4 Image Features

Quel naso dritto come una salita
Quegli occhi allegri da italiano in gita.[1]
Paolo Conte, *Bartali*

This and the following chapter consider the detection, location and representation of special parts of the image, called *image features*, usually corresponding to interesting elements of the scene.

Chapter Overview

Section 4.1 introduces the concept of image feature, and sketches the fundamental issues of *feature detection*, on which many computer vision algorithms are based.

Section 4.2 deals with *edges*, or contour fragments, and how to detect them. Edge detectors are the basis of the line and curve detectors presented in the next chapter.

Section 4.3 presents features which do not correspond necessarily to geometric elements of the scene, but are nevertheless useful.

Section 4.4 discusses surface features and *surface segmentation* for range images.

What You Need to Know to Understand this Chapter

- Working knowledge of Chapter 2 and 3.
- Basic concepts of signal theory.
- Eigenvalues and eigenvectors of a matrix.
- Elementary differential geometry, mainly surface curvatures (Appendix, section A.5).

[1] The nose as straight as an uphill road; The merry eyes of an Italian on holidays.

4.1 What Are Image Features?

In computer vision, the term *image feature* refers to two possible entities:

1. *a global property of an image* or part thereof, for instance the average grey level, the area in pixel *(global feature)*; or
2. *a part of the image with some special properties*, for instance a circle, a line, or a textured region in an intensity image, a planar surface in a range image *(local feature)*.

The sequence of operations of most computer vision systems begins by detecting and locating some features in the input images. In this and the following chapter, we concentrate on the second definition above, and illustrate how to detect special parts of intensity and range images like points, curves, particular structures of grey levels, or surface patches. The reason for this choice is that most algorithms in the following chapters assume that specific, local features have already been located. Here, we provide ways of doing that. Global features are indeed used in computer vision, but are less useful to solve the problems tackled by Chapters 7, 8, 9, 10 and 11. We assume therefore the following definition.

Definition: Image Features

Image features are local, meaningful, detectable parts of the image.

Meaningful means that the features are associated to interesting scene elements via the image formation process. Typical examples of meaningful features are sharp intensity variations created by the contours of the objects in the scene, or image regions with uniform grey levels, for instance images of planar surfaces. Sometimes the image features we look for are not associated obviously to any part or property of the scene, but reflect particular arrangements of image values with desirable properties, like *invariance* or ease of detectability. For instance, section 4.3 discusses an example of features which prove adequate for tracking across several images (Chapter 8). On the other hand, the number of pixels of grey level 134 makes a rather unuseful feature, as, in general, it cannot be associated to any interesting properties of the scene, as individual grey levels change with illumination and viewpoint.

Detectable means that location algorithms must exist, otherwise a particular feature is of no use! Different features are, of course, associated to different detection algorithms; these algorithms output collections of *feature descriptors*, which specify the position and other essential properties of the features found in the image. For instance, a descriptor for line features could specify the coordinates of the segment's central point, the segment's length, and its orientation. Feature descriptors are used by higher-level programs; for instance, in this book, chains of edge points (section 4.2) are used by line detectors (Chapter 5); lines, in turn, are used by calibration (Chapter 6) and recognition algorithms (Chapter 10).

☞ In 3-D computer vision, feature extraction is an intermediate step, not the goal of the system. We do not extract lines, say, just to obtain line maps; we extract lines to navigate robots in corridors, to decide whether an image contains a certain object, to calibrate the intrinsic parameters of a camera, and so on. The important corollary is that *it does not make much sense to pursue "perfect" feature extraction per se*, as the adequacy of a feature detection algorithm should be *ultimately* assessed in the context of the *complete* system.[2] Of course reasonably general performance criteria can and should be applied to test feature extraction modules independently (see section 4.2.4).

4.2 Edge Detection

4.2.1 Basics

Definition: Edges

Edge points, or simply *edges*, are pixels at or around which the image values undergo a sharp variation.

