. This process is automatic and we can not shut it down despite the fact that we know both images are identical.

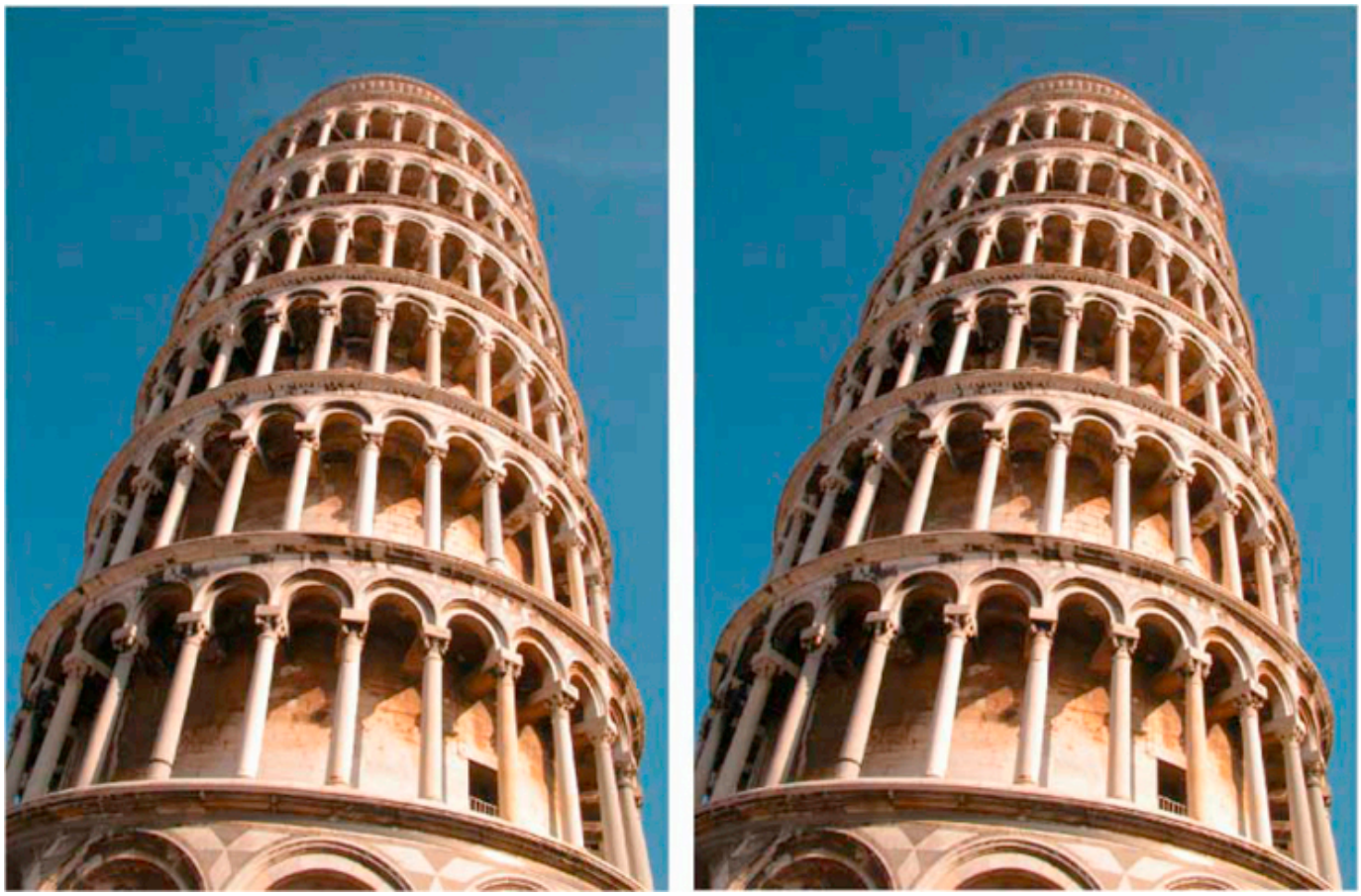


Figure 42.7: The *leaning tower illusion* [258] involves two identical pictures placed side by side. The picture on the right appears to be more tilted than the one on the left.

Homogeneous coordinates offer an alternative way of working with vanishing points. Let's start by writing the parametric vector form of a 3D line using homogeneous coordinates:

$$\mathbf{P}_t = \begin{bmatrix} X_0 + tD_X \\ Y_0 + tD_Y \\ Z_0 + tD_Z \\ 1 \end{bmatrix} = \begin{bmatrix} X_0/t + D_X \\ Y_0/t + D_Y \\ Z_0/t + D_Z \\ 1/t \end{bmatrix} \tag{42.4}$$

The first term is equivalent to the writing of the line in equation (42.1). As homogeneous coordinates are invariant to a global scaling, we can divide the vector by t which results in the second equality.

In the limit $t \rightarrow \infty$ the fourth coordinate of equation (42.4) converges resulting in:

$$\mathbf{P}_\infty = \begin{bmatrix} D_X \\ D_Y \\ D_Z \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix} \tag{42.5}$$

Note that here we are representing a point at infinity with finite homogeneous coordinates. This is another of the advantages of homogeneous coordinates.

> Homogeneous coordinates allow us to represent points at infinity with finite coordinates by setting the last element to 0.

The vanishing point, in homogeneous coordinates, is the result of projecting $\mathbf{P}_\infty$ using the camera model, M:

$$\mathbf{v} = \mathbf{M}\mathbf{P}_\infty = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{T}\end{bmatrix}\begin{bmatrix}\mathbf{D}\\ 0\end{bmatrix} = \mathbf{K}\mathbf{R}\mathbf{D} \tag{42.6}$$

The last equality is obtained by using equation (39.17), and it shows that the vanishing point coordinates do not depend on camera translation. Only a camera rotation will change the location of the vanishing points in an image. If a scene is captured by two identical cameras translated from each other, their corresponding images will have the same vanishing points.

Both equation (42.3) and equation (42.6) are equivalent if we replace **K** by the appropriate perspective camera model.

If instead of perspective projection, we have parallel projection, then all the vanishing points will project at infinity in the image plane.

42.3.3Horizon Line

Any 3D plane becomes a half-plane in the image and has an horizon line. The horizon line is the line that passes by all the vanishing points of all the lines contained in the plane as shown in figure 42.8.

