24 PERCEPTION

In which we connect the computer to the raw, unwashed world.

PERCEPTION
SENSOR

Perception provides agents with information about the world they inhabit by interpreting the response of **sensors**. A sensor measures some aspect of the environment in a form that can be used as input by an agent program. The sensor could be as simple as a switch, which gives one bit telling whether it is on or off, or as complex as the eye. A variety of sensory modalities are available to artificial agents. Those they share with humans include vision, hearing, and touch. Modalities that are not available to the unaided human include radio, infrared, GPS, and wireless signals. Some robots do **active sensing**, meaning they send out a signal, such as radar or ultrasound, and sense the reflection of this signal off of the environment. Rather than trying to cover all of these, this chapter will cover one modality in depth: vision.

We saw in our description of POMDPs (Section 17.4, page 658) that a model-based decision-theoretic agent in a partially observable environment has a **sensor model**—a probability distribution $\mathbf{P}(E \mid S)$ over the evidence that its sensors provide, given a state of the world. Bayes' rule can then be used to update the estimation of the state.

OBJECT MODEL
RENDERING MODEL

For vision, the sensor model can be broken into two components: An **object model** describes the objects that inhabit the visual world—people, buildings, trees, cars, etc. The object model could include a precise 3D geometric model taken from a computer-aided design (CAD) system, or it could be vague constraints, such as the fact that human eyes are usually 5 to 7 cm apart. A **rendering model** describes the physical, geometric, and statistical processes that produce the stimulus from the world. Rendering models are quite accurate, but they are ambiguous. For example, a white object under low light may appear as the same color as a black object under intense light. A small nearby object may look the same as a large distant object. Without additional evidence, we cannot tell if the image that fills the frame is a toy Godzilla or a real monster.

Ambiguity can be managed with prior knowledge—we know Godzilla is not real, so the image must be a toy—or by selectively choosing to ignore the ambiguity. For example, the vision system for an autonomous car may not be able to interpret objects that are far in the distance, but the agent can choose to ignore the problem, because it is unlikely to crash into an object that is miles away.

A decision-theoretic agent is not the only architecture that can make use of vision sensors. For example, fruit flies (*Drosophila*) are in part reflex agents: they have cervical giant fibers that form a direct pathway from their visual system to the wing muscles that initiate an escape response—an immediate reaction, without deliberation. Flies and many other flying animals make used of a closed-loop control architecture to land on an object. The visual system extracts an estimate of the distance to the object, and the control system adjusts the wing muscles accordingly, allowing very fast changes of direction, with no need for a detailed model of the object.

Compared to the data from other sensors (such as the single bit that tells the vacuum robot that it has bumped into a wall), visual observations are extraordinarily rich, both in the detail they can reveal and in the sheer amount of data they produce. A video camera for robotic applications might produce a million 24-bit pixels at 60 Hz; a rate of 10 GB per minute. The problem for a vision-capable agent then is: *Which aspects of the rich visual stimulus should be considered to help the agent make good action choices, and which aspects should be ignored?* Vision—and all perception—serves to further the agent's goals, not as an end to itself.

FEATURE EXTRACTION

RECOGNITION

RECONSTRUCTION

We can characterize three broad approaches to the problem. The **feature extraction** approach, as exhibited by *Drosophila*, emphasizes simple computations applied directly to the sensor observations. In the **recognition** approach an agent draws distinctions among the objects it encounters based on visual and other information. Recognition could mean labeling each image with a yes or no as to whether it contains food that we should forage, or contains Grandma's face. Finally, in the **reconstruction** approach an agent builds a geometric model of the world from an image or a set of images.

The last thirty years of research have produced powerful tools and methods for addressing these approaches. Understanding these methods requires an understanding of the processes by which images are formed. Therefore, we now cover the physical and statistical phenomena that occur in the production of an image.

24.1 IMAGE FORMATION

Imaging distorts the appearance of objects. For example, a picture taken looking down a long straight set of railway tracks will suggest that the rails converge and meet. As another example, if you hold your hand in front of your eye, you can block out the moon, which is not smaller than your hand. As you move your hand back and forth or tilt it, your hand will seem to shrink and grow *in the image*, but it is not doing so in reality (Figure 24.1). Models of these effects are essential for both recognition and reconstruction.

24.1.1 Images without lenses: The pinhole camera

SCENE

IMAGE

Image sensors gather light scattered from objects in a **scene** and create a two-dimensional **image**. In the eye, the image is formed on the retina, which consists of two types of cells: about 100 million rods, which are sensitive to light at a wide range of wavelengths, and 5

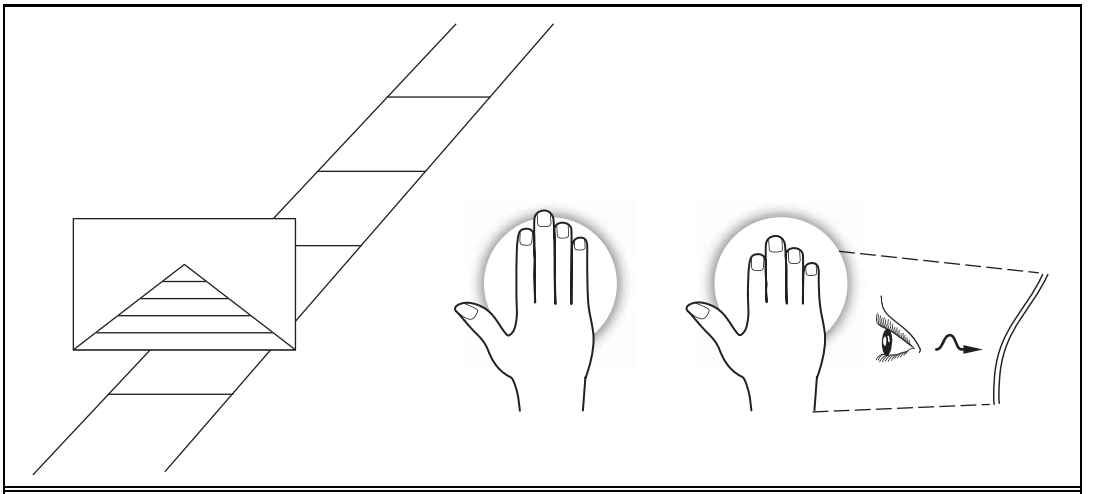


Figure 24.1 Imaging distorts geometry. Parallel lines appear to meet in the distance, as in the image of the railway tracks on the left. In the center, a small hand blocks out most of a large moon. On the right is a foreshortening effect: the hand is tilted away from the eye, making it appear shorter than in the center figure.

million cones. Cones, which are essential for color vision, are of three main types, each of which is sensitive to a different set of wavelengths. In cameras, the image is formed on an image plane, which can be a piece of film coated with silver halides or a rectangular grid of a few million photosensitive **pixels**, each a complementary metal-oxide semiconductor (CMOS) or charge-coupled device (CCD). Each photon arriving at the sensor produces an effect, whose strength depends on the wavelength of the photon. The output of the sensor is the sum of all effects due to photons observed in some time window, meaning that image sensors report a weighted average of the intensity of light arriving at the sensor.

PIXEL

To see a focused image, we must ensure that all the photons from approximately the same spot in the scene arrive at approximately the same point in the image plane. The simplest way to form a focused image is to view stationary objects with a **pinhole camera**, which consists of a pinhole opening, O, at the front of a box, and an image plane at the back of the box (Figure 24.2). Photons from the scene must pass through the pinhole, so if it is small enough then nearby photons in the scene will be nearby in the image plane, and the image will be in focus.

PINHOLE CAMERA

The geometry of scene and image is easiest to understand with the pinhole camera. We use a three-dimensional coordinate system with the origin at the pinhole, and consider a point P in the scene, with coordinates (X, Y, Z). P gets projected to the point P' in the image plane with coordinates (x, y, z). If f is the distance from the pinhole to the image plane, then by similar triangles, we can derive the following equations:

$$\frac{-x}{f} = \frac{X}{Z}, \ \frac{-y}{f} = \frac{Y}{Z} \quad \Rightarrow \quad x = \frac{-fX}{Z}, \ y = \frac{-fY}{Z} \ .$$

PERSPECTIVE PROJECTION

These equations define an image-formation process known as **perspective projection**. Note that the Z in the denominator means that the farther away an object is, the smaller its image

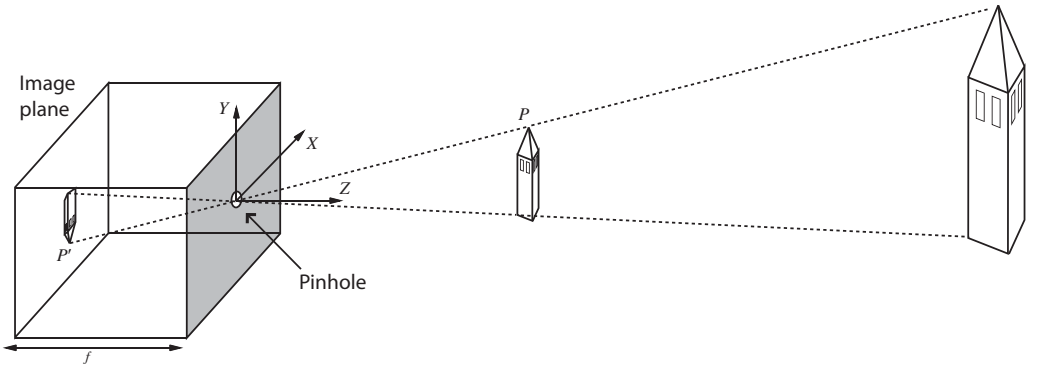


Figure 24.2 Each light-sensitive element in the image plane at the back of a pinhole camera receives light from a the small range of directions that passes through the pinhole. If the pinhole is small enough, the result is a focused image at the back of the pinhole. The process of projection means that large, distant objects look the same as smaller, nearby objects. Note that the image is projected upside down.

will be. Also, note that the minus signs mean that the image is *inverted*, both left–right and up–down, compared with the scene.

Under perspective projection, distant objects look small. This is what allows you to cover the moon with your hand (Figure 24.1). An important result of this effect is that parallel lines converge to a point on the horizon. (Think of railway tracks, Figure 24.1.) A line in the scene in the direction (U, V, W) and passing through the point (X_0, Y_0, Z_0) can be described as the set of points $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$, with λ varying between $-\infty$ and $+\infty$. Different choices of (X_0, Y_0, Z_0) yield different lines parallel to one another. The projection of a point P_λ from this line onto the image plane is given by

$$\left(f\frac{X_0 + \lambda U}{Z_0 + \lambda W}, f\frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right) .$$

As $\lambda \to \infty$ or $\lambda \to -\infty$, this becomes $p_\infty = (fU/W, fV/W)$ if $W \neq 0$. This means that two parallel lines leaving different points in space will converge in the image—for large λ, the image points are nearly the same, whatever the value of (X_0, Y_0, Z_0) (again, think railway tracks, Figure 24.1). We call p_∞ the **vanishing point** associated with the family of straight lines with direction (U, V, W). Lines with the same direction share the same vanishing point.

VANISHING POINT

24.1.2 Lens systems

The drawback of the pinhole camera is that we need a small pinhole to keep the image in focus. But the smaller the pinhole, the fewer photons get through, meaning the image will be dark. We can gather more photons by keeping the pinhole open longer, but then we will get **motion blur**—objects in the scene that move will appear blurred because they send photons to multiple locations on the image plane. If we can't keep the pinhole open longer, we can try to make it bigger. More light will enter, but light from a small patch of object in the scene will now be spread over a patch on the image plane, causing a blurred image.

MOTION BLUR

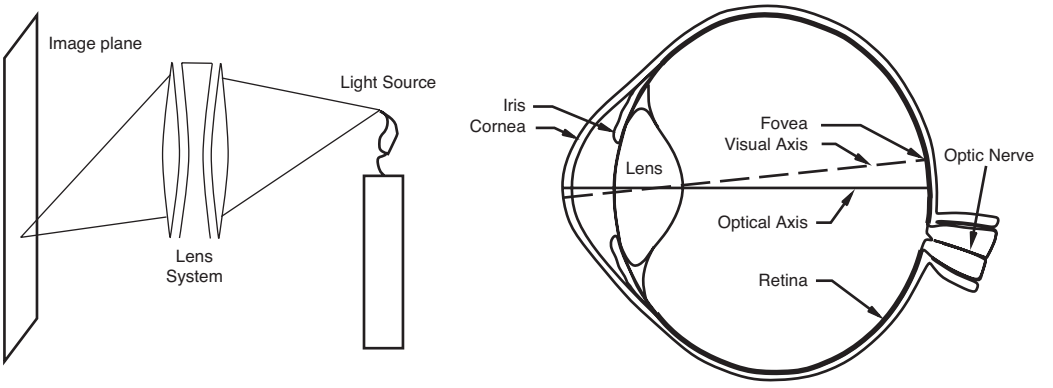


Figure 24.3 Lenses collect the light leaving a scene point in a range of directions, and steer it all to arrive at a single point on the image plane. Focusing works for points lying close to a focal plane in space; other points will not be focused properly. In cameras, elements of the lens system move to change the focal plane, whereas in the eye, the shape of the lens is changed by specialized muscles.

LENS

Vertebrate eyes and modern cameras use a **lens** system to gather sufficient light while keeping the image in focus. A large opening is covered with a lens that focuses light from nearby object locations down to nearby locations in the image plane. However, lens systems have a limited **depth of field**: they can focus light only from points that lie within a range of depths (centered around a **focal plane**). Objects outside this range will be out of focus in the image. To move the focal plane, the lens in the eye can change shape (Figure 24.3); in a camera, the lenses move back and forth.

DEPTH OF FIELD

FOCAL PLANE

24.1.3 Scaled orthographic projection

Perspective effects aren't always pronounced. For example, spots on a distant leopard may look small because the leopard is far away, but two spots that are next to each other will have about the same size. This is because the difference in distance to the spots is small compared to the distance to them, and so we can simplify the projection model. The appropriate model is **scaled orthographic projection**. The idea is as follows: If the depth Z of points on the object varies within some range $Z_0 \pm \Delta Z$, with $\Delta Z \ll Z_0$, then the perspective scaling factor f/Z can be approximated by a constant $s = f/Z_0$. The equations for projection from the scene coordinates (X, Y, Z) to the image plane become $x = sX$ and $y = sY$. Scaled orthographic projection is an approximation that is valid only for those parts of the scene with not much internal depth variation. For example, scaled orthographic projection can be a good model for the features on the front of a distant building.