Problem Statement: Edge Detection

Given an image corrupted by acquisition noise, locate the edges most likely to be generated by scene elements, not by noise.

Figure 4.1 illustrates our definition. It shows an intensity image and the intensity profile along the scanline shown: notice how the main sharp variations correspond to significant contours.[3] Notice that image noise too causes intensity variations, which results in spurious edges; a good edge detection algorithm, or *edge detector*, should suppress most of them.

☞ The term "edge" is also used to refer to *connected chains* of edge points, that is, contour fragments. Edge points are sometimes called *edgels* (for "edge elements").

There are various reasons for our interest in edges. The contours of potentially interesting scene elements like solid objects, marks on surfaces, and shadows, all generate intensity edges. Moreover, image lines, curves and contours, which are often the basic elements for stereopsis, calibration, motion analysis and recognition, are detected from chains of edge points. Finally, line drawings are common and suggestive images for humans. Throughout, we refer to intensity images, but it makes perfect sense to apply edge detection to range images as well (see review questions).

Our next task is to make the problem more precise. Edge detection in computer vision is typically a three-step process.

[2] Incidentally, this is true of *any* intermediate module of a vision system.

[3] Image edges are commonly presented as "discontinuities in the underlying irradiance function," but it seems more accurate to speak of "sharp image variations" than "discontinuities". The reason is that the scene radiance is low-pass filtered by the optics (Chapter 2) and the resulting image brightness cannot have *real* 0-order discontinuities.