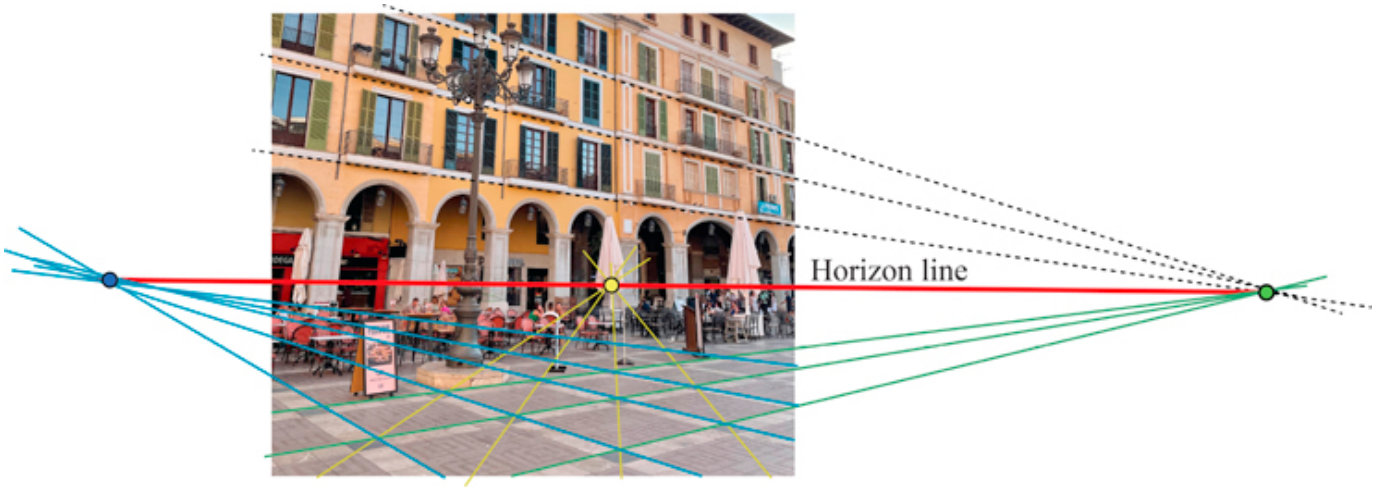


Figure 42.8: All parallel lines contained in a plane converge to a set of vanishing points. All those vanishing points are contained in the **horizon line** of the plane. Parallel planes have the same horizon line.

As parallel lines converge to the same vanishing point, parallel planes also have the same horizon line. In figure 42.8, the lines in the building facade that are parallel to the ground also converge to a vanishing point that lies on the horizon line. In this particular examples, the lines in the facade also happen to be parallel to one of the axis of the floor tiles, converging to the same vanishing point on the right side of the figure.

The ground plane is a special plane because it supports most objects. This is different than a wall. Both are planes, but the ground has an ecological importance that a wall does not have. The point of contact between objects and the ground plane informs about its distance. The distance between the contact point and the horizon line is related to distance between the object and the camera. Objects further away have a contact point projecting onto a higher point in the image plane.

Figure 42.9 illustrates how to detect the horizon line using homogeneous coordinates. If lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, in homogeneous coordinates, correspond to the 2D projection of 3D parallel lines, we can obtain the vanishing point as their intersection (cross-product, $\boldsymbol{l}_1 \times \boldsymbol{l}_2$). If we now have another set of parallel lines, contained in the same plane, $\boldsymbol{l}_3$ and $\boldsymbol{l}_4$, we can get another vanishing point by computing their cross-product. Finally, the horizon line is the line that passes through both points that can be computed as the cross-product of the vanishing points in homogeneous coordinates.

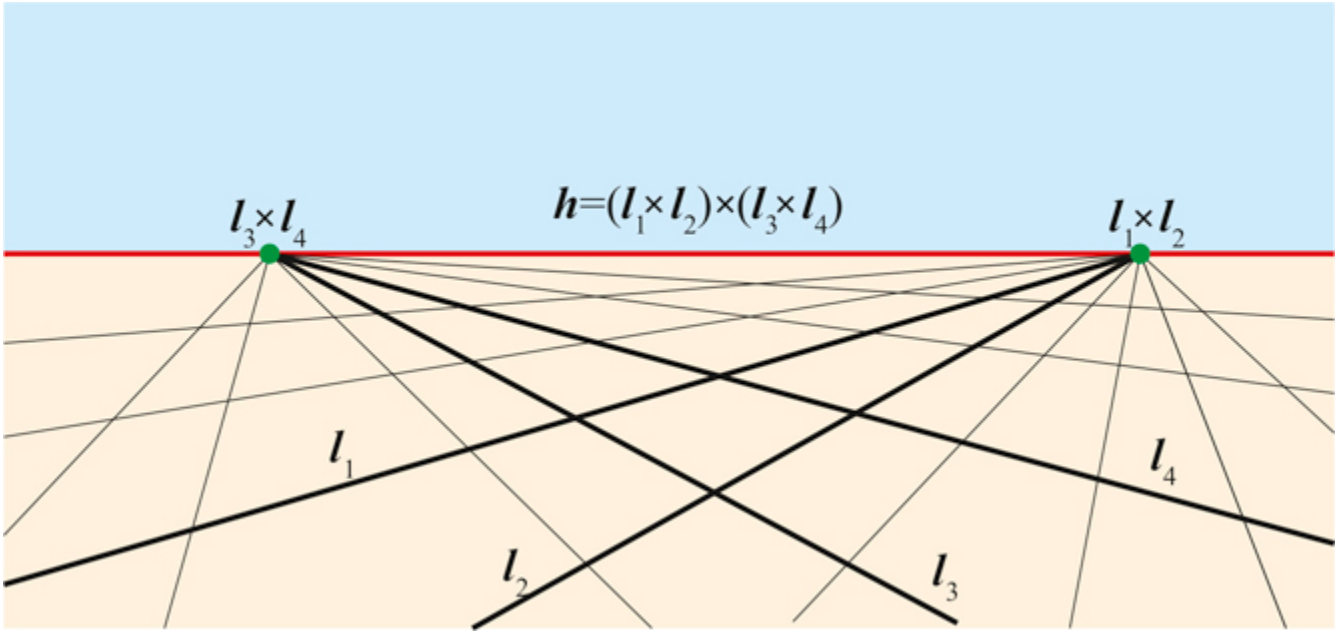


Figure 42.9: Using parallel lines, in homogeneous coordinates, to compute vanishing points and the horizon line.

42.3.4Detecting Vanishing Points

Figure 42.10 shows the office scene and three vanishing points computed manually. In this scene there are three dominant orientations along orthogonal directions. The three vanishing points are estimated by selecting pairs of long edges in the image corresponding to parallel 3D lines in the scene and finding their intersection.