SCALED ORTHOGRAPHIC PROJECTION

24.1.4 Light and shading

The brightness of a pixel in the image is a function of the brightness of the surface patch in the scene that projects to the pixel. We will assume a linear model (current cameras have non-linearities at the extremes of light and dark, but are linear in the middle). Image brightness is

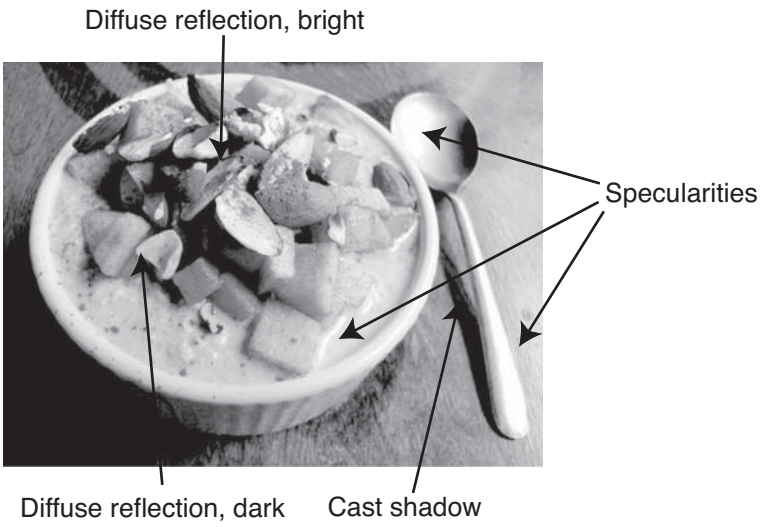


Figure 24.4 A variety of illumination effects. There are specularities on the metal spoon and on the milk. The bright diffuse surface is bright because it faces the light direction. The dark diffuse surface is dark because it is tangential to the illumination direction. The shadows appear at surface points that cannot see the light source. Photo by Mike Linksvayer (mlinksva on flickr).

a strong, if ambiguous, cue to the shape of an object, and from there to its identity. People are usually able to distinguish the three main causes of varying brightness and reverse-engineer the object's properties. The first cause is **overall intensity** of the light. Even though a white object in shadow may be less bright than a black object in direct sunlight, the eye can distinguish relative brightness well, and perceive the white object as white. Second, different points in the scene may **reflect** more or less of the light. Usually, the result is that people perceive these points as lighter or darker, and so see texture or markings on the object. Third, surface patches facing the light are brighter than surface patches tilted away from the light, an effect known as **shading**. Typically, people can tell that this shading comes from the geometry of the object, but sometimes get shading and markings mixed up. For example, a streak of dark makeup under a cheekbone will often look like a shading effect, making the face look thinner.

OVERALL INTENSITY

REFLECT

SHADING

DIFFUSE REFLECTION

Most surfaces reflect light by a process of **diffuse reflection**. Diffuse reflection scatters light evenly across the directions leaving a surface, so the brightness of a diffuse surface doesn't depend on the viewing direction. Most cloth, paints, rough wooden surfaces, vegetation, and rough stone are diffuse. Mirrors are not diffuse, because what you see depends on the direction in which you look at the mirror. The behavior of a perfect mirror is known as **specular reflection**. Some surfaces—such as brushed metal, plastic, or a wet floor—display small patches where specular reflection has occurred, called **specularities**. These are easy to identify, because they are small and bright (Figure 24.4). For almost all purposes, it is enough to model all surfaces as being diffuse with specularities.

SPECULAR REFLECTION

SPECULARITIES

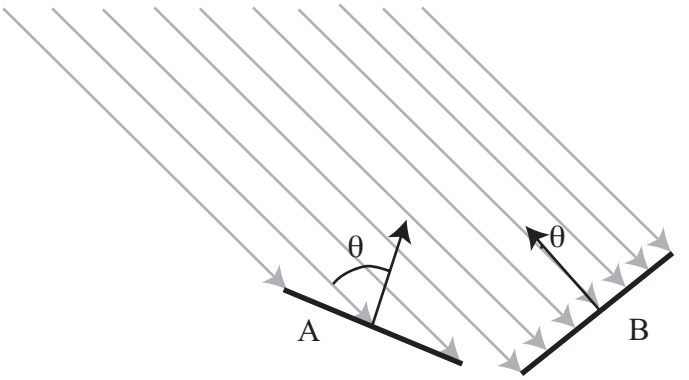


Figure 24.5 Two surface patches are illuminated by a distant point source, whose rays are shown as gray arrowheads. Patch A is tilted away from the source (θ is close to 90^0) and collects less energy, because it cuts fewer light rays per unit surface area. Patch B, facing the source (θ is close to 0^0), collects more energy.

DISTANT POINT LIGHT SOURCE

The main source of illumination outside is the sun, whose rays all travel parallel to one another. We model this behavior as a **distant point light source**. This is the most important model of lighting, and is quite effective for indoor scenes as well as outdoor scenes. The amount of light collected by a surface patch in this model depends on the angle θ between the illumination direction and the normal to the surface.

DIFFUSE ALBEDO

A diffuse surface patch illuminated by a distant point light source will reflect some fraction of the light it collects; this fraction is called the **diffuse albedo**. White paper and snow have a high albedo, about 0.90, whereas flat black velvet and charcoal have a low albedo of about 0.05 (which means that 95% of the incoming light is absorbed within the fibers of the velvet or the pores of the charcoal).

LAMBERT'S COSINE LAW

Lambert's cosine law states that the brightness of a diffuse patch is given by

$$I = \rho I_0 \cos\theta \ ,$$

where ρ is the diffuse albedo, I_0 is the intensity of the light source and θ is the angle between the light source direction and the surface normal (see Figure 24.5). Lampert's law predicts bright image pixels come from surface patches that face the light directly and dark pixels come from patches that see the light only tangentially, so that the shading on a surface provides some shape information. We explore this cue in Section 24.4.5. If the surface is not reached by the light source, then it is in **shadow**. Shadows are very seldom a uniform black, because the shadowed surface receives some light from other sources. Outdoors, the most important such source is the sky, which is quite bright. Indoors, light reflected from other surfaces illuminates shadowed patches. These **interreflections** can have a significant effect on the brightness of other surfaces, too. These effects are sometimes modeled by adding a constant **ambient illumination** term to the predicted intensity.

SHADOW

INTERREFLECTIONS

AMBIENT ILLUMINATION

24.1.5 Color

Fruit is a bribe that a tree offers to animals to carry its seeds around. Trees have evolved to have fruit that turns red or yellow when ripe, and animals have evolved to detect these color changes. Light arriving at the eye has different amounts of energy at different wavelengths; this can be represented by a spectral energy density function. Human eyes respond to light in the 380–750nm wavelength region, with three different types of color receptor cells, which have peak receptiveness at 420mm (blue), 540nm (green), and 570nm (red). The human eye can capture only a small fraction of the full spectral energy density function—but it is enough to tell when the fruit is ripe.

PRINCIPLE OF TRICHROMACY

The **principle of trichromacy** states that for any spectral energy density, no matter how complicated, it is possible to construct another spectral energy density consisting of a mixture of just three colors—usually red, green, and blue—such that a human can't tell the difference between the two. That means that our TVs and computer displays can get by with just the three red/green/blue (or R/G/B) color elements. It makes our computer vision algorithms easier, too. Each surface can be modeled with three different albedos for R/G/B. Similarly, each light source can be modeled with three R/G/B intensities. We then apply Lambert's cosine law to each to get three R/G/B pixel values. This model predicts, correctly, that the same surface will produce different colored image patches under different-colored lights. In fact, human observers are quite good at ignoring the effects of different colored lights and are able to estimate the color of the surface under white light, an effect known as **color constancy**. Quite accurate color constancy algorithms are now available; simple versions show up in the "auto white balance" function of your camera. Note that if we wanted to build a camera for mantis shrimp, we would need 12 different pixel colors, corresponding to the 12 types of color receptors of the crustacean.

COLOR CONSTANCY

24.2 Early Image-Processing Operations

We have seen how light reflects off objects in the scene to form an image consisting of, say, five million 3-byte pixels. With all sensors there will be noise in the image, and in any case there is a lot of data to deal with. So how do we get started on analyzing this data?

In this section we will study three useful image-processing operations: edge detection, texture analysis, and computation of optical flow. These are called "early" or "low-level" operations because they are the first in a pipeline of operations. Early vision operations are characterized by their local nature (they can be carried out in one part of the image without regard for anything more than a few pixels away) and by their lack of knowledge: we can perform these operations without consideration of the objects that might be present in the scene. This makes the low-level operations good candidates for implementation in parallel hardware—either in a graphics processor unit (GPU) or an eye. We will then look at one mid-level operation: segmenting the image into regions.

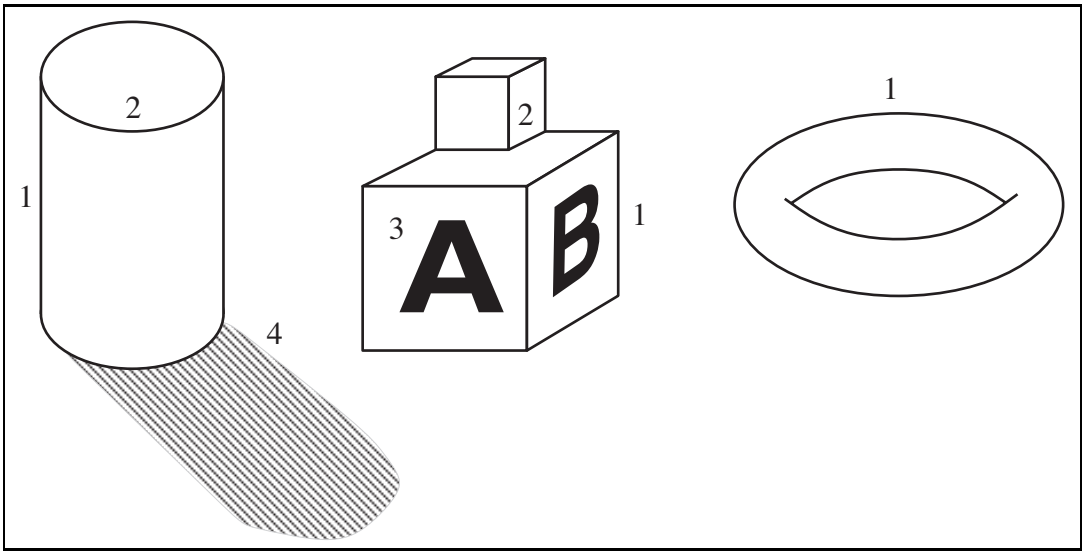


Figure 24.6 Different kinds of edges: (1) depth discontinuities; (2) surface orientation discontinuities; (3) reflectance discontinuities; (4) illumination discontinuities (shadows).

24.2.1 Edge detection

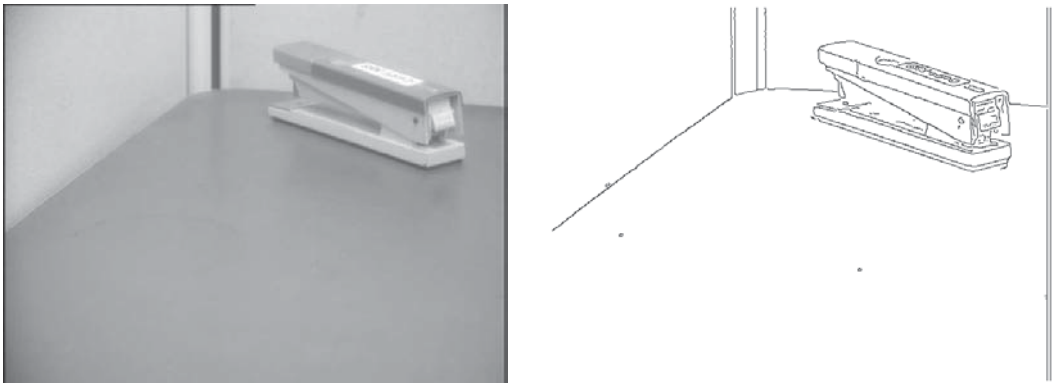
EDGE

Edges are straight lines or curves in the image plane across which there is a "significant" change in image brightness. The goal of edge detection is to abstract away from the messy, multimegabyte image and toward a more compact, abstract representation, as in Figure 24.6. The motivation is that edge contours in the image correspond to important scene contours. In the figure we have three examples of depth discontinuity, labeled 1; two surface-normal discontinuities, labeled 2; a reflectance discontinuity, labeled 3; and an illumination discontinuity (shadow), labeled 4. Edge detection is concerned only with the image, and thus does not distinguish between these different types of scene discontinuities; later processing will.

Figure 24.7(a) shows an image of a scene containing a stapler resting on a desk, and (b) shows the output of an edge-detection algorithm on this image. As you can see, there is a difference between the output and an ideal line drawing. There are gaps where no edge appears, and there are "noise" edges that do not correspond to anything of significance in the scene. Later stages of processing will have to correct for these errors.

How do we detect edges in an image? Consider the profile of image brightness along a one-dimensional cross-section perpendicular to an edge—for example, the one between the left edge of the desk and the wall. It looks something like what is shown in Figure 24.8 (top).

Edges correspond to locations in images where the brightness undergoes a sharp change, so a naive idea would be to differentiate the image and look for places where the magnitude of the derivative $I'(x)$ is large. That almost works. In Figure 24.8 (middle), we see that there is indeed a peak at $x = 50$, but there are also subsidiary peaks at other locations (e.g., $x = 75$). These arise because of the presence of noise in the image. If we smooth the image first, the spurious peaks are diminished, as we see in the bottom of the figure.



(a) (b)

Figure 24.7 (a) Photograph of a stapler. (b) Edges computed from (a).

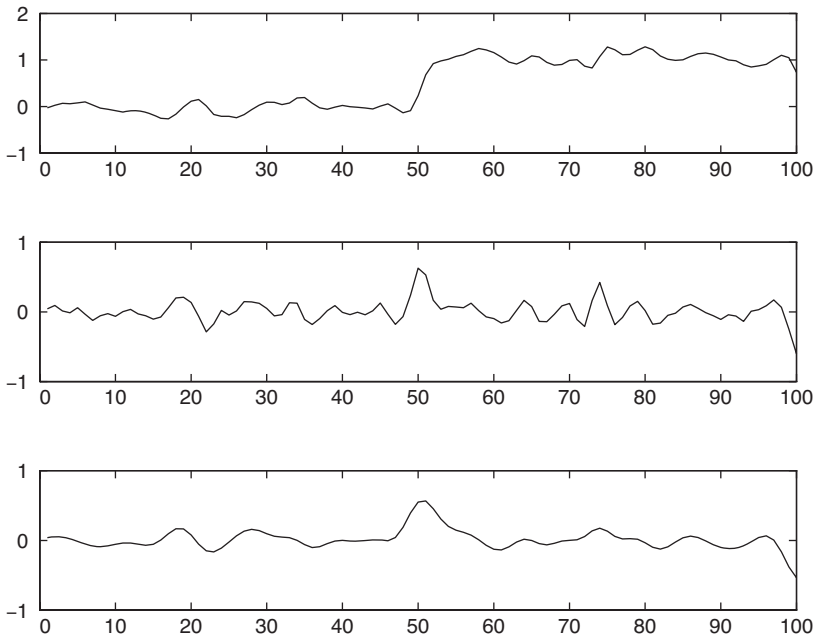


Figure 24.8 Top: Intensity profile $I(x)$ along a one-dimensional section across an edge at $x = 50$. Middle: The derivative of intensity, $I'(x)$. Large values of this function correspond to edges, but the function is noisy. Bottom: The derivative of a smoothed version of the intensity, $(I * G_\sigma)'$, which can be computed in one step as the convolution $I * G'_\sigma$. The noisy candidate edge at $x = 75$ has disappeared.