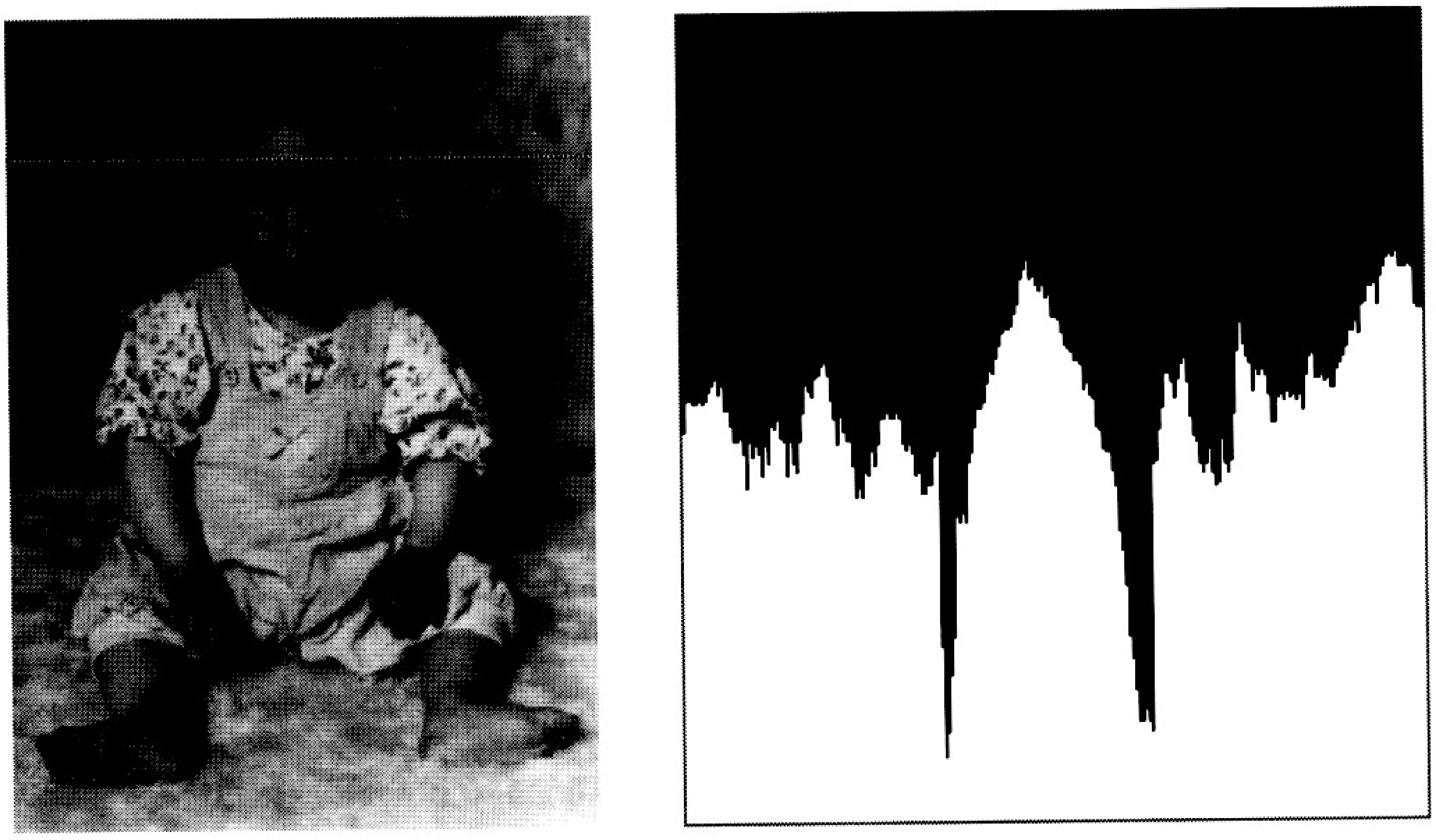


Figure 4.1 (a) A 325 × 237-pixel image, with scanline $i = 56$ highlighted. (b) The intensity profile along the highlighted scanline. Notice how the main intensity variations indicate the borders of the hair region along the scanline.

The Three Steps of Edge Detection

Noise Smoothing. Suppress as much of the image noise as possible, without destroying the true edges. In the absence of specific information, assume the noise white and Gaussian.

Edge Enhancement. Design a filter responding to edges; that is, the filter's output is large at edge pixels and low elsewhere, so that edges can be located as the local maxima in the filter's output.

Edge Localization. Decide which local maxima in the filter's output are edges and which are just caused by noise. This involves:

- thinning wide edges to 1-pixel width (*nonmaximum suppression*);
- establishing the minimum value to declare a local maxima an edge (*thresholding*).

Edge detection algorithms are found in their tens in the literature of computer vision and image processing. Many produce similar results. Instead of taking you through a plethora of algorithms, we introduce directly the *Canny edge detector*, probably the most used edge detector in today's machine vision community. Canny's detector is *optimal* in a precise, mathematical sense; going through the main ideas behind its derivation is an instructive example of good practice in the design of low-level vision algorithms. We shall also sketch two other edge detection algorithms.

4.2.2 The Canny Edge Detector

To arrive at Canny's edge detector, we need to:

1. formulate a mathematical model of edges and noise;
2. formulate quantitative performance criteria, formalizing desirable properties of the detector (e.g., good immunity to noise);
3. synthesize the best filter given models and performance criteria. We shall be looking for a linear filter (see Chapter 3), as it is easy to manipulate and implement.

Here is the skeleton of the algorithm we are going to derive. We shall build the missing algorithms in the next sections.

Algorithm CANNY_EDGE_DETECTOR

Given an image I:

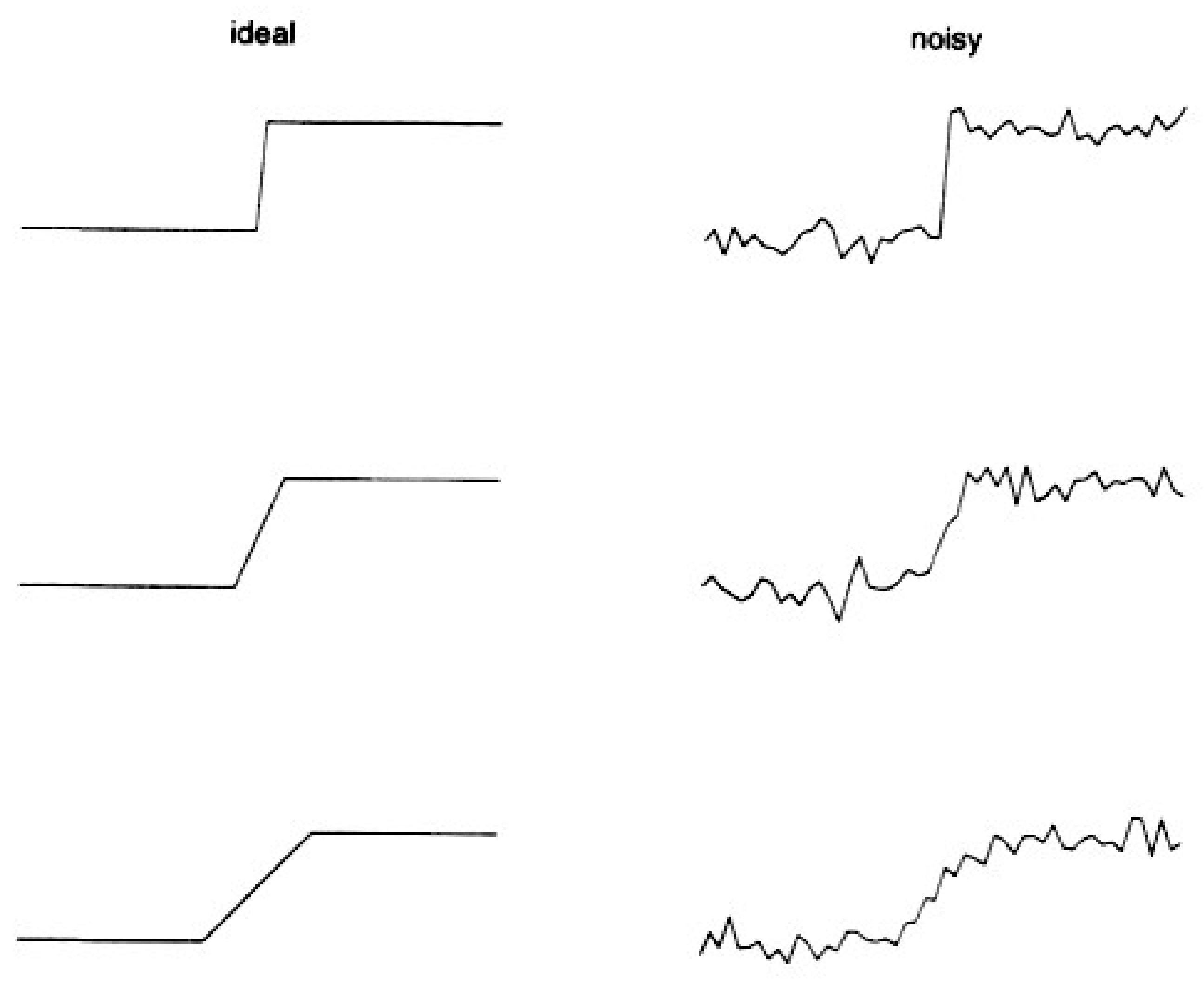
1. apply CANNY_ENHANCER to I;
2. apply NONMAX_SUPPRESSION to the output of CANNY_ENHANCER;
3. apply HYSTERESIS_THRESH to the output of NONMAX_SUPPRESSION.

Modelling Edges and Noise. Edges of intensity images can be modelled according to their *intensity profiles*. For most practical purposes, a few models are sufficient to cover all interesting edges, and these are illustrated in Figure 4.2 for the 1-D case.

☞ You should think of the 1-D signals shown in Figure 4.2 as cross-sections of 2-D images along a line of arbitrary orientation (not necessarily a row or column).