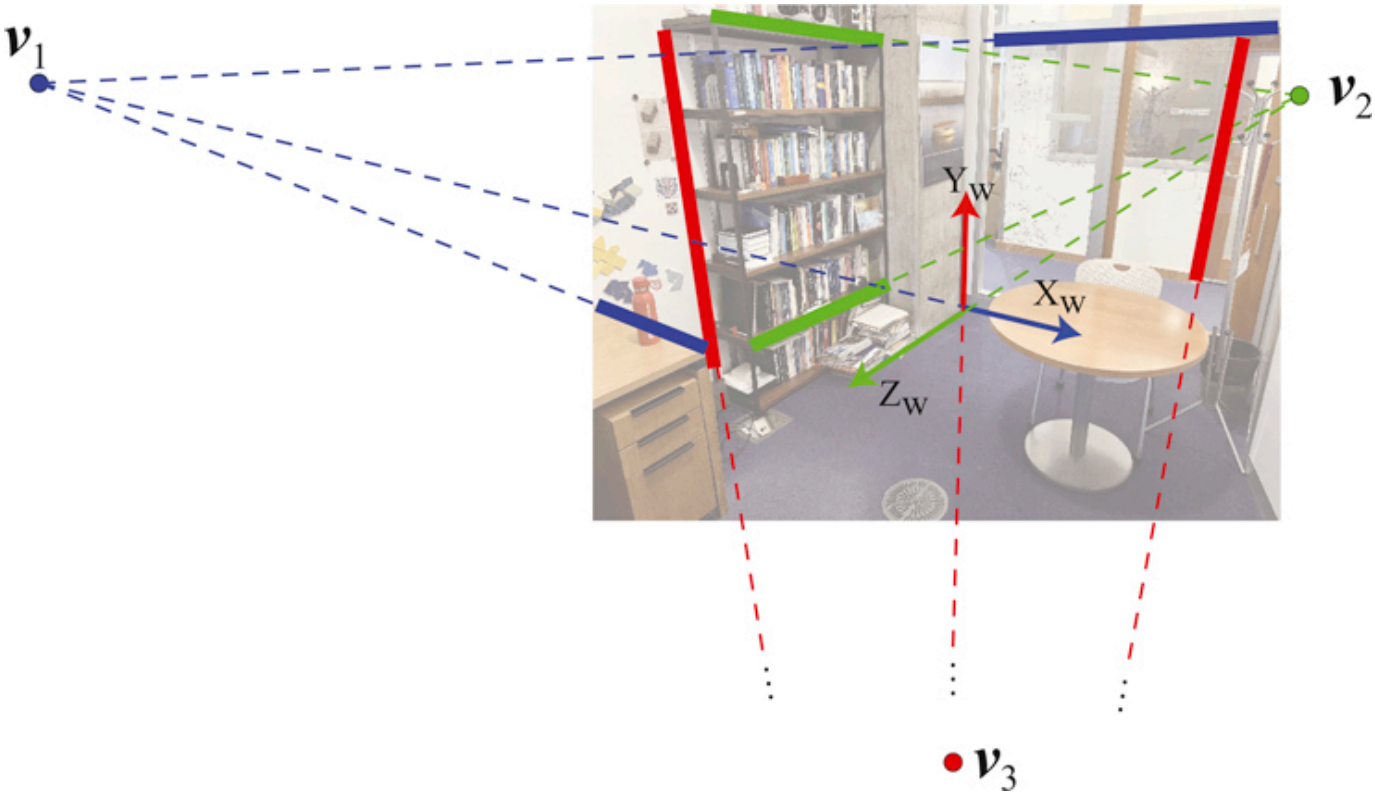


Figure 42.10: Most object boundaries in this photograph are oriented parallel to one of the world-coordinates reference frame (the world-coordinates frame is chosen in this way on purpose). All the parallel 3D lines converge on the image plane in a vanishing point. In this picture there are three main vanishing points, each of them related to one of the world-coordinates frame.

Vanishing points can be extracted automatically from images by first detecting image edges and their orientations (like we did in chapter 2) and then using RANSAC (section 41.3.2) to group the edges: every pair of edges will vote for a vanishing point and the point with the largest support will be considered a vanishing point.

There are also learning-based approaches for vanishing point detection using graphical models [84], or deep learning based approaches such as DeepVP [74] and NeurVPS [530].

42.4Measuring Heights Using Parallel Lines

Let's now use what we have seen up to now to measurement distances in the world from a single image. We will discuss in this section how to make those measurements using three vanishing points, or just the horizon line and one vanishing point. We will not use camera calibration or the projection matrix. For a deeper analysis we refer the reader to the wonderful thesis of A. Criminisi [85].

Before we can make any 3D measurements we need to study an important invariant of perspective projection: the cross-ratio.

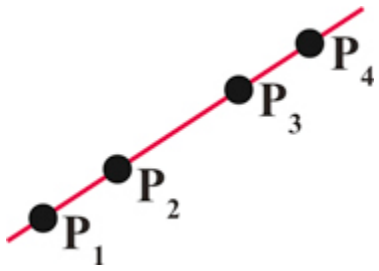
42.4.1Cross-Ratio

As we have discussed before, perspective projection transforms many aspects of the 3D world when projecting into the 2D image: angles between lines are not preserved, the relative lengths between lines is not preserved, parallel 3D lines do not project into 2D parallel lines in the image (unless they are also parallel to the camera plane), and so on. Fortunately, some geometric properties are preserved under perspective projection. For instance, straight lines remain straight. Are there other quantities that are preserved? The cross-ratio is one of those quantities.

The cross-ratio of four collinear points is defined by the expression:

$$CR(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4) = \frac{|\mathbf{P}_3 - \mathbf{P}_1|\,|\mathbf{P}_4 - \mathbf{P}_2|}{|\mathbf{P}_3 - \mathbf{P}_2|\,|\mathbf{P}_4 - \mathbf{P}_1|} \tag{42.7}$$

where $|\mathbf{P}_i - \mathbf{P}_j|$ is the euclidean distance between points $\mathbf{P}_i$ and $\mathbf{P}_j$, using heterogeneous coordinates.



The cross-ratio is considered the most important projective invariant. Two sets of points **P** and **Q** related by a projective transformation, as shown in figure 42.11, have the same cross-ratio (regardless of the choice of origin **O** or scale factor). That is:

$$CR(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4) = CR(\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3, \mathbf{Q}_4) \tag{42.8}$$