The measurement of brightness at a pixel in a CCD camera is based on a physical process involving the absorption of photons and the release of electrons; inevitably there will be statistical fluctuations of the measurement—noise. The noise can be modeled with

a Gaussian probability distribution, with each pixel independent of the others. One way to smooth an image is to assign to each pixel the average of its neighbors. This tends to cancel out extreme values. But how many neighbors should we consider—one pixel away, or two, or more? One good answer is a weighted average that weights the nearest pixels the most, then gradually decreases the weight for more distant pixels. The **Gaussian filter** does just that. (Users of Photoshop recognize this as the *Gaussian blur* operation.) Recall that the Gaussian function with standard deviation σ and mean 0 is

GAUSSIAN FILTER

$$N_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \qquad \text{in one dimension, or}$$
$$N_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{in two dimensions.}$$

The application of the Gaussian filter replaces the intensity $I(x_0, y_0)$ with the sum, over all (x, y) pixels, of $I(x, y)\, N_\sigma(d)$, where d is the distance from (x_0, y_0) to (x, y). This kind of weighted sum is so common that there is a special name and notation for it. We say that the function h is the **convolution** of two functions f and g (denoted $f * g$) if we have

CONVOLUTION

$$h(x) = (f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)\, g(x-u) \qquad \text{in one dimension, or}$$

$$h(x,y) = (f * g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)\, g(x-u, y-v) \qquad \text{in two.}$$

So the smoothing function is achieved by convolving the image with the Gaussian, $I * N_\sigma$. A σ of 1 pixel is enough to smooth over a small amount of noise, whereas 2 pixels will smooth a larger amount, but at the loss of some detail. Because the Gaussian's influence fades quickly at a distance, we can replace the $\pm\infty$ in the sums with $\pm 3\sigma$.

We can optimize the computation by combining smoothing and edge finding into a single operation. It is a theorem that for any functions f and g, the derivative of the convolution, $(f * g)'$, is equal to the convolution with the derivative, $f * (g')$. So rather than smoothing the image and then differentiating, we can just convolve the image with the derivative of the smoothing function, N'_σ. We then mark as edges those peaks in the response that are above some threshold.

There is a natural generalization of this algorithm from one-dimensional cross sections to general two-dimensional images. In two dimensions edges may be at any angle θ. Considering the image brightness as a scalar function of the variables x, y, its gradient is a vector

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} = \begin{pmatrix} I_x \\ I_y \end{pmatrix} .$$

Edges correspond to locations in images where the brightness undergoes a sharp change, and so the magnitude of the gradient, $\|\nabla I\|$, should be large at an edge point. Of independent interest is the direction of the gradient

$$\frac{\nabla I}{\|\nabla I\|} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} .$$

ORIENTATION

This gives us a $\theta = \theta(x, y)$ at every pixel, which defines the edge **orientation** at that pixel.

As in one dimension, to form the gradient we don't compute ∇I, but rather $\nabla(I * N_\sigma)$, the gradient after smoothing the image by convolving it with a Gaussian. And again, the shortcut is that this is equivalent to convolving the image with the partial derivatives of a Gaussian. Once we have computed the gradient, we can obtain edges by finding edge points and linking them together. To tell whether a point is an edge point, we must look at other points a small distance forward and back along the direction of the gradient. If the gradient magnitude at one of these points is larger, then we could get a better edge point by shifting the edge curve very slightly. Furthermore, if the gradient magnitude is too small, the point cannot be an edge point. So at an edge point, the gradient magnitude is a local maximum along the direction of the gradient, and the gradient magnitude is above a suitable threshold.

Once we have marked edge pixels by this algorithm, the next stage is to link those pixels that belong to the same edge curves. This can be done by assuming that any two neighboring edge pixels with consistent orientations must belong to the same edge curve.

24.2.2 Texture

TEXTURE

In everyday language, **texture** is the visual feel of a surface—what you see evokes what the surface might feel like if you touched it ("texture" has the same root as "textile"). In computational vision, texture refers to a spatially repeating pattern on a surface that can be sensed visually. Examples include the pattern of windows on a building, stitches on a sweater, spots on a leopard, blades of grass on a lawn, pebbles on a beach, and people in a stadium. Sometimes the arrangement is quite periodic, as in the stitches on a sweater; in other cases, such as pebbles on a beach, the regularity is only statistical.

Whereas brightness is a property of individual pixels, the concept of texture makes sense only for a multipixel patch. Given such a patch, we could compute the orientation at each pixel, and then characterize the patch by a histogram of orientations. The texture of bricks in a wall would have two peaks in the histogram (one vertical and one horizontal), whereas the texture of spots on a leopard's skin would have a more uniform distribution of orientations.

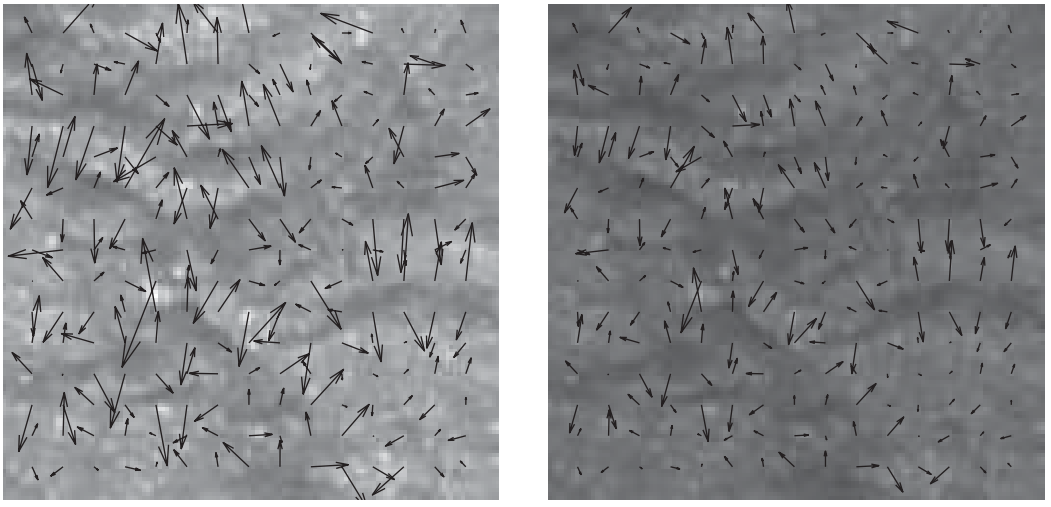
Figure 24.9 shows that orientations are largely invariant to changes in illumination. This makes texture an important clue for object recognition, because other clues, such as edges, can yield different results in different lighting conditions.

In images of textured objects, edge detection does not work as well as it does for smooth objects. This is because the most important edges can be lost among the texture elements. Quite literally, we may miss the tiger for the stripes. The solution is to look for differences in texture properties, just the way we look for differences in brightness. A patch on a tiger and a patch on the grassy background will have very different orientation histograms, allowing us to find the boundary curve between them.

24.2.3 Optical flow

Next, let us consider what happens when we have a video sequence, instead of just a single static image. When an object in the video is moving, or when the camera is moving relative to an object, the resulting apparent motion in the image is called **optical flow**. Optical flow describes the direction and speed of motion of features *in the image*—the optical flow of a

OPTICAL FLOW



(a) (b)

Figure 24.9 Two images of the same texture of crumpled rice paper, with different illumination levels. The gradient vector field (at every eighth pixel) is plotted on top of each one. Notice that, as the light gets darker, all the gradient vectors get shorter. The vectors do not rotate, so the gradient orientations do not change.

video of a race car would be measured in pixels per second, not miles per hour. The optical flow encodes useful information about scene structure. For example, in a video of scenery taken from a moving train, distant objects have slower apparent motion than close objects; thus, the rate of apparent motion can tell us something about distance. Optical flow also enables us to recognize actions. In Figure 24.10(a) and (b), we show two frames from a video of a tennis player. In (c) we display the optical flow vectors computed from these images, showing that the racket and front leg are moving fastest.

The optical flow vector field can be represented at any point (x, y) by its components $v_x(x, y)$ in the x direction and $v_y(x, y)$ in the y direction. To measure optical flow we need to find corresponding points between one time frame and the next. A simple-minded technique is based on the fact that image patches around corresponding points have similar intensity patterns. Consider a block of pixels centered at pixel p, (x_0, y_0), at time t_0. This block of pixels is to be compared with pixel blocks centered at various candidate pixels at $(x_0 + D_x, y_0 + D_y)$ at time $t_0 + D_t$. One possible measure of similarity is the **sum of squared differences** (SSD):

SUM OF SQUARED DIFFERENCES

$$\text{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x, y, t) - I(x + D_x, y + D_y, t + D_t))^2 \ .$$

Here, (x, y) ranges over pixels in the block centered at (x_0, y_0). We find the (D_x, D_y) that minimizes the SSD. The optical flow at (x_0, y_0) is then $(v_x, v_y) = (D_x/D_t, D_y/D_t)$. Note that for this to work, there needs to be some texture or variation in the scene. If one is looking at a uniform white wall, then the SSD is going to be nearly the same for the different can-

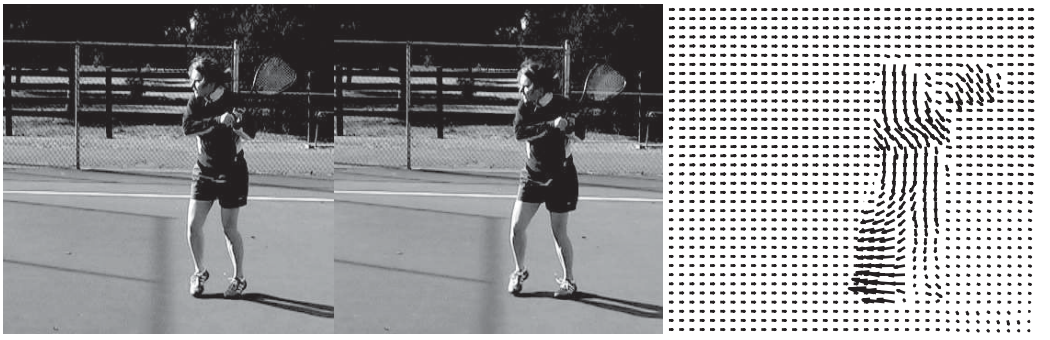


Figure 24.10 Two frames of a video sequence. On the right is the optical flow field corresponding to the displacement from one frame to the other. Note how the movement of the tennis racket and the front leg is captured by the directions of the arrows. (Courtesy of Thomas Brox.)

didate matches, and the algorithm is reduced to making a blind guess. The best-performing algorithms for measuring optical flow rely on a variety of additional constraints when the scene is only partially textured.

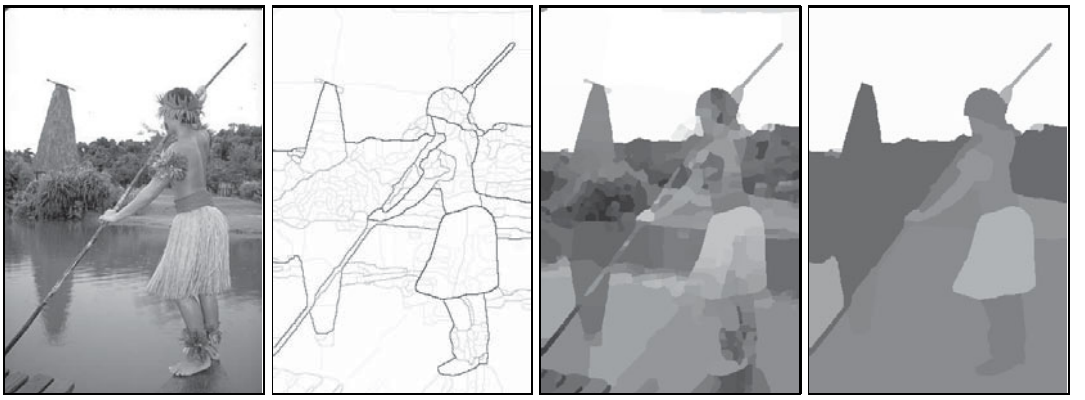
24.2.4 Segmentation of images

SEGMENTATION
REGIONS

Segmentation is the process of breaking an image into **regions** of similar pixels. Each image pixel can be associated with certain visual properties, such as brightness, color, and texture. Within an object, or a single part of an object, these attributes vary relatively little, whereas across an inter-object boundary there is typically a large change in one or more of these attributes. There are two approaches to segmentation, one focusing on detecting the boundaries of these regions, and the other on detecting the regions themselves (Figure 24.11).

A boundary curve passing through a pixel (x, y) will have an orientation θ, so one way to formalize the problem of detecting boundary curves is as a machine learning classification problem. Based on features from a local neighborhood, we want to compute the probability $P_b(x, y, \theta)$ that indeed there is a boundary curve at that pixel along that orientation. Consider a circular disk centered at (x, y), subdivided into two half disks by a diameter oriented at θ. If there is a boundary at (x, y, θ) the two half disks might be expected to differ significantly in their brightness, color, and texture. Martin, Fowlkes, and Malik (2004) used features based on differences in histograms of brightness, color, and texture values measured in these two half disks, and then trained a classifier. For this they used a data set of natural images where humans had marked the "ground truth" boundaries, and the goal of the classifier was to mark exactly those boundaries marked by humans and no others.

Boundaries detected by this technique turn out to be significantly better than those found using the simple edge-detection technique described previously. But still there are two limitations. (1) The boundary pixels formed by thresholding $P_b(x, y, \theta)$ are not guaranteed to form closed curves, so this approach doesn't deliver regions, and (2) the decision making exploits only local context and does not use global consistency constraints.



(a) (b) (c) (d)

Figure 24.11 (a) Original image. (b) Boundary contours, where the higher the P_b value, the darker the contour. (c) Segmentation into regions, corresponding to a fine partition of the image. Regions are rendered in their mean colors. (d) Segmentation into regions, corresponding to a coarser partition of the image, resulting in fewer regions. (Courtesy of Pablo Arbelaez, Michael Maire, Charles Fowlkes, and Jitendra Malik)

The alternative approach is based on trying to "cluster" the pixels into regions based on their brightness, color, and texture. Shi and Malik (2000) set this up as a graph partitioning problem. The nodes of the graph correspond to pixels, and edges to connections between pixels. The weight W_{ij} on the edge connecting a pair of pixels i and j is based on how similar the two pixels are in brightness, color, texture, etc. Partitions that minimize a *normalized cut* criterion are then found. Roughly speaking, the criterion for partitioning the graph is to minimize the sum of weights of connections across the groups of pixels and maximize the sum of weights of connections within the groups.

Segmentation based purely on low-level, local attributes such as brightness and color cannot be expected to deliver the final correct boundaries of all the objects in the scene. To reliably find object boundaries we need high-level knowledge of the likely kinds of objects in the scene. Representing this knowledge is a topic of active research. A popular strategy is to produce an over-segmentation of an image, containing hundreds of homogeneous regions known as **superpixels**. From there, knowledge-based algorithms can take over; they will find it easier to deal with hundreds of superpixels rather than millions of raw pixels. How to exploit high-level knowledge of objects is the subject of the next section.