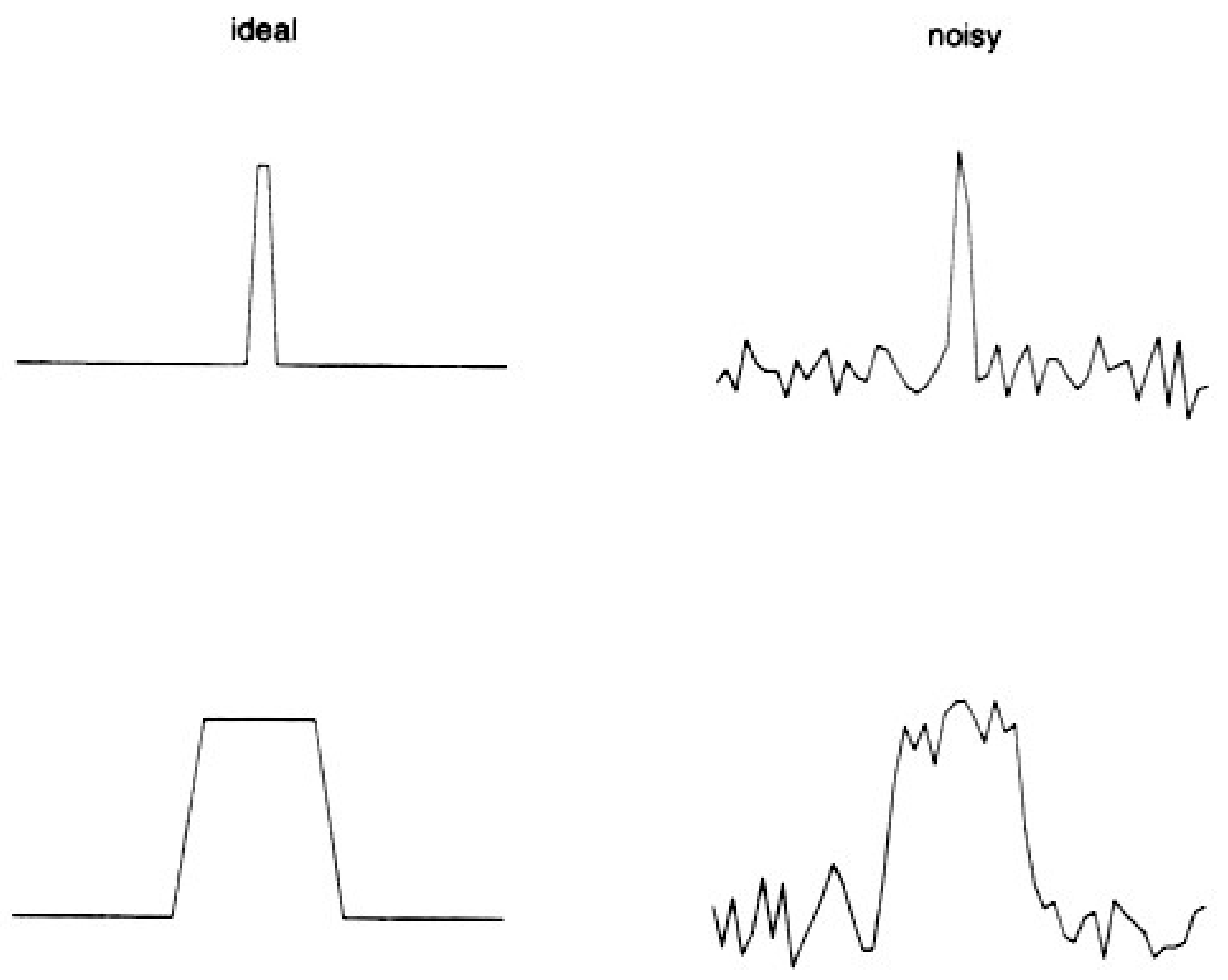
Step edges (Figure 4.2 (a)) are probably the most common type in intensity images. They occur typically at contours of image regions of different intensities. In most cases, the transition occurs over several pixels, not just one; one speaks then of *ramp edges*. *Ridge edges* (Figure 4.2 (b)) are generated by thin lines. Clearly, wide ridges can be modelled by two step edges. *Roof edges* (Figure 4.2 (c)) are relatively rare, but may appear along the intersection of surfaces. Notice that steps and ridges correspond to sharp variations of the intensity values, while roofs correspond to variations of their first derivatives.