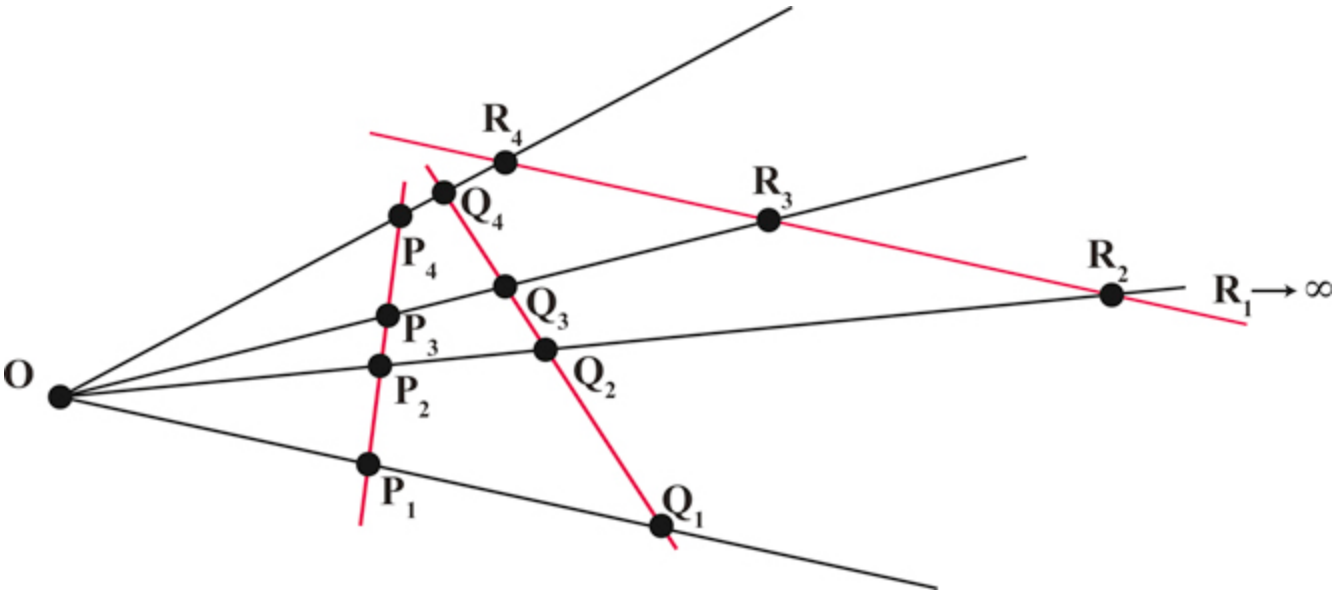


Figure 42.11: The set of points **P** are related to the points **Q** (and **R**) by the cross-ratio. The point $\mathbf{R}_1$ is in infinity because the line connecting the points **R** is parallel to the line passing by **O**, $\mathbf{P}_1$, and $\mathbf{Q}_1$.

The order in which these points are considered matters in the calculation. There are 4! = 24 possible orderings resulting in six different values. If two sets of points, related by a projection, have the same ordering, then their cross ratio will be preserved. We do not provide the proof for the cross-ratio invariant here.

When one of the points is at infinity, all the terms that contain that point are cancelled out (this can also be seen by taking a limit). In the case of the points **R** from figure 42.11 we get that cross-ratio is:

$$CR(\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3, \mathbf{R}_4) = \frac{|\mathbf{R}_4 - \mathbf{R}_2|}{|\mathbf{R}_3 - \mathbf{R}_2|} \tag{42.9}$$

This value is equal to the cross-ratio of the points **P** and **Q**.

To gain some intuition about the cross-ratio invariant, let's look at an empirical demonstration. Figure 42.12 shows a ruler seen under different view points. The blue dots correspond to same ruler locations with P_1 = 0 in, P_2 = 6 in, P_3 = 8 in, and P_4 = 10 in (we are making these measurements in inches, but note that the units do not affect the cross-ratio as it is scale invariant). Using those measurements, the cross-ratio is: (8 × 4)/(2 × 10) = 1.6. The same cross-ratio is obtained if we measure the positions of the four points using image coordinates and measuring distances in the image plane.

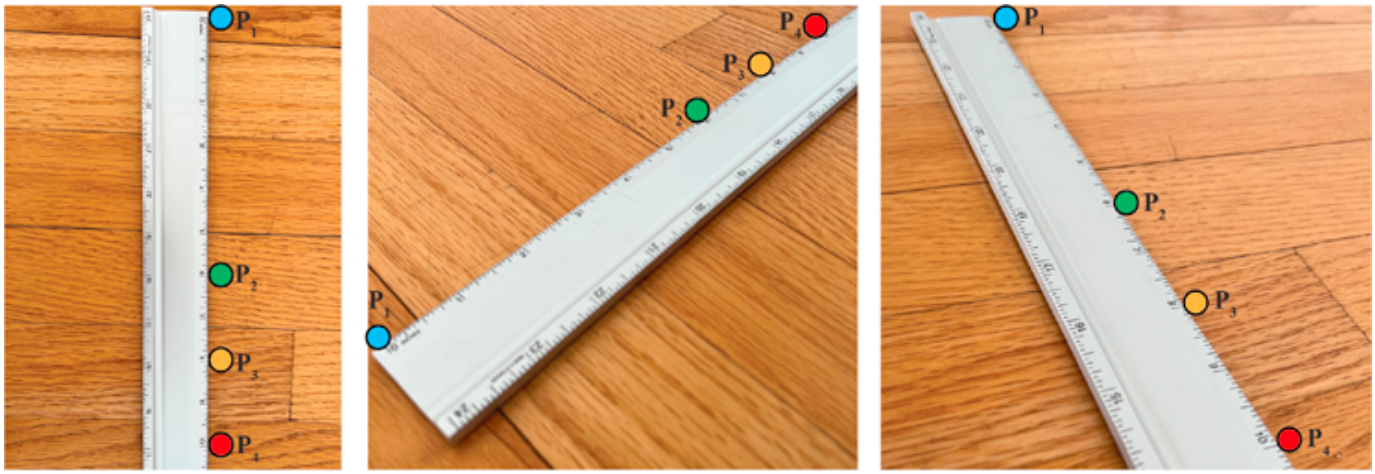


Figure 42.12: Empirical demonstration of the cross-ratio invariant. You can check that measuring distances between the three dots on each image and computing the cross-ratio results in the value 1.6 for the three images.

42.4.2 Measuring the Height of an Object

The problem we will address in this section is how to measure the height of one object given the height of a reference object in the scene. As shown in figure 42.13, the height of the bookshelf is $h_{\text{bookshelf}} = 197$ cm (in this example we will make our measurements using centimeters, but again, the choice of units is not important as long as all the measurements are consistent). Can we use this information together with the geometry of the scene to estimate the height of the desk, h_{desk}?