SUPERPIXELS

24.3 Object Recognition by Appearance

APPEARANCE

Appearance is shorthand for what an object tends to look like. Some object categories—for example, baseballs—vary rather little in appearance; all of the objects in the category look about the same under most circumstances. In this case, we can compute a set of features describing each class of images likely to contain the object, then test it with a classifier.

Other object categories—for example, houses or ballet dancers—vary greatly. A house can have different size, color, and shape and can look different from different angles. A dancer looks different in each pose, or when the stage lights change colors. A useful abstraction is to say that some objects are made up of local patterns which tend to move around with respect to one another. We can then find the object by looking at local histograms of detector responses, which expose whether some part is present but suppress the details of where it is.

Testing each class of images with a learned classifier is an important general recipe. It works extremely well for faces looking directly at the camera, because at low resolution and under reasonable lighting, all such faces look quite similar. The face is round, and quite bright compared to the eye sockets; these are dark, because they are sunken, and the mouth is a dark slash, as are the eyebrows. Major changes of illumination can cause some variations in this pattern, but the range of variation is quite manageable. That makes it possible to detect face positions in an image that contains faces. Once a computational challenge, this feature is now commonplace in even inexpensive digital cameras.

For the moment, we will consider only faces where the nose is oriented vertically; we will deal with rotated faces below. We sweep a round window of fixed size over the image, compute features for it, and present the features to a classifier. This strategy is sometimes called the **sliding window**. Features need to be robust to shadows and to changes in brightness caused by illumination changes. One strategy is to build features out of gradient orientations. Another is to estimate and correct the illumination in each image window. To find faces of different sizes, repeat the sweep over larger or smaller versions of the image. Finally, we postprocess the responses across scales and locations to produce the final set of detections.

SLIDING WINDOW

Postprocessing is important, because it is unlikely that we have chosen a window size that is exactly the right size for a face (even if we use multiple sizes). Thus, we will likely have several overlapping windows that each report a match for a face. However, if we use a classifier that can report strength of response (for example, logistic regression or a support vector machine) we can combine these partial overlapping matches at nearby locations to yield a single high-quality match. That gives us a face detector that can search over locations and scales. To search rotations as well, we use two steps. We train a regression procedure to estimate the best orientation of any face present in a window. Now, for each window, we estimate the orientation, reorient the window, then test whether a vertical face is present with our classifier. All this yields a system whose architecture is sketched in Figure 24.12.

Training data is quite easily obtained. There are several data sets of marked-up face images, and rotated face windows are easy to build (just rotate a window from a training data set). One trick that is widely used is to take each example window, then produce new examples by changing the orientation of the window, the center of the window, or the scale very slightly. This is an easy way of getting a bigger data set that reflects real images fairly well; the trick usually improves performance significantly. Face detectors built along these lines now perform very well for frontal faces (side views are harder).

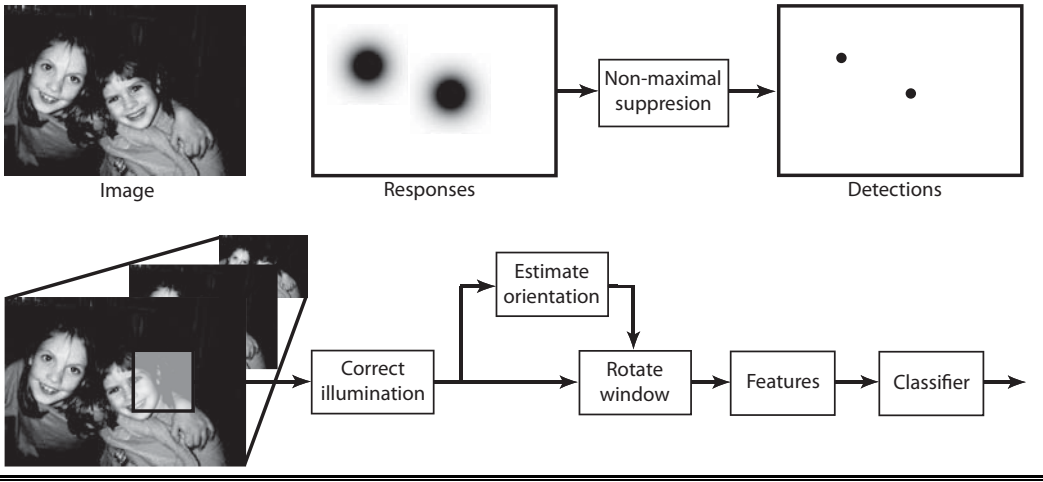


Figure 24.12 Face finding systems vary, but most follow the architecture illustrated in two parts here. On the top, we go from images to responses, then apply non-maximum suppression to find the strongest local response. The responses are obtained by the process illustrated on the bottom. We sweep a window of fixed size over larger and smaller versions of the image, so as to find smaller or larger faces, respectively. The illumination in the window is corrected, and then a regression engine (quite often, a neural net) predicts the orientation of the face. The window is corrected to this orientation and then presented to a classifier. Classifier outputs are then postprocessed to ensure that only one face is placed at each location in the image.

24.3.1 Complex appearance and pattern elements

Many objects produce much more complex patterns than faces do. This is because several effects can move features around in an image of the object. Effects include (Figure 24.13)

- **Foreshortening**, which causes a pattern viewed at a slant to be significantly distorted.
- **Aspect**, which causes objects to look different when seen from different directions. Even as simple an object as a doughnut has several aspects; seen from the side, it looks like a flattened oval, but from above it is an annulus.
- **Occlusion**, where some parts are hidden from some viewing directions. Objects can occlude one another, or parts of an object can occlude other parts, an effect known as self-occlusion.
- **Deformation**, where internal degrees of freedom of the object change its appearance. For example, people can move their arms and legs around, generating a very wide range of different body configurations.

However, our recipe of searching across location and scale can still work. This is because some structure will be present in the images produced by the object. For example, a picture of a car is likely to show some of headlights, doors, wheels, windows, and hubcaps, though they may be in somewhat different arrangements in different pictures. This suggests modeling objects with pattern elements—collections of parts. These pattern elements may move around

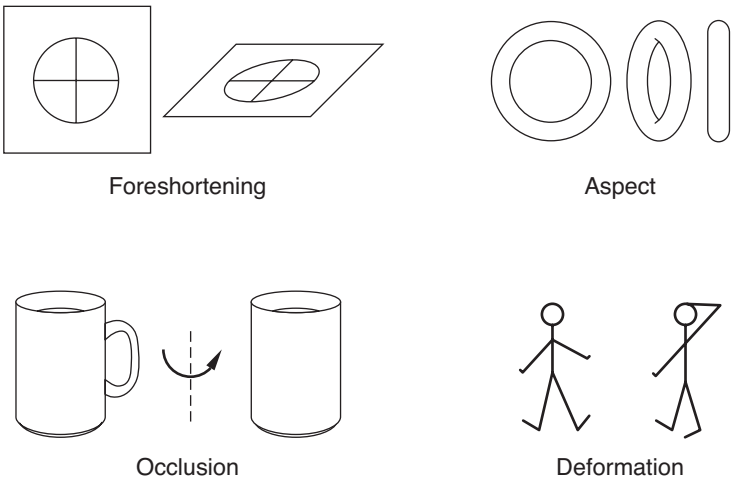


Figure 24.13 Sources of appearance variation. First, elements can foreshorten, like the circular patch on the top left. This patch is viewed at a slant, and so is elliptical in the image. Second, objects viewed from different directions can change shape quite dramatically, a phenomenon known as aspect. On the top right are three different aspects of a doughnut. Occlusion causes the handle of the mug on the bottom left to disappear when the mug is rotated. In this case, because the body and handle belong to the same mug, we have self-occlusion. Finally, on the bottom right, some objects can deform dramatically.

with respect to one another, but if most of the pattern elements are present in about the right place, then the object is present. An object recognizer is then a collection of features that can tell whether the pattern elements are present, and whether they are in about the right place.

The most obvious approach is to represent the image window with a histogram of the pattern elements that appear there. This approach does not work particularly well, because too many patterns get confused with one another. For example, if the pattern elements are color pixels, the French, UK, and Netherlands flags will get confused because they have approximately the same color histograms, though the colors are arranged in very different ways. Quite simple modifications of histograms yield very useful features. The trick is to preserve some spatial detail in the representation; for example, headlights tend to be at the front of a car and wheels tend to be at the bottom. Histogram-based features have been successful in a wide variety of recognition applications; we will survey pedestrian detection.

24.3.2 Pedestrian detection with HOG features

The World Bank estimates that each year car accidents kill about 1.2 million people, of whom about two thirds are pedestrians. This means that detecting pedestrians is an important application problem, because cars that can automatically detect and avoid pedestrians might save many lives. Pedestrians wear many different kinds of clothing and appear in many different configurations, but, at relatively low resolution, pedestrians can have a fairly characteristic appearance. The most usual cases are lateral or frontal views of a walk. In these cases,

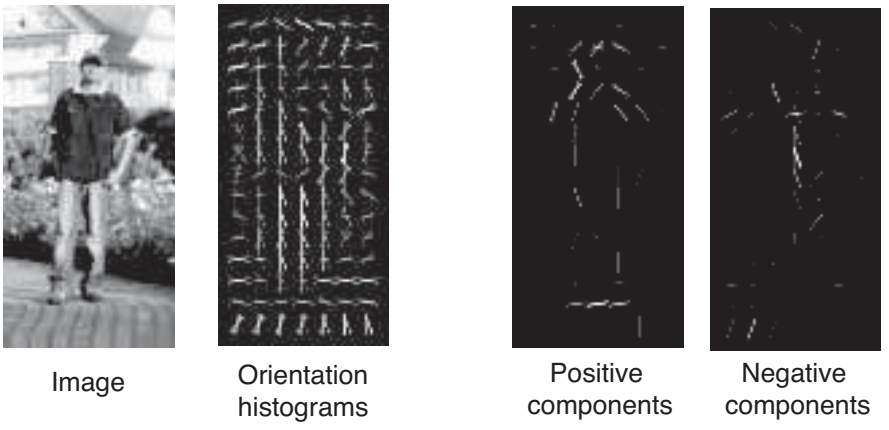


Figure 24.14 Local orientation histograms are a powerful feature for recognizing even quite complex objects. On the left, an image of a pedestrian. On the center left, local orientation histograms for patches. We then apply a classifier such as a support vector machine to find the weights for each histogram that best separate the positive examples of pedestrians from non-pedestrians. We see that the positively weighted components look like the outline of a person. The negative components are less clear; they represent all the patterns that are not pedestrians. Figure from Dalal and Triggs (2005) © IEEE.

we see either a "lollipop" shape — the torso is wider than the legs, which are together in the stance phase of the walk — or a "scissor" shape — where the legs are swinging in the walk. We expect to see some evidence of arms and legs, and the curve around the shoulders and head also tends to visible and quite distinctive. This means that, with a careful feature construction, we can build a useful moving-window pedestrian detector.

There isn't always a strong contrast between the pedestrian and the background, so it is better to use orientations than edges to represent the image window. Pedestrians can move their arms and legs around, so we should use a histogram to suppress some spatial detail in the feature. We break up the window into cells, which could overlap, and build an orientation histogram in each cell. Doing so will produce a feature that can tell whether the head-and-shoulders curve is at the top of the window or at the bottom, but will not change if the head moves slightly.

One further trick is required to make a good feature. Because orientation features are not affected by illumination brightness, we cannot treat high-contrast edges specially. This means that the distinctive curves on the boundary of a pedestrian are treated in the same way as fine texture detail in clothing or in the background, and so the signal may be submerged in noise. We can recover contrast information by counting gradient orientations with weights that reflect how significant a gradient is compared to other gradients in the same cell. We will write $\| \nabla I_{\mathbf{x}} \|$ for the gradient magnitude at point $\mathbf{x}$ in the image, write $\mathcal{C}$ for the cell whose histogram we wish to compute, and write $w_{\mathbf{x},\mathcal{C}}$ for the weight that we will use for the

Figure 24.15 Another example of object recognition, this one using the SIFT feature (Scale Invariant Feature Transform), an earlier version of the HOG feature. On the **left**, images of a shoe and a telephone that serve as object models. In the **center**, a test image. On the **right**, the shoe and the telephone have been detected by: finding points in the image whose SIFT feature descriptions match a model; computing an estimate of pose of the model; and verifying that estimate. A strong match is usually verified with rare false positives. Images from Lowe (1999) © IEEE.

orientation at $\mathbf{x}$ for this cell. A natural choice of weight is

$$w_{\mathbf{x},\mathcal{C}} = \frac{\|\nabla I_{\mathbf{x}}\|}{\sum_{\mathbf{u}\in\mathcal{C}} \|\nabla I_{\mathbf{u}}\|} .$$

This compares the gradient magnitude to others in the cell, so gradients that are large compared to their neighbors get a large weight. The resulting feature is usually called a **HOG feature** (for Histogram Of Gradient orientations).

HOG FEATURE

This feature construction is the main way in which pedestrian detection differs from face detection. Otherwise, building a pedestrian detector is very like building a face detector. The detector sweeps a window across the image, computes features for that window, then presents it to a classifier. Non-maximum suppression needs to be applied to the output. In most applications, the scale and orientation of typical pedestrians is known. For example, in driving applications in which a camera is fixed to the car, we expect to view mainly vertical pedestrians, and we are interested only in nearby pedestrians. Several pedestrian data sets have been published, and these can be used for training the classifier.

Pedestrians are not the only type of object we can detect. In Figure 24.15 we see that similar techniques can be used to find a variety of objects in different contexts.

24.4 Reconstructing the 3D World

In this section we show how to go from the two-dimensional image to a three-dimensional representation of the scene. The fundamental question is this: Given that all points in the scene that fall along a ray to the pinhole are projected to the same point in the image, how do we recover three-dimensional information? Two ideas come to our rescue:

- If we have two (or more) images from different camera positions, then we can triangulate to find the position of a point in the scene.
- We can exploit background knowledge about the physical scene that gave rise to the image. Given an object model $\mathbf{P}(Scene)$ and a rendering model $\mathbf{P}(Image \mid Scene)$, we can compute a posterior distribution $\mathbf{P}(Scene \mid Image)$.

There is as yet no single unified theory for scene reconstruction. We survey eight commonly used visual cues: **motion**, **binocular stereopsis**, **multiple views**, **texture**, **shading**, **contour**, and **familiar objects**.

24.4.1 Motion parallax

If the camera moves relative to the three-dimensional scene, the resulting apparent motion in the image, optical flow, can be a source of information for both the movement of the camera and depth in the scene. To understand this, we state (without proof) an equation that relates the optical flow to the viewer's translational velocity $\mathbf{T}$ and the depth in the scene.

The components of the optical flow field are

$$v_x(x,y) = \frac{-T_x + xT_z}{Z(x,y)}, \qquad v_y(x,y) = \frac{-T_y + yT_z}{Z(x,y)} ,$$

where $Z(x,y)$ is the z-coordinate of the point in the scene corresponding to the point in the image at (x,y).