A good step-edge detector will actually find all edges necessary for most purposes, and for this reason we concentrate on *step edge detectors*. The output we want is a list of *step edge descriptors*, each of which should include the essential properties shown in the box that follows:



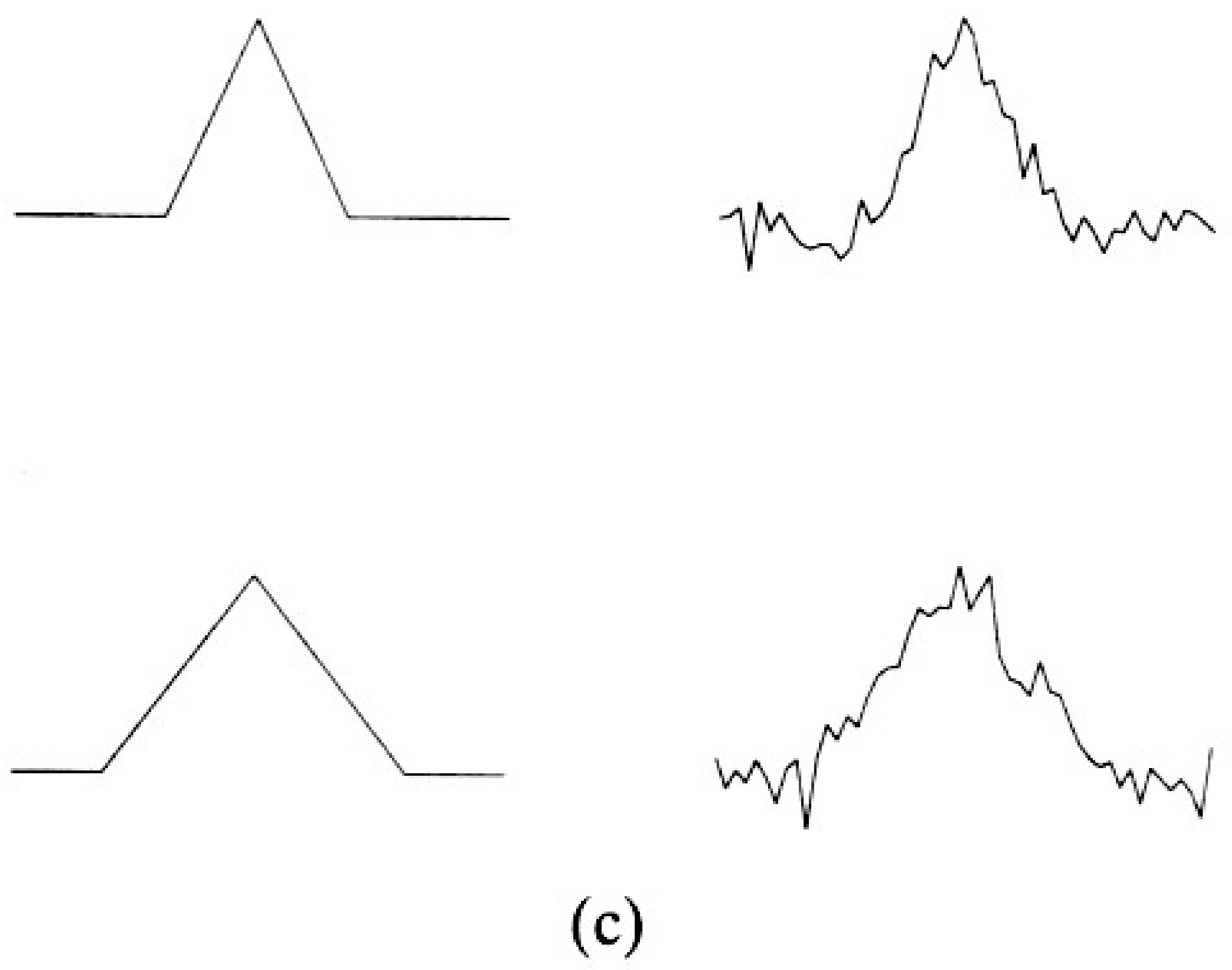
(a)