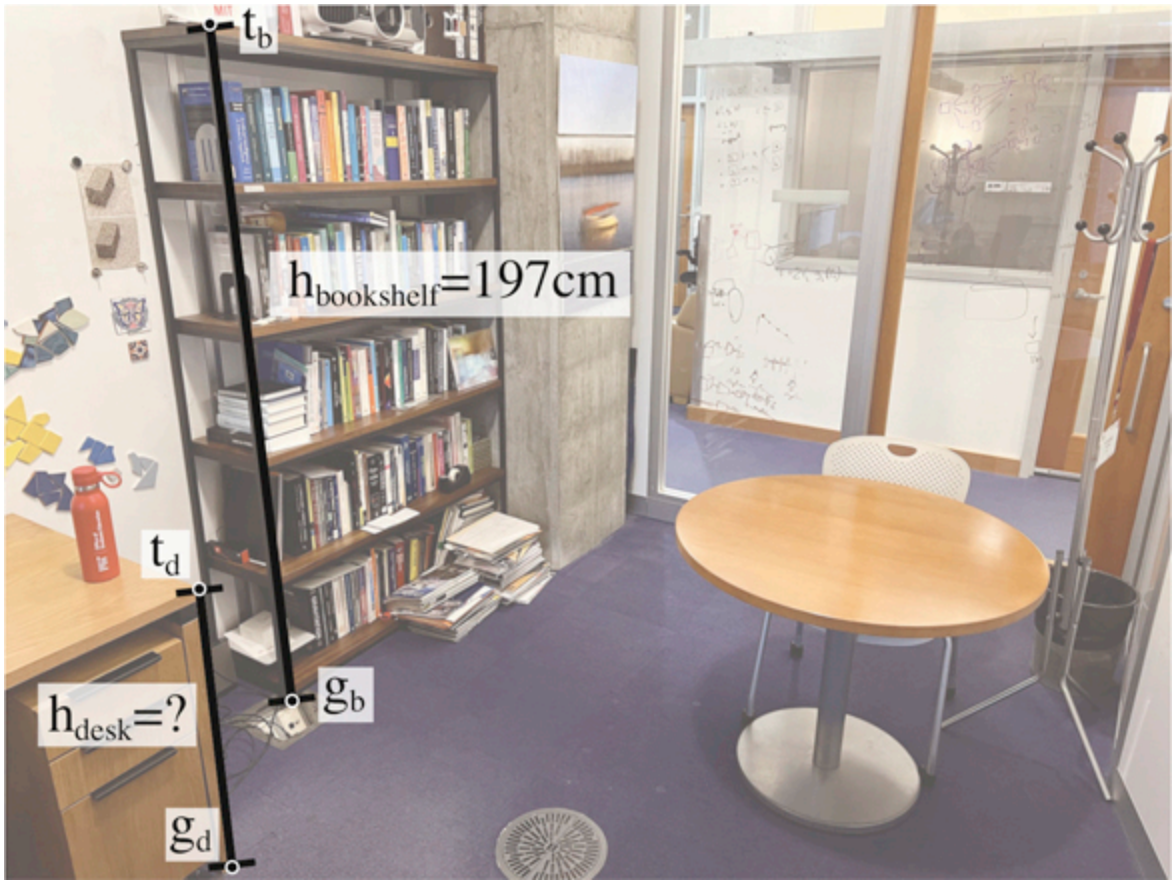


Figure 42.13: Given the height of a reference object, the location of the horizon line and one vanishing point, how can we estimate the height of other objects in the scene?

Due to perspective projection, we can not directly compare the relative height of the desk with respect to the height of the bookshelf using distances measured over the image. In order to be able to compare 3D distances using 2D image distances both objects need to be put in correspondence first. The bookshelf height is the distance between the 3D points that correspond to the 2D locations of the bottom of the bookshelf, $\mathbf{g}_b$, and the top of the bookshelf, $\mathbf{t}_b$. Similarly, the desk height is the distance between the 3D points that correspond to $\mathbf{g}_d$ and $\mathbf{t}_d$. We can not use directly the points $\mathbf{g}_b$ and $\mathbf{t}_b$, and $\mathbf{g}_d$ and $\mathbf{t}_d$ as they are given in image coordinates. In order to be able to use distances in the image domain we need to first translate the points defining the desk height on top of the bookshelf, and then use the cross-ratio invariant to relate the ratio between image distances with the ratios of 3D heights.

In the rest, all the vectors refer to image coordinates in homogeneous coordinates. Let's start by projecting the desk height onto the bookshelf. To do this we need to follow several steps:

- Estimate the line that passes by the ground points $\mathbf{g}_d$ and $\mathbf{g}_b$. Using homogeneous coordinates, the line can be computed using the cross-product:
$$\boldsymbol{l}_1 = \mathbf{g}_d \times \mathbf{g}_b \tag{42.10}$$
- Compute the intersection of the line $\boldsymbol{l}_1$ with the horizon line $\mathbf{h}$ as:
$$\mathbf{a} = \boldsymbol{l}_1 \times \mathbf{h} \tag{42.11}$$

As the line $\boldsymbol{l}_1$ is on the ground plane, the point **a** is the vanishing point of line $\boldsymbol{l}_1$.

- Compute the line that passes by **a** and the top of the desk $\mathbf{t}_d$ as:

$$\boldsymbol{l}_2 = \mathbf{a} \times \mathbf{t}_d \tag{42.12}$$

Lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ are parallel on 3D and vertically aligned.

- Finally we can compute where line $\boldsymbol{l}_2$ intersects with the bookshelf in the image plane which in this case will also correspond to an intersection of the corresponding 3D lines. We first compute the line that connects the bottom and top points of the bookshelf $\boldsymbol{l}_3 = \mathbf{g}_b \times \mathbf{t}_b$, and now we obtain the intersection with $\boldsymbol{l}_2$ as:

$$\mathbf{b} = \boldsymbol{l}_2 \times \boldsymbol{l}_3 \tag{42.13}$$

We can write all the previous steps into a single equation:

$$\mathbf{b} = (((\mathbf{g}_d \times \mathbf{g}_b) \times \mathbf{h}) \times \mathbf{t}_d) \times (\mathbf{g}_b \times \mathbf{t}_b) \tag{42.14}$$

Point **b** is the location where the top of the desk will be if we physically rearrange the furniture to bring the desk and the bookshelf into contact. The segment between points **b** and $\mathbf{g}_b$ is the projection of the desk height on the bookshelf. However, due to the foreshortening effect induced by perspective projection, we cannot simply rely on the ratio of image distances to estimate the desk's true height. Instead, we will utilize the cross-ratio invariant to make this estimation.

> As J. J. Gibson pointed out, the ground plane is very useful for all kinds of 3D scene measurements.

In order to use the cross-ratio invariant we need two sets of four colinear points related by a projective geometry. Figure 42.15 shows a sketch of the setup we have in figure 42.14.