FOCUS OF EXPANSION

Note that both components of the optical flow, $v_x(x,y)$ and $v_y(x,y)$, are zero at the point $x = T_x/T_z, y = T_y/T_z$. This point is called the **focus of expansion** of the flow field. Suppose we change the origin in the x–y plane to lie at the focus of expansion; then the expressions for optical flow take on a particularly simple form. Let (x', y') be the new coordinates defined by $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Then

$$v_x(x',y') = \frac{x'T_z}{Z(x',y')}, \qquad v_y(x',y') = \frac{y'T_z}{Z(x',y')} .$$

Note that there is a scale-factor ambiguity here. If the camera was moving twice as fast, and every object in the scene was twice as big and at twice the distance to the camera, the optical flow field would be exactly the same. But we can still extract quite useful information.

1. Suppose you are a fly trying to land on a wall and you want to know the time-to-contact at the current velocity. This time is given by Z/T_z. Note that although the instantaneous optical flow field cannot provide either the distance Z or the velocity component T_z, it can provide the ratio of the two and can therefore be used to control the landing approach. There is considerable experimental evidence that many different animal species exploit this cue.
2. Consider two points at depths Z_1, Z_2, respectively. We may not know the absolute value of either of these, but by considering the inverse of the ratio of the optical flow magnitudes at these points, we can determine the depth ratio Z_1/Z_2. This is the cue of motion parallax, one we use when we look out of the side window of a moving car or train and infer that the slower moving parts of the landscape are farther away.

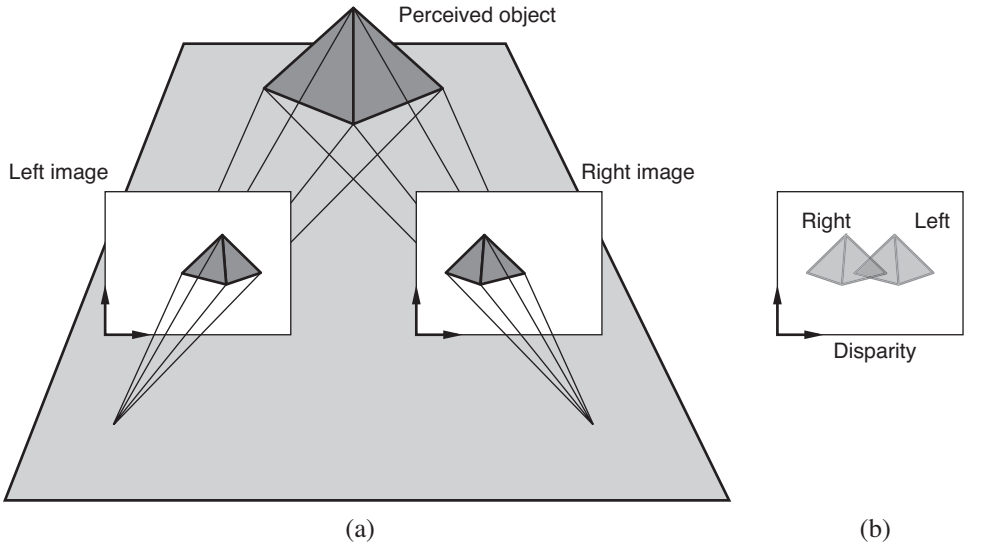


Figure 24.16 Translating a camera parallel to the image plane causes image features to move in the camera plane. The disparity in positions that results is a cue to depth. If we superimpose left and right image, as in (b), we see the disparity.

24.4.2 Binocular stereopsis

BINOCULAR STEREOPSIS

DISPARITY

Most vertebrates have *two* eyes. This is useful for redundancy in case of a lost eye, but it helps in other ways too. Most prey have eyes on the side of the head to enable a wider field of vision. Predators have the eyes in the front, enabling them to use **binocular stereopsis**. The idea is similar to motion parallax, except that instead of using images over time, we use two (or more) images separated in space. Because a given feature in the scene will be in a different place relative to the z-axis of each image plane, if we superpose the two images, there will be a **disparity** in the location of the image feature in the two images. You can see this in Figure 24.16, where the nearest point of the pyramid is shifted to the left in the right image and to the right in the left image.

Note that to measure disparity we need to solve the correspondence problem, that is, determine for a point in the left image, the point in the right image that results from the projection of the same scene point. This is analogous to what one has to do in measuring optical flow, and the most simple-minded approaches are somewhat similar and based on comparing blocks of pixels around corresponding points using the sum of squared differences. In practice, we use much more sophisticated algorithms, which exploit additional constraints.

Assuming that we can measure disparity, how does this yield information about depth in the scene? We will need to work out the geometrical relationship between disparity and depth. First, we will consider the case when both the eyes (or cameras) are looking forward with their optical axes parallel. The relationship of the right camera to the left camera is then just a displacement along the x-axis by an amount b, the baseline. We can use the optical flow equations from the previous section, if we think of this as resulting from a translation

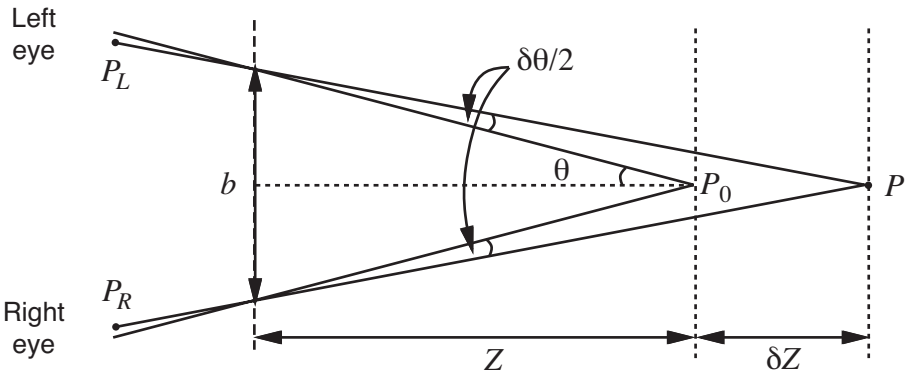


Figure 24.17 The relation between disparity and depth in stereopsis. The centers of projection of the two eyes are b apart, and the optical axes intersect at the fixation point P_0. The point P in the scene projects to points P_L and P_R in the two eyes. In angular terms, the disparity between these is $\delta\theta$. See text.

vector **T** acting for time δt, with $T_x = b/\delta t$ and $T_y = T_z = 0$. The horizontal and vertical disparity are given by the optical flow components, multiplied by the time step δt, $H = v_x\,\delta t$, $V = v_y\,\delta t$. Carrying out the substitutions, we get the result that $H = b/Z$, $V = 0$. In words, the horizontal disparity is equal to the ratio of the baseline to the depth, and the vertical disparity is zero. Given that we know b, we can measure H and recover the depth Z.

FIXATE

Under normal viewing conditions, humans **fixate**; that is, there is some point in the scene at which the optical axes of the two eyes intersect. Figure 24.17 shows two eyes fixated at a point P_0, which is at a distance Z from the midpoint of the eyes. For convenience, we will compute the *angular* disparity, measured in radians. The disparity at the point of fixation P_0 is zero. For some other point P in the scene that is δZ farther away, we can compute the angular displacements of the left and right images of P, which we will call P_L and P_R, respectively. If each of these is displaced by an angle $\delta\theta/2$ relative to P_0, then the displacement between P_L and P_R, which is the disparity of P, is just $\delta\theta$. From Figure 24.17, $\tan\theta = \frac{b/2}{Z}$ and $\tan(\theta - \delta\theta/2) = \frac{b/2}{Z+\delta Z}$, but for small angles, $\tan\theta \approx \theta$, so

$$\delta\theta/2 = \frac{b/2}{Z} - \frac{b/2}{Z+\delta Z} \approx \frac{b\delta Z}{2Z^2}$$

and, since the actual disparity is $\delta\theta$, we have

$$\text{disparity} = \frac{b\delta Z}{Z^2}\,.$$

BASELINE

In humans, b (the **baseline** distance between the eyes) is about 6 cm. Suppose that Z is about 100 cm. If the smallest detectable $\delta\theta$ (corresponding to the pixel size) is about 5 seconds of arc, this gives a δZ of 0.4 mm. For $Z = 30$ cm, we get the impressively small value $\delta Z = 0.036$ mm. That is, at a distance of 30 cm, humans can discriminate depths that differ by as little as 0.036 mm, enabling us to thread needles and the like.

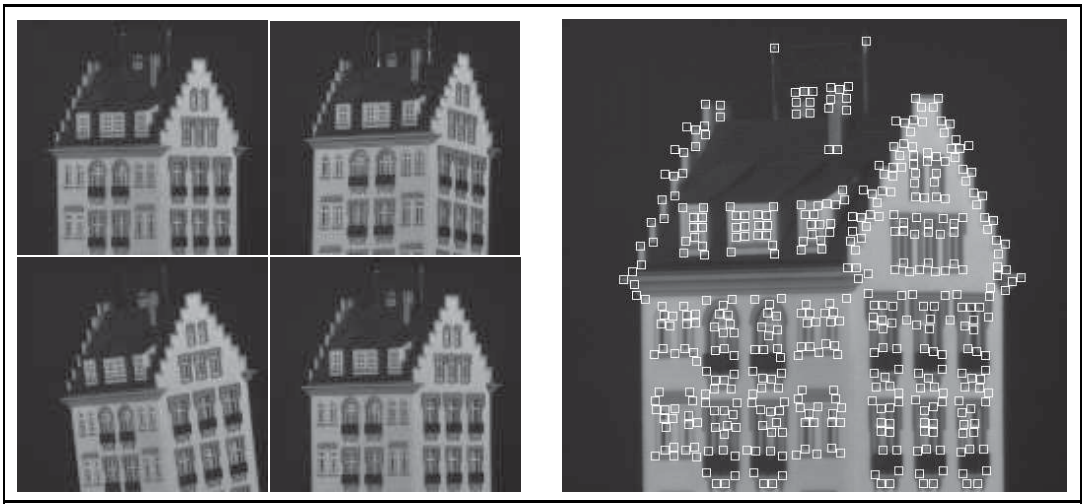


Figure 24.18 (a) Four frames from a video sequence in which the camera is moved and rotated relative to the object. (b) The first frame of the sequence, annotated with small boxes highlighting the features found by the feature detector. (Courtesy of Carlo Tomasi.)

24.4.3 Multiple views

Shape from optical flow or binocular disparity are two instances of a more general framework, that of exploiting multiple views for recovering depth. In computer vision, there is no reason for us to be restricted to differential motion or to only use two cameras converging at a fixation point. Therefore, techniques have been developed that exploit the information available in multiple views, even from hundreds or thousands of cameras. Algorithmically, there are three subproblems that need to be solved:

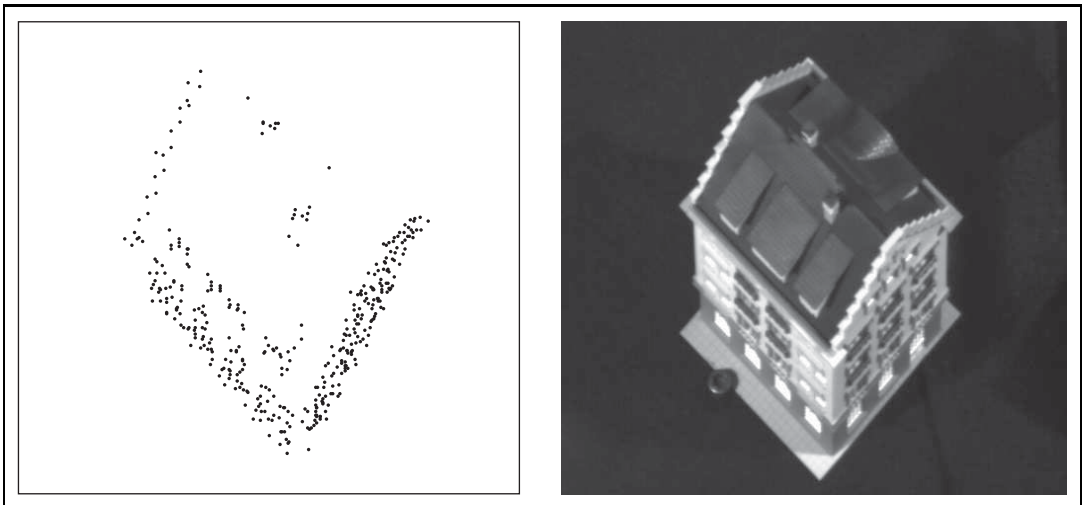
- The correspondence problem, i.e., identifying features in the different images that are projections of the same feature in the three-dimensional world.
- The relative orientation problem, i.e., determining the transformation (rotation and translation) between the coordinate systems fixed to the different cameras.
- The depth estimation problem, i.e., determining the depths of various points in the world for which image plane projections were available in at least two views

The development of robust matching procedures for the correspondence problem, accompanied by numerically stable algorithms for solving for relative orientations and scene depth, is one of the success stories of computer vision. Results from one such approach due to Tomasi and Kanade (1992) are shown in Figures 24.18 and 24.19.

24.4.4 Texture

Earlier we saw how texture was used for segmenting objects. It can also be used to estimate distances. In Figure 24.20 we see that a homogeneous texture in the scene results in varying texture elements, or **texels**, in the image. All the paving tiles in (a) are identical in the scene. They appear different in the image for two reasons:

TEXEL



(a) (b)

Figure 24.19 (a) Three-dimensional reconstruction of the locations of the image features in Figure 24.18, shown from above. (b) The real house, taken from the same position.

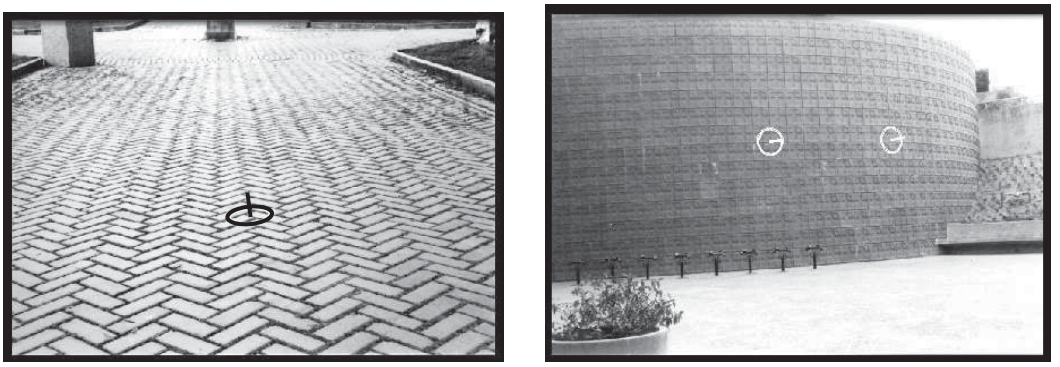
1. *Differences in the distances of the texels from the camera.* Distant objects appear smaller by a scaling factor of $1/Z$.
2. *Differences in the foreshortening of the texels.* If all the texels are in the ground plane then distance ones are viewed at an angle that is farther off the perpendicular, and so are more foreshortened. The magnitude of the foreshortening effect is proportional to $\cos\sigma$, where σ is the slant, the angle between the Z-axis and $\mathbf{n}$, the surface normal to the texel.

Researchers have developed various algorithms that try to exploit the variation in the appearance of the projected texels as a basis for determining surface normals. However, the accuracy and applicability of these algorithms is not anywhere as general as those based on using multiple views.