Left: ideal step (top, transition occurs over one pixel) and ramp edges. Right: corresponding noisy version, obtained by adding Gaussian noise (standard deviation 5% of the step height).



(b)

Left: ideal ridge edges. Right: corresponding noisy version, obtained as for step edges.



(c)

Left: ideal roof edges. Right: corresponding noisy version, obtained as for step edges.

Figure 4.2 Three types of 1-D edge profiles.

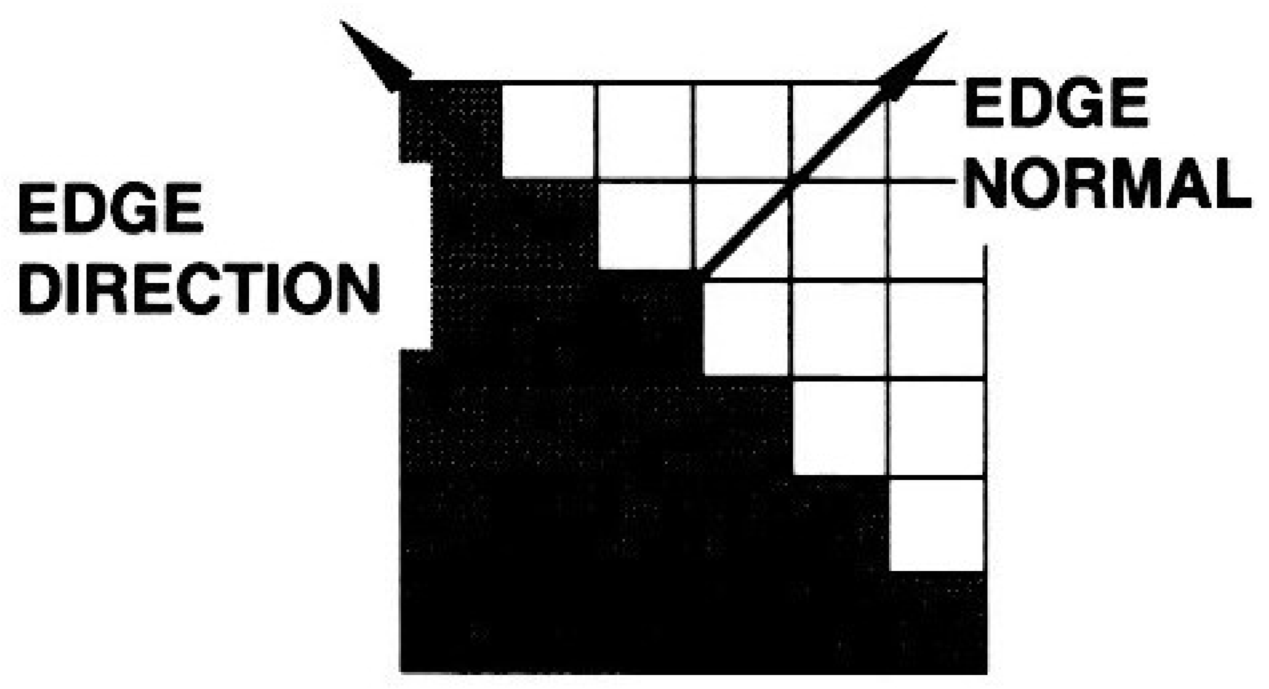


Figure 4.3 Illustration of edge normal and edge direction. The edge position considered is (2,2), with the origin in the upper left corner.