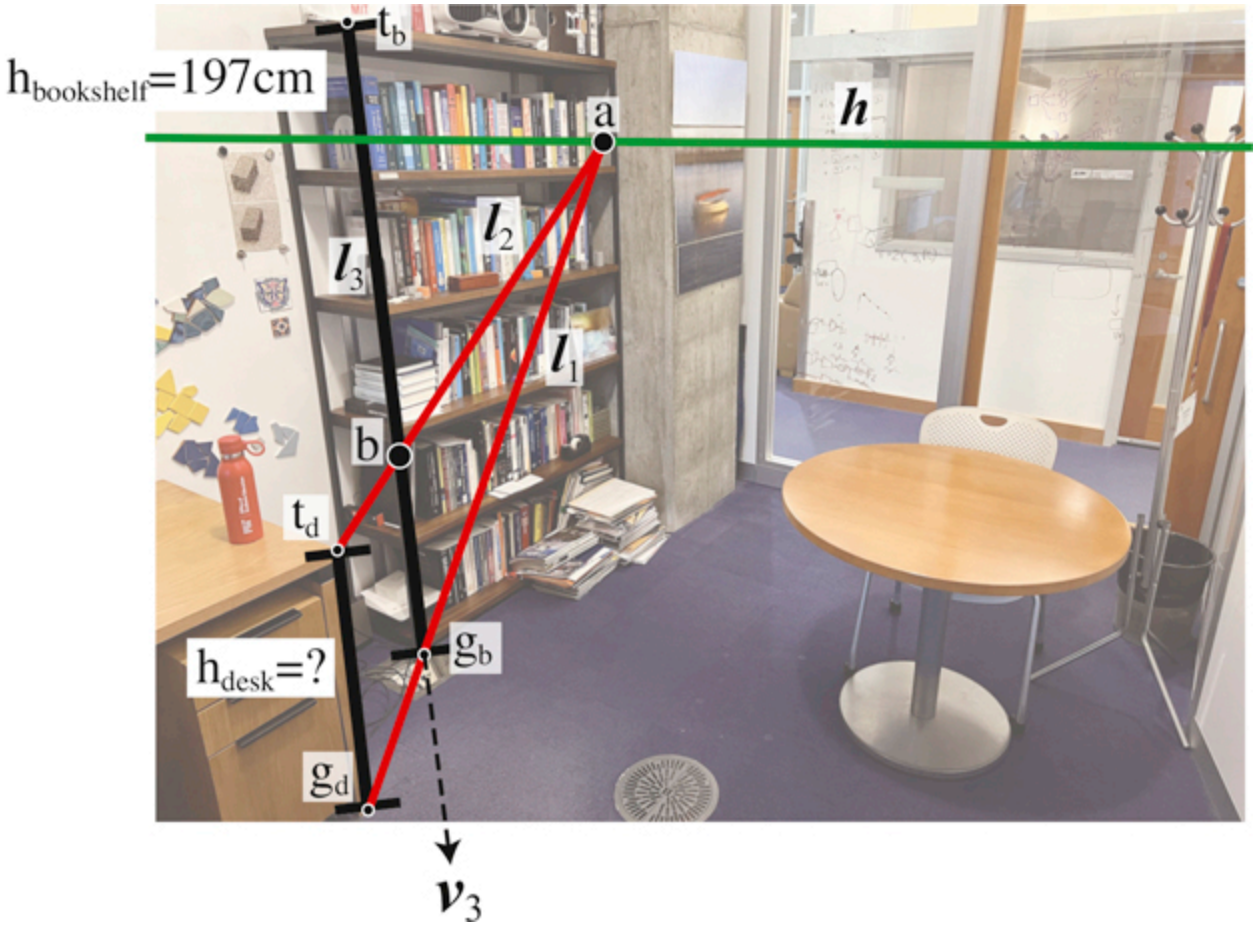


Figure 42.14: Projection of desk height onto the bookshelf. We start with the bottom and top image coordinates of points along vertical edges of the bookshelf: $\mathbf{t}_b = [702\ 2{,}958]^\mathsf{T}$ $\mathbf{g}_b = [987\ 618]^\mathsf{T}$ and the desk: $\mathbf{t}_d = [679\ 1{,}018]^\mathsf{T}$ $\mathbf{g}_d = [793\ 59]^\mathsf{T}$ We compute the points $\mathbf{a} = [4{,}471\ 2{,}486]^\mathsf{T}$ and $\mathbf{b} = [2{,}237\ 2{,}765]^\mathsf{T}$ as described in the text.

In the image plane, we have the four image points $\mathbf{t}_b$, $\mathbf{b}$, $\mathbf{g}_b$ and $\mathbf{v}_3$. Those four 2D image points correspond to the 3D locations given by the top of the bookshelf, desk height, ground and infinity (which is the location of the vertical vanishing point in 3D).

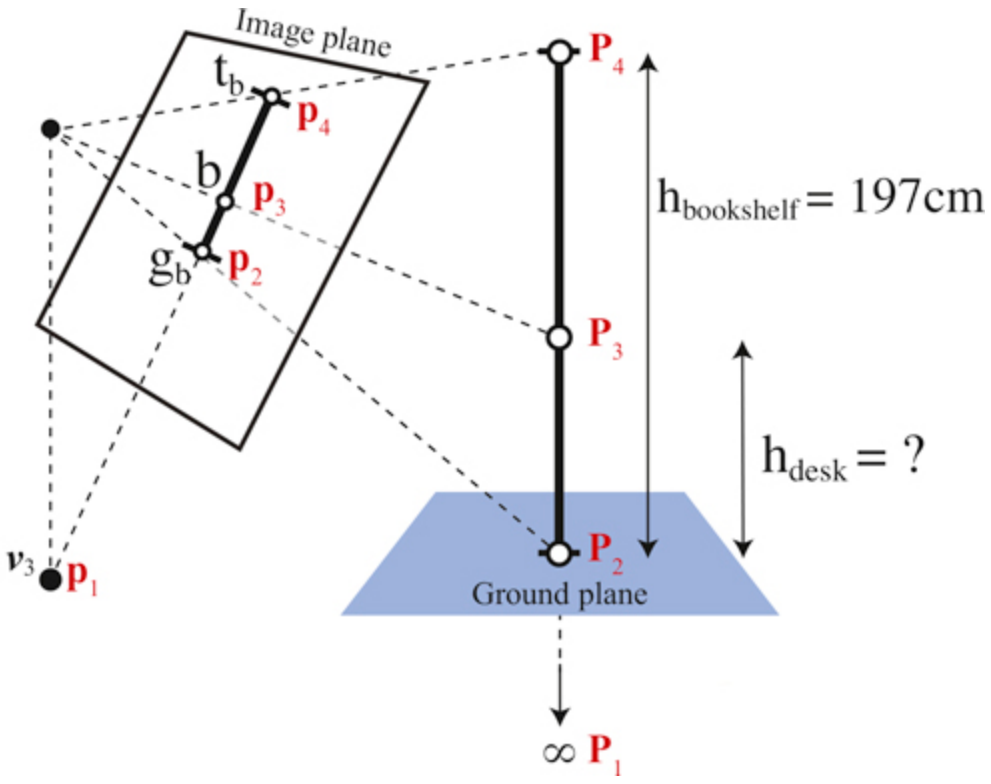


Figure 42.15: Sketch of the setting from figure 42.14. There are two sets of colinear points related by central projection. Their cross-ratios are identical.

The rest of calculations requires computing distances between points, which only works when using heterogeneous coordinates (or homogeneous coordinates if the last component is equal to 1 for all points). The cross-ratio invariant between both sets of four points gives as the equality:

$$CR(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4) = CR(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) \tag{42.15}$$

The left side of the equality corresponds to the distances between the 3D points and the right side to the distances of the image points measured in pixel units. As $\mathbf{P}_1$ is in infinity (as it corresponds to the vanishing point), the left side ratio only has two terms, resulting in the ratio between the bookshelf and desk heights. To compute the hand side we need the coordinates of the corresponding four image points. Those are provided in figure 42.14. Replacing all those values in equation (42.15) we obtain:

$$\frac{h_{\text{bookshelf}}}{h_{\text{desk}}} = \frac{\left|\mathbf{b} - \mathbf{v}_3\right| \left|\mathbf{t}_b - \mathbf{g}_b\right|}{\left|\mathbf{b} - \mathbf{g}_b\right| \left|\mathbf{t}_b - \mathbf{v}_3\right|} \tag{42.16}$$

The only unknown value in this equality is the height of the desk, h_{desk}, which results in h_{desk} 73.1 cm, which is close to the actual height of the desk (which is around 76 cm).

Projective invariants are measures that do not change after perspective projection. The cross-ratio is the most important projective invariant.