24.4.5 Shading

Shading—variation in the intensity of light received from different portions of a surface in a scene—is determined by the geometry of the scene and by the reflectance properties of the surfaces. In computer graphics, the objective is to compute the image brightness $I(x, y)$, given the scene geometry and reflectance properties of the objects in the scene. Computer vision aims to invert the process—that is, to recover the geometry and reflectance properties, given the image brightness $I(x, y)$. This has proved to be difficult to do in anything but the simplest cases.

From the physical model of section 24.1.4, we know that if a surface normal points toward the light source, the surface is brighter, and if it points away, the surface is darker. We cannot conclude that a dark patch has its normal pointing away from the light; instead, it could have low albedo. Generally, albedo changes quite quickly in images, and shading



(a) (b)

Figure 24.20 (a) A textured scene. Assuming that the real texture is uniform allows recovery of the surface orientation. The computed surface orientation is indicated by overlaying a black circle and pointer, transformed as if the circle were painted on the surface at that point. (b) Recovery of shape from texture for a curved surface (white circle and pointer this time). Images courtesy of Jitendra Malik and Ruth Rosenholtz (1994).

changes rather slowly, and humans seem to be quite good at using this observation to tell whether low illumination, surface orientation, or albedo caused a surface patch to be dark. To simplify the problem, let us assume that the albedo is known at every surface point. It is still difficult to recover the normal, because the image brightness is one measurement but the normal has two unknown parameters, so we cannot simply solve for the normal. The key to this situation seems to be that nearby normals will be similar, because most surfaces are smooth—they do not have sharp changes.

The real difficulty comes in dealing with interreflections. If we consider a typical indoor scene, such as the objects inside an office, surfaces are illuminated not only by the light sources, but also by the light reflected from other surfaces in the scene that effectively serve as secondary light sources. These mutual illumination effects are quite significant and make it quite difficult to predict the relationship between the normal and the image brightness. Two surface patches with the same normal might have quite different brightnesses, because one receives light reflected from a large white wall and the other faces only a dark bookcase. Despite these difficulties, the problem is important. Humans seem to be able to ignore the effects of interreflections and get a useful perception of shape from shading, but we know frustratingly little about algorithms to do this.

24.4.6 Contour

When we look at a line drawing, such as Figure 24.21, we get a vivid perception of three-dimensional shape and layout. How? It is a combination of recognition of familiar objects in the scene and the application of generic constraints such as the following:

- Occluding contours, such as the outlines of the hills. One side of the contour is nearer to the viewer, the other side is farther away. Features such as local convexity and sym-

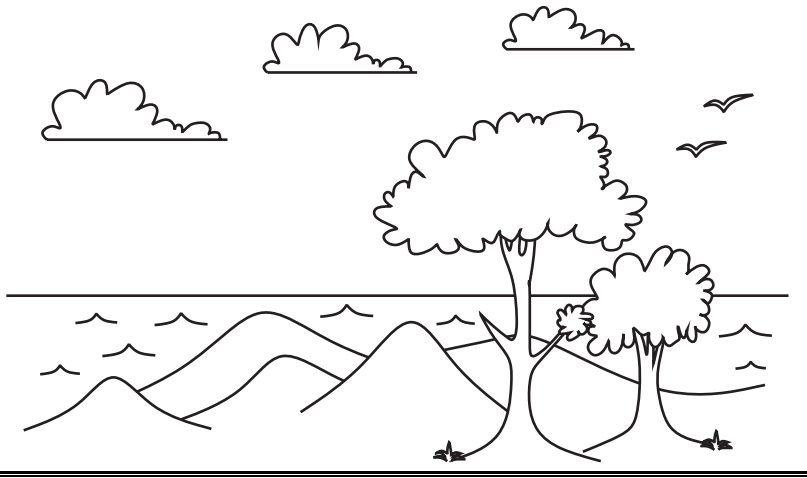


Figure 24.21 An evocative line drawing. (Courtesy of Isha Malik.)

FIGURE-GROUND

metry provide cues to solving the **figure-ground** problem—assigning which side of the contour is figure (nearer), and which is ground (farther). At an occluding contour, the line of sight is tangential to the surface in the scene.

- T-junctions. When one object occludes another, the contour of the farther object is interrupted, assuming that the nearer object is opaque. A T-junction results in the image.
- Position on the ground plane. Humans, like many other terrestrial animals, are very often in a scene that contains a **ground plane**, with various objects at different locations on this plane. Because of gravity, typical objects don't float in air but are supported by this ground plane, and we can exploit the very special geometry of this viewing scenario.

GROUND PLANE

Let us work out the projection of objects of different heights and at different locations on the ground plane. Suppose that the eye, or camera, is at a height h_c above the ground plane. Consider an object of height δY resting on the ground plane, whose bottom is at $(X, -h_c, Z)$ and top is at $(X, \delta Y - h_c, Z)$. The bottom projects to the image point $(fX/Z, -fh_c/Z)$ and the top to $(fX/Z, f(\delta Y - h_c)/Z)$. The bottoms of nearer objects (small Z) project to points lower in the image plane; farther objects have bottoms closer to the horizon.

24.4.7 Objects and the geometric structure of scenes

A typical adult human head is about 9 inches long. This means that for someone who is 43 feet away, the angle subtended by the head at the camera is 1 degree. If we see a person whose head appears to subtend just half a degree, Bayesian inference suggests we are looking at a normal person who is 86 feet away, rather than someone with a half-size head. This line of reasoning supplies us with a method to check the results of a pedestrian detector, as well as a method to estimate the distance to an object. For example, all pedestrians are about the same height, and they tend to stand on a ground plane. If we know where the horizon is in an image, we can rank pedestrians by distance to the camera. This works because we know where their

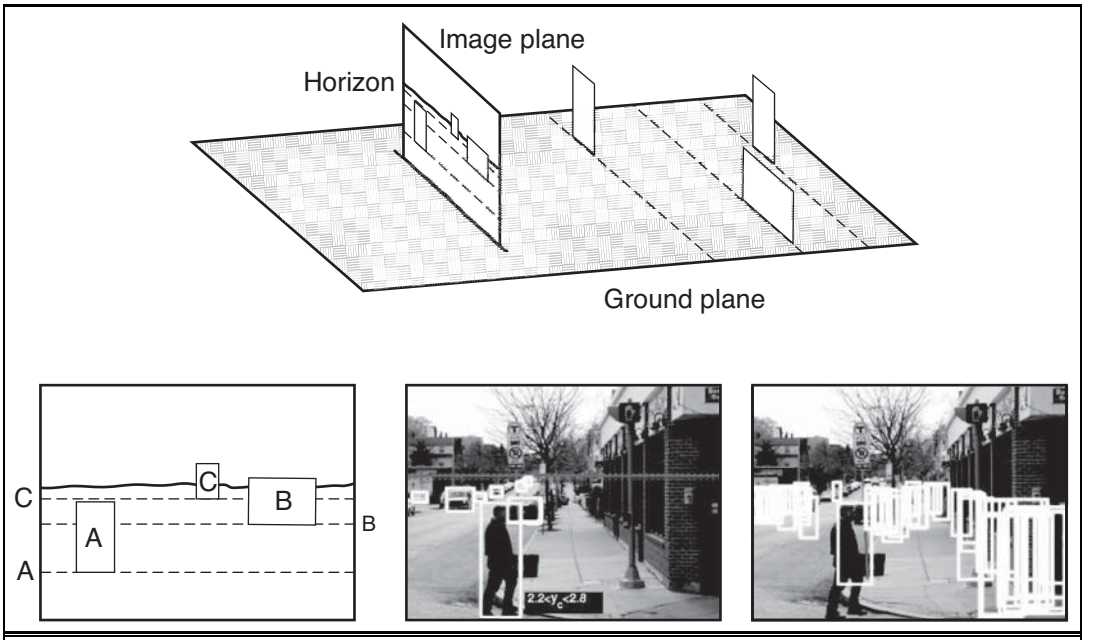


Figure 24.22 In an image of people standing on a ground plane, the people whose feet are closer to the horizon in the image must be farther away (top drawing). This means they must look smaller in the image (left lower drawing). This means that the size and location of real pedestrians in an image depend upon one another and on the location of the horizon. To exploit this, we need to identify the ground plane, which is done using shape-from-texture methods. From this information, and from some likely pedestrians, we can recover a horizon as shown in the center image. On the right, acceptable pedestrian boxes given this geometric context. Notice that pedestrians who are higher in the scene must be smaller. If they are not, then they are false positives. Images from Hoiem *et al.* (2008) © IEEE.

feet are, and pedestrians whose feet are closer to the horizon in the image are farther away from the camera (Figure 24.22). Pedestrians who are farther away from the camera must also be smaller in the image. This means we can rule out some detector responses — if a detector finds a pedestrian who is large in the image and whose feet are close to the horizon, it has found an enormous pedestrian; these don't exist, so the detector is wrong. In fact, many or most image windows are not acceptable pedestrian windows, and need not even be presented to the detector.

There are several strategies for finding the horizon, including searching for a roughly horizontal line with a lot of blue above it, and using surface orientation estimates obtained from texture deformation. A more elegant strategy exploits the reverse of our geometric constraints. A reasonably reliable pedestrian detector is capable of producing estimates of the horizon, if there are several pedestrians in the scene at different distances from the camera. This is because the relative scaling of the pedestrians is a cue to where the horizon is. So we can extract a horizon estimate from the detector, then use this estimate to prune the pedestrian detector's mistakes.

If the object is familiar, we can estimate more than just the distance to it, because what it looks like in the image depends very strongly on its pose, i.e., its position and orientation with respect to the viewer. This has many applications. For instance, in an industrial manipulation task, the robot arm cannot pick up an object until the pose is known. In the case of rigid objects, whether three-dimensional or two-dimensional, this problem has a simple and well-defined solution based on the **alignment method**, which we now develop.

ALIGNMENT METHOD

The object is represented by M features or distinguished points $m_1, m_2, \ldots, m_M$ in three-dimensional space—perhaps the vertices of a polyhedral object. These are measured in some coordinate system that is natural for the object. The points are then subjected to an unknown three-dimensional rotation $\mathbf{R}$, followed by translation by an unknown amount $\mathbf{t}$ and then projection to give rise to image feature points $p_1, p_2, \ldots, p_N$ on the image plane. In general, $N \neq M$, because some model points may be occluded, and the feature detector could miss some features (or invent false ones due to noise). We can express this as

$$p_i = \Pi(\mathbf{R}m_i + \mathbf{t}) = Q(m_i)$$

for a three-dimensional model point m_i and the corresponding image point p_i. Here, $\mathbf{R}$ is a rotation matrix, $\mathbf{t}$ is a translation, and Π denotes perspective projection or one of its approximations, such as scaled orthographic projection. The net result is a transformation Q that will bring the model point m_i into alignment with the image point p_i. Although we do not know Q initially, we do know (for rigid objects) that Q must be the *same* for all the model points.

We can solve for Q, given the three-dimensional coordinates of three model points and their two-dimensional projections. The intuition is as follows: we can write down equations relating the coordinates of p_i to those of m_i. In these equations, the unknown quantities correspond to the parameters of the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$. If we have enough equations, we ought to be able to solve for Q. We will not give a proof here; we merely state the following result:

> Given three noncollinear points m_1, m_2, and m_3 in the model, and their scaled orthographic projections p_1, p_2, and p_3 on the image plane, there exist exactly two transformations from the three-dimensional model coordinate frame to a two-dimensional image coordinate frame.

These transformations are related by a reflection around the image plane and can be computed by a simple closed-form solution. If we could identify the corresponding model features for three features in the image, we could compute Q, the pose of the object.

Let us specify position and orientation in mathematical terms. The position of a point P in the scene is characterized by three numbers, the (X, Y, Z) coordinates of P in a coordinate frame with its origin at the pinhole and the Z-axis along the optical axis (Figure 24.2 on page 931). What we have available is the perspective projection (x, y) of the point in the image. This specifies the ray from the pinhole along which P lies; what we do not know is the distance. The term "orientation" could be used in two senses:

1. **The orientation of the object as a whole.** This can be specified in terms of a three-dimensional rotation relating its coordinate frame to that of the camera.

2. **The orientation of the surface of the object at P.** This can be specified by a normal vector, **n**—which is a vector specifying the direction that is perpendicular to the surface. Often we express the surface orientation using the variables **slant** and **tilt**. Slant is the angle between the Z-axis and **n**. Tilt is the angle between the X-axis and the projection of **n** on the image plane.

SLANT
TILT

When the camera moves relative to an object, both the object's distance and its orientation change. What is preserved is the **shape** of the object. If the object is a cube, that fact is not changed when the object moves. Geometers have been attempting to formalize shape for centuries, the basic concept being that shape is what remains unchanged under some group of transformations—for example, combinations of rotations and translations. The difficulty lies in finding a representation of global shape that is general enough to deal with the wide variety of objects in the real world—not just simple forms like cylinders, cones, and spheres—and yet can be recovered easily from the visual input. The problem of characterizing the *local* shape of a surface is much better understood. Essentially, this can be done in terms of curvature: how does the surface normal change as one moves in different directions on the surface? For a plane, there is no change at all. For a cylinder, if one moves parallel to the axis, there is no change, but in the perpendicular direction, the surface normal rotates at a rate inversely proportional to the radius of the cylinder, and so on. All this is studied in the subject called differential geometry.

SHAPE

The shape of an object is relevant for some manipulation tasks (e.g., deciding where to grasp an object), but its most significant role is in object recognition, where geometric shape along with color and texture provide the most significant cues to enable us to identify objects, classify what is in the image as an example of some class one has seen before, and so on.

24.5 Object Recognition from Structural Information

Putting a box around pedestrians in an image may well be enough to avoid driving into them. We have seen that we can find a box by pooling the evidence provided by orientations, using histogram methods to suppress potentially confusing spatial detail. If we want to know more about what someone is doing, we will need to know where their arms, legs, body, and head lie in the picture. Individual body parts are quite difficult to detect on their own using a moving window method, because their color and texture can vary widely and because they are usually small in images. Often, forearms and shins are as small as two to three pixels wide. Body parts do not usually appear on their own, and representing what is connected to what could be quite powerful, because parts that are easy to find might tell us where to look for parts that are small and hard to detect.

Inferring the layout of human bodies in pictures is an important task in vision, because the layout of the body often reveals what people are doing. A model called a **deformable template** can tell us which configurations are acceptable: the elbow can bend but the head is never joined to the foot. The simplest deformable template model of a person connects lower arms to upper arms, upper arms to the torso, and so on. There are richer models: for example,

DEFORMABLE TEMPLATE

we could represent the fact that left and right upper arms tend to have the same color and texture, as do left and right legs. These richer models remain difficult to work with, however.

24.5.1 The geometry of bodies: Finding arms and legs

For the moment, we assume that we know what the person's body parts look like (e.g., we know the color and texture of the person's clothing). We can model the geometry of the body as a tree of eleven segments (upper and lower left and right arms and legs respectively, a torso, a face, and hair on top of the face) each of which is rectangular. We assume that the position and orientation (**pose**) of the left lower arm is independent of all other segments given the pose of the left upper arm; that the pose of the left upper arm is independent of all segments given the pose of the torso; and extend these assumptions in the obvious way to include the right arm and the legs, the face, and the hair. Such models are often called "cardboard people" models. The model forms a tree, which is usually rooted at the torso. We will search the image for the best match to this cardboard person using inference methods for a tree-structured Bayes net (see Chapter 14).

POSE

There are two criteria for evaluating a configuration. First, an image rectangle should look like its segment. For the moment, we will remain vague about precisely what that means, but we assume we have a function ϕ_i that scores how well an image rectangle matches a body segment. For each pair of related segments, we have another function ψ that scores how well relations between a pair of image rectangles match those to be expected from the body segments. The dependencies between segments form a tree, so each segment has only one parent, and we could write $\psi_{i,\text{pa}(i)}$. All the functions will be larger if the match is better, so we can think of them as being like a log probability. The cost of a particular match that allocates image rectangle m_i to body segment i is then

$$\sum_{i \in \text{segments}} \phi_i(m_i) + \sum_{i \in \text{segments}} \psi_{i,\text{pa}(i)}(m_i, m_{\text{pa}(i)}) \; .$$

Dynamic programming can find the best match, because the relational model is a tree.

It is inconvenient to search a continuous space, and we will discretize the space of image rectangles. We do so by discretizing the location and orientation of rectangles of fixed size (the sizes may be different for different segments). Because ankles and knees are different, we need to distinguish between a rectangle and the same rectangle rotated by $180\,^\circ$. One could visualize the result as a set of very large stacks of small rectangles of image, cut out at different locations and orientations. There is one stack per segment. We must now find the best allocation of rectangles to segments. This will be slow, because there are many image rectangles and, for the model we have given, choosing the right torso will be $O(M^6)$ if there are M image rectangles. However, various speedups are available for an appropriate choice of ψ, and the method is practical (Figure 24.23). The model is usually known as a **pictorial structure model**.

PICTORIAL STRUCTURE MODEL

Recall our assumption that we know what we need to know about what the person looks like. If we are matching a person in a single image, the most useful feature for scoring segment matches turns out to be color. Texture features don't work well in most cases, because folds on loose clothing produce strong shading patterns that overlay the image texture. These

Figure 24.23 A pictorial structure model evaluates a match between a set of image rectangles and a cardboard person (shown on the left) by scoring the similarity in appearance between body segments and image segments and the spatial relations between the image segments. Generally, a match is better if the image segments have about the right appearance and are in about the right place with respect to one another. The appearance model uses average colors for hair, head, torso, and upper and lower arms and legs. The relevant relations are shown as arrows. On the right, the best match for a particular image, obtained using dynamic programming. The match is a fair estimate of the configuration of the body. Figure from Felzenszwalb and Huttenlocher (2000) © IEEE.

patterns are strong enough to disrupt the true texture of the cloth. In current work, ψ typically reflects the need for the ends of the segments to be reasonably close together, but there are usually no constraints on the angles. Generally, we don't know what a person looks like, and must build a model of segment appearances. We call the description of what a person looks like the **appearance model**. If we must report the configuration of a person in a single image, we can start with a poorly tuned appearance model, estimate configuration with this, then re-estimate appearance, and so on. In video, we have many frames of the same person, and this will reveal their appearance.

APPEARANCE MODEL

24.5.2 Coherent appearance: Tracking people in video

Tracking people in video is an important practical problem. If we could reliably report the location of arms, legs, torso, and head in video sequences, we could build much improved game interfaces and surveillance systems. Filtering methods have not had much success with this problem, because people can produce large accelerations and move quite fast. This means that for 30 Hz video, the configuration of the body in frame i doesn't constrain the configuration of the body in frame $i+1$ all that strongly. Currently, the most effective methods exploit the fact that appearance changes very slowly from frame to frame. If we can infer an appearance model of an individual from the video, then we can use this information in a pictorial structure model to detect that person in each frame of the video. We can then link these locations across time to make a track.

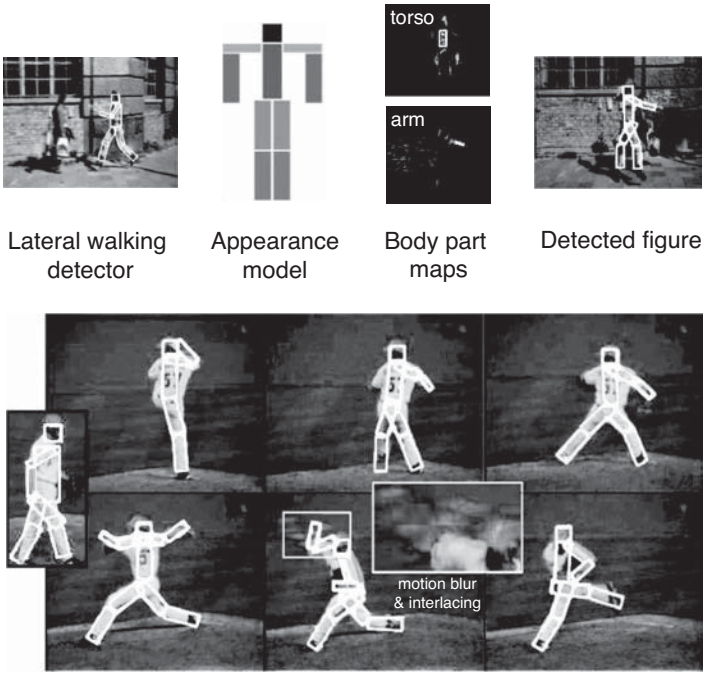


Figure 24.24 We can track moving people with a pictorial structure model by first obtaining an appearance model, then applying it. To obtain the appearance model, we scan the image to find a lateral walking pose. The detector does not need to be very accurate, but should produce few false positives. From the detector response, we can read off pixels that lie on each body segment, and others that do not lie on that segment. This makes it possible to build a discriminative model of the appearance of each body part, and these are tied together into a pictorial structure model of the person being tracked. Finally, we can reliably track by detecting this model in each frame. As the frames in the lower part of the image suggest, this procedure can track complicated, fast-changing body configurations, despite degradation of the video signal due to motion blur. Figure from Ramanan *et al.* (2007) © IEEE.

There are several ways to infer a good appearance model. We regard the video as a large stack of pictures of the person we wish to track. We can exploit this stack by looking for appearance models that explain many of the pictures. This would work by detecting body segments in each frame, using the fact that segments have roughly parallel edges. Such detectors are not particularly reliable, but the segments we want to find are special. They will appear at least once in most of the frames of video; such segments can be found by clustering the detector responses. It is best to start with the torso, because it is big and because torso detectors tend to be reliable. Once we have a torso appearance model, upper leg segments should appear near the torso, and so on. This reasoning yields an appearance model, but it can be unreliable if people appear against a near-fixed background where the segment detector generates lots of false positives. An alternative is to estimate appearance for many of the frames of video by repeatedly reestimating configuration and appearance; we then see if one appearance model explains many frames. Another alternative, which is quite

Figure 24.25 Some complex human actions produce consistent patterns of appearance and motion. For example, drinking involves movements of the hand in front of the face. The first three images are correct detections of drinking; the fourth is a false-positive (the cook is looking into the coffee pot, but not drinking from it). Figure from Laptev and Perez (2007) © IEEE.

reliable in practice, is to apply a detector for a fixed body configuration to all of the frames. A good choice of configuration is one that is easy to detect reliably, and where there is a strong chance the person will appear in that configuration even in a short sequence (lateral walking is a good choice). We tune the detector to have a low false positive rate, so we know when it responds that we have found a real person; and because we have localized their torso, arms, legs, and head, we know what these segments look like.

24.6 Using Vision

If vision systems could analyze video and understood what people are doing, we would be able to: design buildings and public places better by collecting and using data about what people do in public; build more accurate, more secure, and less intrusive surveillance systems; build computer sports commentators; and build human-computer interfaces that watch people and react to their behavior. Applications for reactive interfaces range from computer games that make a player get up and move around to systems that save energy by managing heat and light in a building to match where the occupants are and what they are doing.

Some problems are well understood. If people are relatively small in the video frame, and the background is stable, it is easy to detect the people by subtracting a background image from the current frame. If the absolute value of the difference is large, this **background subtraction** declares the pixel to be a foreground pixel; by linking foreground blobs over time, we obtain a track.

BACKGROUND SUBTRACTION

Structured behaviors like ballet, gymnastics, or tai chi have specific vocabularies of actions. When performed against a simple background, videos of these actions are easy to deal with. Background subtraction identifies the major moving regions, and we can build HOG features (keeping track of flow rather than orientation) to present to a classifier. We can detect consistent patterns of action with a variant of our pedestrian detector, where the orientation features are collected into histogram buckets over time as well as space (Figure 24.25).

More general problems remain open. The big research question is to link observations of the body and the objects nearby to the goals and intentions of the moving people. One source of difficulty is that we lack a simple vocabulary of human behavior. Behavior is a lot

like color, in that people tend to think they know a lot of behavior names but can't produce long lists of such words on demand. There is quite a lot of evidence that behaviors combine—you can, for example, drink a milkshake while visiting an ATM—but we don't yet know what the pieces are, how the composition works, or how many composites there might be. A second source of difficulty is that we don't know what features expose what is happening. For example, knowing someone is close to an ATM may be enough to tell that they're visiting the ATM. A third difficulty is that the usual reasoning about the relationship between training and test data is untrustworthy. For example, we cannot argue that a pedestrian detector is safe simply because it performs well on a large data set, because that data set may well omit important, but rare, phenomena (for example, people mounting bicycles). We wouldn't want our automated driver to run over a pedestrian who happened to do something unusual.

24.6.1 Words and pictures

Many Web sites offer collections of images for viewing. How can we find the images we want? Let's suppose the user enters a text query, such as "bicycle race." Some of the images will have keywords or captions attached, or will come from Web pages that contain text near the image. For these, image retrieval can be like text retrieval: ignore the images and match the image's text against the query (see Section 22.3 on page 867).

However, keywords are usually incomplete. For example, a picture of a cat playing in the street might be tagged with words like "cat" and "street," but it is easy to forget to mention the "garbage can" or the "fish bones." Thus an interesting task is to annotate an image (which may already have a few keywords) with additional appropriate keywords.

In the most straightforward version of this task, we have a set of correctly tagged example images, and we wish to tag some test images. This problem is sometimes known as auto-annotation. The most accurate solutions are obtained using nearest-neighbors methods. One finds the training images that are closest to the test image in a feature space metric that is trained using examples, then reports their tags.

Another version of the problem involves predicting which tags to attach to which regions in a test image. Here we do not know which regions produced which tags for the training data. We can use a version of expectation maximization to guess an initial correspondence between text and regions, and from that estimate a better decomposition into regions, and so on.

24.6.2 Reconstruction from many views

Binocular stereopsis works because for each point we have four measurements constraining three unknown degrees of freedom. The four measurements are the (x, y) positions of the point in each view, and the unknown degrees of freedom are the (x, y, z) coordinate values of the point in the scene. This rather crude argument suggests, correctly, that there are geometric constraints that prevent most pairs of points from being acceptable matches. Many images of a set of points should reveal their positions unambiguously.

We don't always need a second picture to get a second view of a set of points. If we believe the original set of points comes from a familiar rigid 3D object, then we might have

an object model available as a source of information. If this object model consists of a set of 3D points or of a set of pictures of the object, and if we can establish point correspondences, we can determine the parameters of the camera that produced the points in the original image. This is very powerful information. We could use it to evaluate our original hypothesis that the points come from an object model. We do this by using some points to determine the parameters of the camera, then projecting model points in this camera and checking to see whether there are image points nearby.

We have sketched here a technology that is now very highly developed. The technology can be generalized to deal with views that are not orthographic; to deal with points that are observed in only some views; to deal with unknown camera properties like focal length; to exploit various sophisticated searches for appropriate correspondences; and to do reconstruction from very large numbers of points and of views. If the locations of points in the images are known with some accuracy and the viewing directions are reasonable, very high accuracy camera and point information can be obtained. Some applications are

- **Model-building:** For example, one might build a modeling system that takes a video sequence depicting an object and produces a very detailed three-dimensional mesh of textured polygons for use in computer graphics and virtual reality applications. Models like this can now be built from apparently quite unpromising sets of pictures. For example, Figure 24.26 shows a model of the Statue of Liberty built from pictures found on the Internet.
- **Matching moves:** To place computer graphics characters into real video, we need to know how the camera moved for the real video, so that we can render the character correctly.
- **Path reconstruction:** Mobile robots need to know where they have been. If they are moving in a world of rigid objects, then performing a reconstruction and keeping the camera information is one way to obtain a path.

24.6.3 Using vision for controlling movement

One of the principal uses of vision is to provide information both for manipulating objects—picking them up, grasping them, twirling them, and so on—and for navigating while avoiding obstacles. The ability to use vision for these purposes is present in the most primitive of animal visual systems. In many cases, the visual system is minimal, in the sense that it extracts from the available light field just the information the animal needs to inform its behavior. Quite probably, modern vision systems evolved from early, primitive organisms that used a photosensitive spot at one end to orient themselves toward (or away from) the light. We saw in Section 24.4 that flies use a very simple optical flow detection system to land on walls. A classic study, *What the Frog's Eye Tells the Frog's Brain* (Lettvin *et al.*, 1959), observes of a frog that, "He will starve to death surrounded by food if it is not moving. His choice of food is determined only by size and movement."

Let us consider a vision system for an automated vehicle driving on a freeway. The tasks faced by the driver include the following:

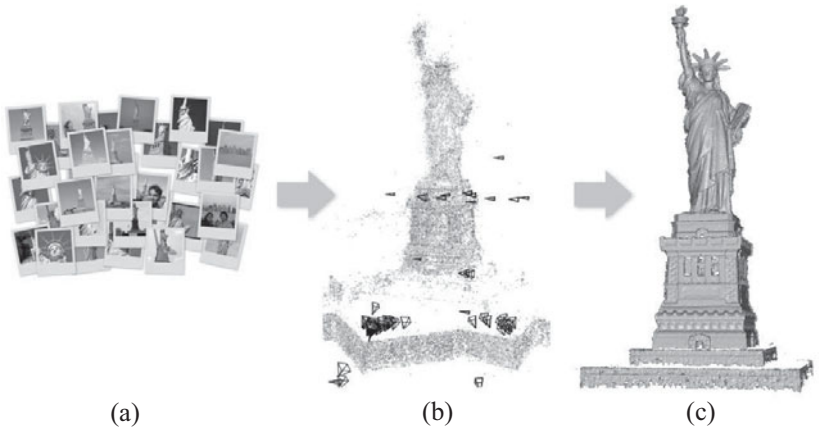


Figure 24.26 The state of the art in multiple-view reconstruction is now highly advanced. This figure outlines a system built by Michael Goesele and colleagues from the University of Washington, TU Darmstadt, and Microsoft Research. From a collection of pictures of a monument taken by a large community of users and posted on the Internet (a), their system can determine the viewing directions for those pictures, shown by the small black pyramids in (b) and a comprehensive 3D reconstruction shown in (c).

1. Lateral control—ensure that the vehicle remains securely within its lane or changes lanes smoothly when required.
2. Longitudinal control—ensure that there is a safe distance to the vehicle in front.
3. Obstacle avoidance—monitor vehicles in neighboring lanes and be prepared for evasive maneuvers if one of them decides to change lanes.