42.4.3Height Propagation to Supported Objects

The method described in the previous section relies on using vertical lines in contact with the ground plane to propagate information from one point in space to another in order to make measurements. The method can not work if we can not establish the vertical projection of a point into the ground plane (or any other plane of reference). But once we have estimated the height of objects that are in contact with the ground, we can use them to propagate 3D information to other objects not directly on top of the ground. We can estimate the height of objects that are not in contact with the ground if they are on top of objects of known height. As an example, let's estimate the red bottle's height, which we can do after we estimated the height of the desk.

Figure 42.16 shows how to estimate the height of the bottle.

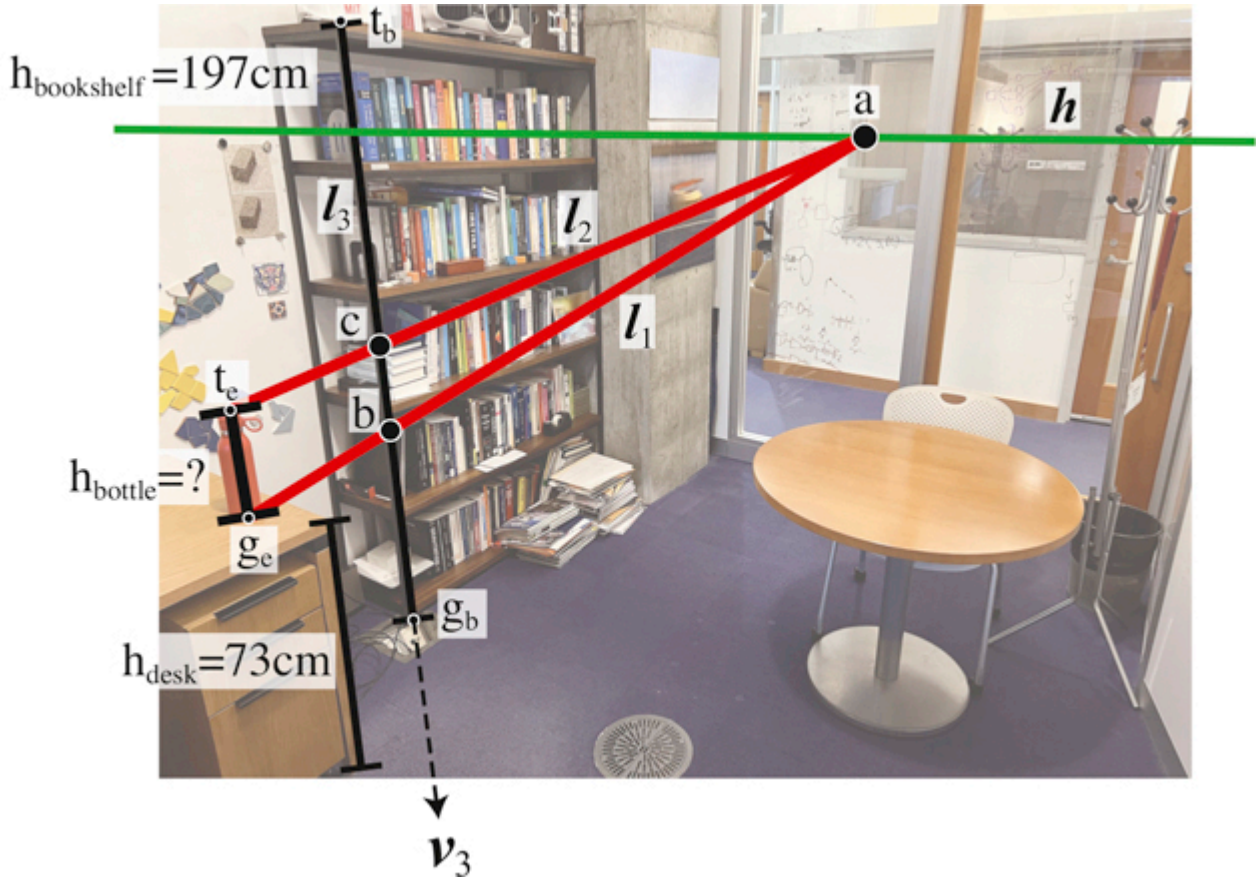


Figure 42.16: Estimating the height of the red bottle. For an object not in the ground we can propagate information from the objects that support it.

As before, we will project the bottle height on the reference line on the corner of the bookshelf. To do this we need to find two vertically aligned parallel 3D lines that connect the bottle's top and bottom points with the bookshelf. The first line, $\boldsymbol{l}_1$, is the line connecting the bottom of the bottle, $\mathbf{g}_e$, and the point **b**. These two points are at the same height from the ground plane as **b** corresponds to the top of the desk and $\mathbf{g}_e$ is the point of the bottle that is in contact with the top of the desk. The intersection of the line $\boldsymbol{l}_1$ with the horizon line **h** give us point **a**. Connecting point **a** with the top of the bottle, point $\mathbf{t}_e$

, gives us the second line, $\boldsymbol{l}_2$. The intersection of this line with the reference bookshelf line is the point **c**. The distance between the point **c** and the ground in 3D is the sum $h_{\text{bookshelf}} + h_{\text{desk}}$. Writing everything into a compact equation results in:

$$\mathbf{c} = (((\mathbf{g}_e \times \mathbf{b}) \times \mathbf{h}) \times \mathbf{t}_e) \times (\mathbf{g}_b \times \mathbf{t}_b) \tag{42.17}$$

Note that this equation relies on having computed **b** first.

Using the cross-ratio invariant we arrive to the following equation:

$$h_{\text{bottle}} = h_{\text{bookshelf}} \frac{|\mathbf{c} - \mathbf{g}_b| \, |\mathbf{t}_b - \mathbf{v}_3|}{|\mathbf{c} - \mathbf{v}_3| \, |\mathbf{t}_b - \mathbf{g}_b|} - h_{\text{desk}} \tag{42.18}$$

This last equation shows how the height of the bottle is estimated by propagating information from the ground via its supporting object, the desk. Learning based approaches that estimated 3D from single images will have to perform such propagation implicitly. The result that we get is h_{bottle} 27.6 cm which is close to the real height of 25.5 cm.

This procedure highlights the importance of the correct parsing of the **supported-by hierarchy** between objects in the scene.

> A **scene graph** is a representation of a scene using a graph where the nodes correspond to objects, and the edges encode the relationship between them. One important relationship is **supported-by**.

42.53D Metrology from a Single View

In the previous section we showed how to use a reference object to measure other objects. Let's now discuss a more general framework to locate 3D points [85].

The office picture from figure 42.17 shows the projection of the 3D coordinates frame into the image plane. The axes directions are aligned with the three dominant orthogonal orientations present in the scene and are aligned with the image vanishing points. We will show how can we use the world-coordinates in order to extract 3D object locations from a single image.

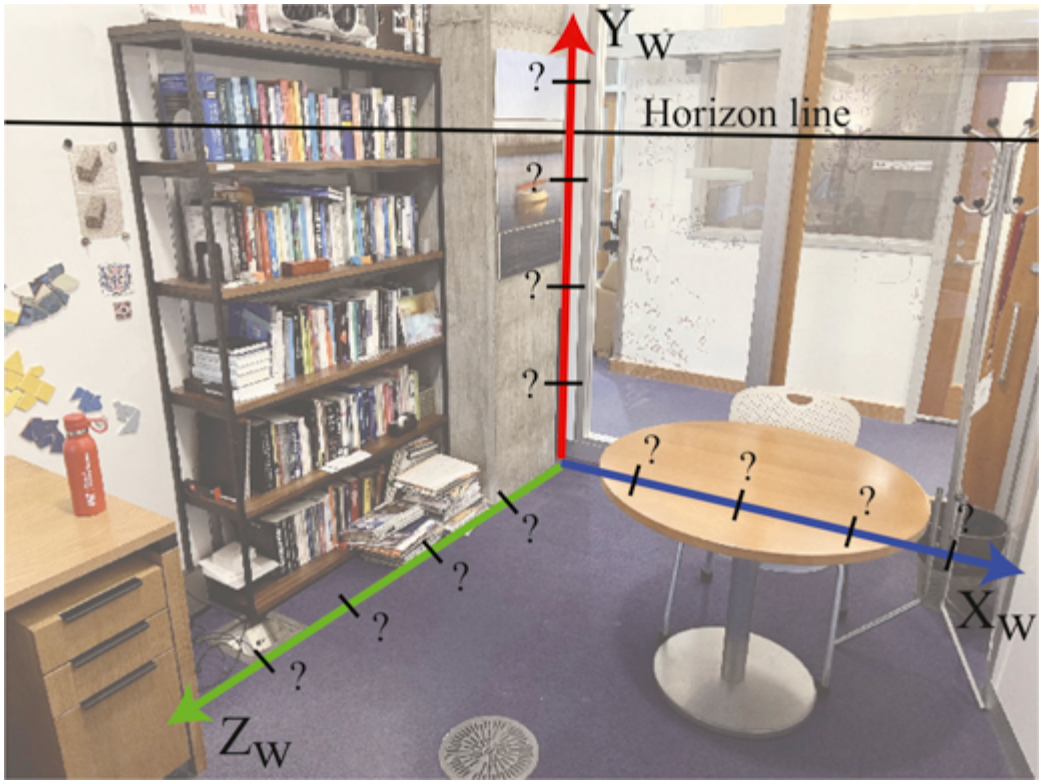


Figure 42.17: Can we find out where to put the ticks in the world-coordinate axes as seen by the camera so that they correspond to calibrated measurements? Simply placing them at evenly spaced intervals along the three axes in the image won't be correct due to perspective projection.

In this section we will describe first how to calibrate the projected world-coordinate axes in the image plane and then we will show how to transport points into the 3D world axes in order to measure object sizes and points locations.

42.5.1 Calibration of the Projected World Axis

Where should we position the tick marks that correspond to 1, 2, 3, … meters from the origin on each axis in figure 42.17? Simply placing them at evenly spaced intervals along the axis in the image won't be correct. Due to geometric distortion introduced by perspective projection, the tick marks are not evenly spaced within the image plane. This distortion affects each axis in a distinct manner.

Let's start assuming we know the location of some arbitrary 3D distance to the origin on each axis that we will denote by α_X, α_Y and α_Z. The question is, where do we place the ticks for $k\alpha_X$, $k\alpha_Y$ and $k\alpha_Z$ for all k? We will use the cross-ratio to calibrate the projection of the world coordinate system in the image plane. For doing this we will need the location of the horizon line of a reference plane (i.e. the ground plane), the position of the orthogonal vanishing point and the real measure of one object in the scene.

As shown in figure 42.18(a) we can use the cross-ratio to estimate the location of the points $Y = k\alpha_X$ where α_X is an arbitrary constant as shown in figure 42.18(b), and the corresponding image location r

, for $k = 1$, is chosen arbitrarily as shown in figure 42.18(a). Due to perspective projection, the image projection of the point $Y = k\alpha_X$ will not be evenly spaced in the image. In this particular example we chose the orientation of the world axis in image coordinates to be aligned with the location of the vanishing points.

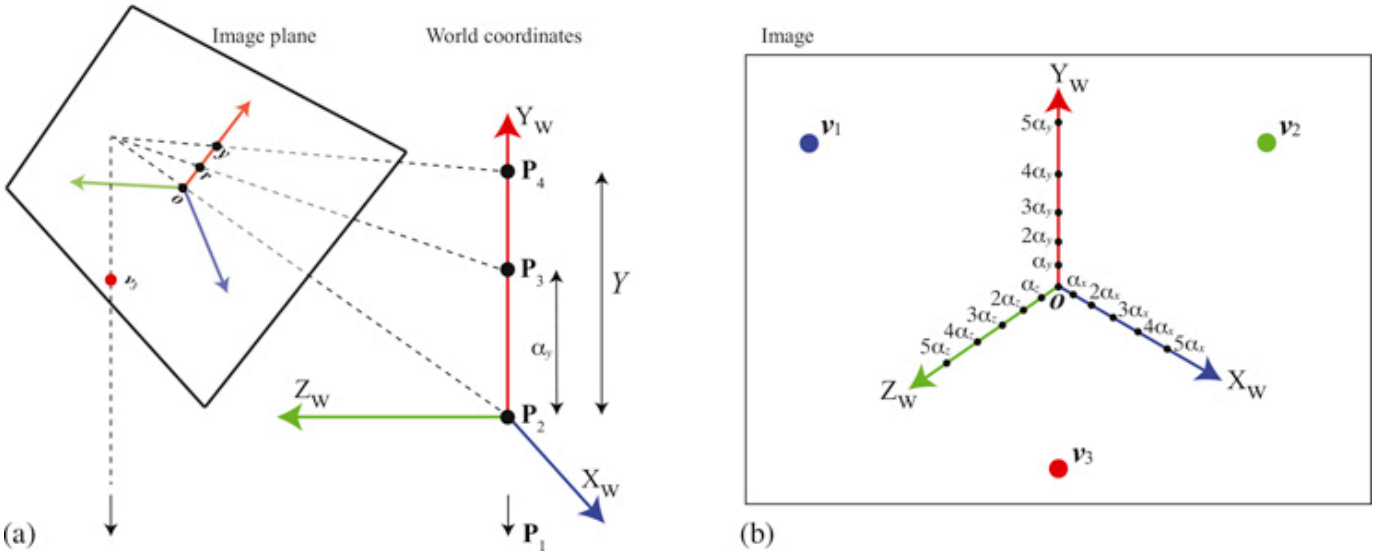


Figure 42.18: Using the cross-ratio invariant to project the world coordinates into the image using only the vanishing points.

In order to calibrate the axis we need to know the dimensions of an object in the scene. We need measurements along the three axis. In this example, we used the bookshelf height (197 cm) and the radius of the table base (50 cm). Using these reference measurements we can solve for the image locations of the ticks $\alpha_X = \alpha_Y = \alpha_Z = 1$ m.

Figure 42.19 shows the final calibrated axes with ticks placed each 50 cm along each of the three axes.