The problem for the driver is to generate appropriate steering, acceleration, and braking actions to best accomplish these tasks.

For lateral control, one needs to maintain a representation of the position and orientation of the car relative to the lane. We can use edge-detection algorithms to find edges corresponding to the lane-marker segments. We can then fit smooth curves to these edge elements. The parameters of these curves carry information about the lateral position of the car, the direction it is pointing relative to the lane, and the curvature of the lane. This information, along with information about the dynamics of the car, is all that is needed by the steering-control system. If we have good detailed maps of the road, then the vision system serves to confirm our position (and to watch for obstacles that are not on the map).

For longitudinal control, one needs to know distances to the vehicles in front. This can be accomplished with binocular stereopsis or optical flow. Using these techniques, vision-controlled cars can now drive reliably at highway speeds.

The more general case of mobile robots navigating in various indoor and outdoor environments has been studied, too. One particular problem, localizing the robot in its environment, now has pretty good solutions. A group at Sarnoff has developed a system based on two cameras looking forward that track feature points in 3D and use that to reconstruct the

position of the robot relative to the environment. In fact, they have two stereoscopic camera systems, one looking front and one looking back—this gives greater robustness in case the robot has to go through a featureless patch due to dark shadows, blank walls, and the like. It is unlikely that there are no features either in the front or in the back. Now of course, that could happen, so a backup is provided by using an inertial motion unit (IMU) somewhat akin to the mechanisms for sensing acceleration that we humans have in our inner ears. By integrating the sensed acceleration twice, one can keep track of the change in position. Combining the data from vision and the IMU is a problem of probabilistic evidence fusion and can be tackled using techniques, such as Kalman filtering, we have studied elsewhere in the book.

In the use of visual odometry (estimation of change in position), as in other problems of odometry, there is the problem of "drift," positional errors accumulating over time. The solution for this is to use landmarks to provide absolute position fixes: as soon as the robot passes a location in its internal map, it can adjust its estimate of its position appropriately. Accuracies on the order of centimeters have been demonstrated with the these techniques.

The driving example makes one point very clear: *for a specific task, one does not need to recover all the information that, in principle, can be recovered from an image.* One does not need to recover the exact shape of every vehicle, solve for shape-from-texture on the grass surface adjacent to the freeway, and so on. Instead, a vision system should compute just what is needed to accomplish the task.

24.7 SUMMARY

Although perception appears to be an effortless activity for humans, it requires a significant amount of sophisticated computation. The goal of vision is to extract information needed for tasks such as manipulation, navigation, and object recognition.

- The process of **image formation** is well understood in its geometric and physical aspects. Given a description of a three-dimensional scene, we can easily produce a picture of it from some arbitrary camera position (the graphics problem). Inverting the process by going from an image to a description of the scene is more difficult.
- To extract the visual information necessary for the tasks of manipulation, navigation, and recognition, intermediate representations have to be constructed. Early vision **image-processing** algorithms extract primitive features from the image, such as edges and regions.
- There are various cues in the image that enable one to obtain three-dimensional information about the scene: motion, stereopsis, texture, shading, and contour analysis. Each of these cues relies on background assumptions about physical scenes to provide nearly unambiguous interpretations.
- Object recognition in its full generality is a very hard problem. We discussed brightness-based and feature-based approaches. We also presented a simple algorithm for pose estimation. Other possibilities exist.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The eye developed in the Cambrian explosion (530 million years ago), apparently in a common ancestor. Since then, endless variations have developed in different creatures, but the same gene, Pax-6, regulates the development of the eye in animals as diverse as humans, mice, and *Drosophila*.

Systematic attempts to understand human vision can be traced back to ancient times. Euclid (ca. 300 B.C.) wrote about natural perspective—the mapping that associates, with each point P in the three-dimensional world, the direction of the ray OP joining the center of projection O to the point P. He was well aware of the notion of motion parallax. The use of perspective in art was developed in ancient Roman culture, as evidenced by art found in the ruins of Pompeii (A.D. 79), but was then largely lost for 1300 years. The mathematical understanding of perspective projection, this time in the context of projection onto planar surfaces, had its next significant advance in the 15th-century in Renaissance Italy. Brunelleschi (1413) is usually credited with creating the first paintings based on geometrically correct projection of a three-dimensional scene. In 1435, Alberti codified the rules and inspired generations of artists whose artistic achievements amaze us to this day. Particularly notable in their development of the science of perspective, as it was called in those days, were Leonardo da Vinci and Albrecht Dürer. Leonardo's late 15th century descriptions of the interplay of light and shade (chiaroscuro), umbra and penumbra regions of shadows, and aerial perspective are still worth reading in translation (Kemp, 1989). Stork (2004) analyzes the creation of various pieces of Renaissance art using computer vision techniques.

Although perspective was known to the ancient Greeks, they were curiously confused by the role of the eyes in vision. Aristotle thought of the eyes as devices emitting rays, rather in the manner of modern laser range finders. This mistaken view was laid to rest by the work of Arab scientists, such as Abu Ali Alhazen, in the 10th century. Alhazen also developed the *camera obscura*, a room (*camera* is Latin for "room" or "chamber") with a pinhole that casts an image on the opposite wall. Of course the image was inverted, which caused no end of confusion. If the eye was to be thought of as such an imaging device, how do we see right-side up? This enigma exercised the greatest minds of the era (including Leonardo). Kepler first proposed that the lens of the eye focuses an image on the retina, and Descartes surgically removed an ox eye and demonstrated that Kepler was right. There was still puzzlement as to why we do not see everything upside down; today we realize it is just a question of accessing the retinal data structure in the right way.

In the first half of the 20th century, the most significant research results in vision were obtained by the Gestalt school of psychology, led by Max Wertheimer. They pointed out the importance of perceptual organization: for a human observer, the image is not a collection of pointillist photoreceptor outputs (pixels in computer vision terminology); rather it is organized into coherent groups. One could trace the motivation in computer vision of finding regions and curves back to this insight. The Gestaltists also drew attention to the "figure–ground" phenomenon—a contour separating two image regions that, in the world, are at different depths, appears to belong only to the nearer region, the "figure," and not the farther

region, the "ground." The computer vision problem of classifying image curves according to their significance in the scene can be thought of as a generalization of this insight.

The period after World War II was marked by renewed activity. Most significant was the work of J. J. Gibson (1950, 1979), who pointed out the importance of optical flow, as well as texture gradients in the estimation of environmental variables such as surface slant and tilt. He reemphasized the importance of the stimulus and how rich it was. Gibson emphasized the role of the active observer whose self-directed movement facilitates the pickup of information about the external environment.

Computer vision was founded in the 1960s. Roberts's (1963) thesis at MIT was one of the earliest publications in the field, introducing key ideas such as edge detection and model-based matching. There is an urban legend that Marvin Minsky assigned the problem of "solving" computer vision to a graduate student as a summer project. According to Minsky the legend is untrue—it was actually an undergraduate student. But it was an exceptional undergraduate, Gerald Jay Sussman (who is now a professor at MIT) and the task was not to "solve" vision, but to investigate some aspects of it.

In the 1960s and 1970s, progress was slow, hampered considerably by the lack of computational and storage resources. Low-level visual processing received a lot of attention. The widely used Canny edge-detection technique was introduced in Canny (1986). Techniques for finding texture boundaries based on multiscale, multiorientation filtering of images date to work such as Malik and Perona (1990). Combining multiple clues—brightness, texture and color—for finding boundary curves in a learning framework was shown by Martin, Fowlkes and Malik (2004) to considerably improve performance.

The closely related problem of finding regions of coherent brightness, color, and texture, naturally lends itself to formulations in which finding the best partition becomes an optimization problem. Three leading examples are the Markov Random Fields approach of Geman and Geman (1984), the variational formulation of Mumford and Shah (1989), and normalized cuts by Shi and Malik (2000).

Through much of the 1960s, 1970s and 1980s, there were two distinct paradigms in which visual recognition was pursued, dictated by different perspectives on what was perceived to be the primary problem. Computer vision research on object recognition largely focused on issues arising from the projection of three-dimensional objects onto two-dimensional images. The idea of alignment, also first introduced by Roberts, resurfaced in the 1980s in the work of Lowe (1987) and Huttenlocher and Ullman (1990). Also popular was an approach based on describing shapes in terms of volumetric primitives, with **generalized cylinders**, introduced by Tom Binford (1971), proving particularly popular.

GENERALIZED CYLINDER

In contrast, the pattern recognition community viewed the 3D-to-2D aspects of the problem as not significant. Their motivating examples were in domains such as optical character recognition and handwritten zip code recognition where the primary concern is that of learning the typical variations characteristic of a class of objects and separating them from other classes. See LeCun *et al.* (1995) for a comparison of approaches.

In the late 1990s, these two paradigms started to converge, as both sides adopted the probabilistic modeling and learning techniques that were becoming popular throughout AI. Two lines of work contributed significantly. One was research on face detection, such as that

of Rowley, Baluja and Kanade (1996), and of Viola and Jones (2002b) which demonstrated the power of pattern recognition techniques on clearly important and useful tasks. The other was the development of point descriptors, which enable one to construct feature vectors from parts of objects. This was pioneered by Schmid and Mohr (1996). Lowe's (2004) SIFT descriptor is widely used. The HOG descriptor is due to Dalal and Triggs (2005).

Ullman (1979) and Longuet-Higgins (1981) are influential early works in reconstruction from multiple images. Concerns about the stability of structure from motion were significantly allayed by the work of Tomasi and Kanade (1992) who showed that with the use of multiple frames shape could be recovered quite accurately. In the 1990s, with great increase in computer speed and storage, motion analysis found many new applications. Building geometrical models of real-world scenes for rendering by computer graphics techniques proved particularly popular, led by reconstruction algorithms such as the one developed by Debevec, Taylor, and Malik (1996). The books by Hartley and Zisserman (2000) and Faugeras *et al.* (2001) provide a comprehensive treatment of the geometry of multiple views.

For single images, inferring shape from shading was first studied by Horn (1970), and Horn and Brooks (1989) present an extensive survey of the main papers from a period when this was a much-studied problem. Gibson (1950) was the first to propose texture gradients as a cue to shape, though a comprehensive analysis for curved surfaces first appears in Garding (1992) and Malik and Rosenholtz (1997). The mathematics of occluding contours, and more generally understanding the visual events in the projection of smooth curved objects, owes much to the work of Koenderink and van Doorn, which finds an extensive treatment in Koenderink's (1990) *Solid Shape*. In recent years, attention has turned to treating the problem of shape and surface recovery from a single image as a probabilistic inference problem, where geometrical cues are not modeled explicitly, but used implicitly in a learning framework. A good representative is the work of Hoiem, Efros, and Hebert (2008).

For the reader interested in human vision, Palmer (1999) provides the best comprehensive treatment; Bruce *et al.* (2003) is a shorter textbook. The books by Hubel (1988) and Rock (1984) are friendly introductions centered on neurophysiology and perception respectively. David Marr's book *Vision* (Marr, 1982) played a historical role in connecting computer vision to psychophysics and neurobiology. While many of his specific models haven't stood the test of time, the theoretical perspective from which each task is analyzed at an informational, computational, and implementation level is still illuminating.

For computer vision, the most comprehensive textbook is Forsyth and Ponce (2002). Trucco and Verri (1998) is a shorter account. Horn (1986) and Faugeras (1993) are two older and still useful textbooks.

The main journals for computer vision are IEEE *Transactions on Pattern Analysis and Machine Intelligence* and *International Journal of Computer Vision*. Computer vision conferences include ICCV (International Conference on Computer Vision), CVPR (Computer Vision and Pattern Recognition), and ECCV (European Conference on Computer Vision). Research with a machine learning component is also published in the NIPS (Neural Information Processing Systems) conference, and work on the interface with computer graphics often appears at the ACM SIGGRAPH (Special Interest Group in Graphics) conference.

EXERCISES

24.1 In the shadow of a tree with a dense, leafy canopy, one sees a number of light spots. Surprisingly, they all appear to be circular. Why? After all, the gaps between the leaves through which the sun shines are not likely to be circular.

24.2 Consider a picture of a white sphere floating in front of a black backdrop. The image curve separating white pixels from black pixels is sometimes called the "outline" of the sphere. Show that the outline of a sphere, viewed in a perspective camera, can be an ellipse. Why do spheres not look like ellipses to you?

24.3 Consider an infinitely long cylinder of radius r oriented with its axis along the y-axis. The cylinder has a Lambertian surface and is viewed by a camera along the positive z-axis. What will you expect to see in the image if the cylinder is illuminated by a point source at infinity located on the positive x-axis? Draw the contours of constant brightness in the projected image. Are the contours of equal brightness uniformly spaced?

24.4 Edges in an image can correspond to a variety of events in a scene. Consider Figure 24.4 (page 933), and assume that it is a picture of a real three-dimensional scene. Identify ten different brightness edges in the image, and for each, state whether it corresponds to a discontinuity in (a) depth, (b) surface orientation, (c) reflectance, or (d) illumination.

24.5 A stereoscopic system is being contemplated for terrain mapping. It will consist of two CCD cameras, each having 512×512 pixels on a 10 cm $\times$ 10 cm square sensor. The lenses to be used have a focal length of 16 cm, with the focus fixed at infinity. For corresponding points (u_1, v_1) in the left image and (u_2, v_2) in the right image, $v_1 = v_2$ because the x-axes in the two image planes are parallel to the epipolar lines—the lines from the object to the camera. The optical axes of the two cameras are parallel. The baseline between the cameras is 1 meter.

a. If the nearest distance to be measured is 16 meters, what is the largest disparity that will occur (in pixels)?

b. What is the distance resolution at 16 meters, due to the pixel spacing?

c. What distance corresponds to a disparity of one pixel?

24.6 Which of the following are true, and which are false?

a. Finding corresponding points in stereo images is the easiest phase of the stereo depth-finding process.

b. Shape-from-texture can be done by projecting a grid of light-stripes onto the scene.

c. Lines with equal lengths in the scene always project to equal lengths in the image.

d. Straight lines in the image necessarily correspond to straight lines in the scene.

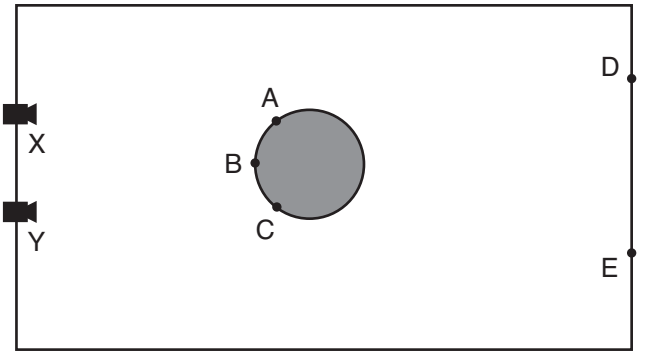


Figure 24.27 Top view of a two-camera vision system observing a bottle with a wall behind it.

24.7 (Courtesy of Pietro Perona.) Figure 24.27 shows two cameras at X and Y observing a scene. Draw the image seen at each camera, assuming that all named points are in the same horizontal plane. What can be concluded from these two images about the relative distances of points A, B, C, D, and E from the camera baseline, and on what basis?