The Essential Edge Descriptor

Edge normal: for edges in 2-D images, the direction (unit vector) of the maximum intensity variation at the edge point. This identifies the direction perpendicular to the edge (Figure 4.3).

Edge direction: the direction perpendicular to the edge normal, and therefore tangent to the contour of which the edge is part (Figure 4.3). This identifies the direction tangent to the edge.

Edge position or center: the image position at which the edge is located, along the perpendicular to the edge. This is usually saved in a binary image (1 for edge, 0 for no edge).

Edge strength: a measure of the local image contrast; i.e., how marked the intensity variation is across the edge (along the normal).

We model the *ideal, 1-D step edge* as

$$G(x) = \begin{cases} 0 & x < 0 \\ A & x \geq 0 \end{cases} \tag{4.1}$$

And here is a summary of the assumptions we are making.

Assumptions

- The edge enhancement filter is linear.
- The filter must be optimal for noisy step edges.
- The image noise is additive, white and Gaussian.

Criteria for Optimal Edge Detection. We must now express the characteristics we expect of an optimal edge detector, that is, formalize optimality criteria.

Criterion 1: Good Detection. *The optimal detector must minimise the probability of false positives (detecting spurious edges caused by noise), as well as that of missing real edges.*

This is achieved by maximising the signal-to-noise ratio (SNR), defined here as the ratio of the root-mean-squared (RMS) responses of the filter to the ideal step edge (4.1) and the noise, respectively. Let f be the impulse response of the 1-D filter, and assume the filter's width is $2W$. Consider the 1-D signal in (4.1) centered in $x = 0$ (the edge's center). The response of a linear filter f to this edge is

$$\int_{-W}^{W} G(-t)f(t)dt = A\int_{-W}^{0} f(t)dt, \tag{4.2}$$

and the good-detection criterion becomes

$$SNR = \frac{A\|\int_{-W}^{0} f(t)dt\|}{n_0\sqrt{\int_{-W}^{W} f^2(t)dt}} \tag{4.3}$$

where n_0^2 is the RMS noise amplitude per unit length.[4]

Criterion 2: Good Localization. *The edges detected must be as close as possible to the true edges.*

This can be formalized by the reciprocal of the RMS distance of the detected edge from the center of the true edge. It can be proven that, in our assumptions, this results in

$$LOC = \frac{A\|f'(0)\|}{n_0\sqrt{\int_{-W}^{W} f'^2(t)dt}}. \tag{4.4}$$

We omit the rather complex derivation (see Further Readings). We now notice that (4.3) and (4.4) identify two performance measures for step edge detectors (call them Σ and Λ), *which depend only on the filter*, and not on either noise or step magnitude. In particular, Σ and Λ are defined by

$$SNR = \frac{A}{n_0}\Sigma(f) \qquad LOC = \frac{A}{n_0}\Lambda(f'). \tag{4.5}$$

The product $\Sigma\Lambda$ is a measure of how well the filter f satisfies both criteria simultaneously, so we must look for the f maximing $\Sigma\Lambda$. One can prove that this product is maximized for $f(x) = G(-x)$ in $[-W, W]$. Therefore, *the optimal, 1-D step edge detector is a simple difference operator*, or *box filter*.

Unfortunately, something is still missing. The response of a difference operator to a noisy edge contains many local maxima (Figure 4.4), not just the one we are after. We need, therefore, a third criterion, which is really a *constraint*.

Single Response Constraint. *The detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge created by noise.* Without going into details, this is formalized by imposing that the mean distance between local maxima caused by the noise (which can be estimated from the statistics of the noise) only be some fraction of the filter's half-width W.

[4] Incidentally, you have just learned (or been reminded of) the expression of the RMS response of a linear filter to white, Gaussian noise; that is, the denominator of (4.3).

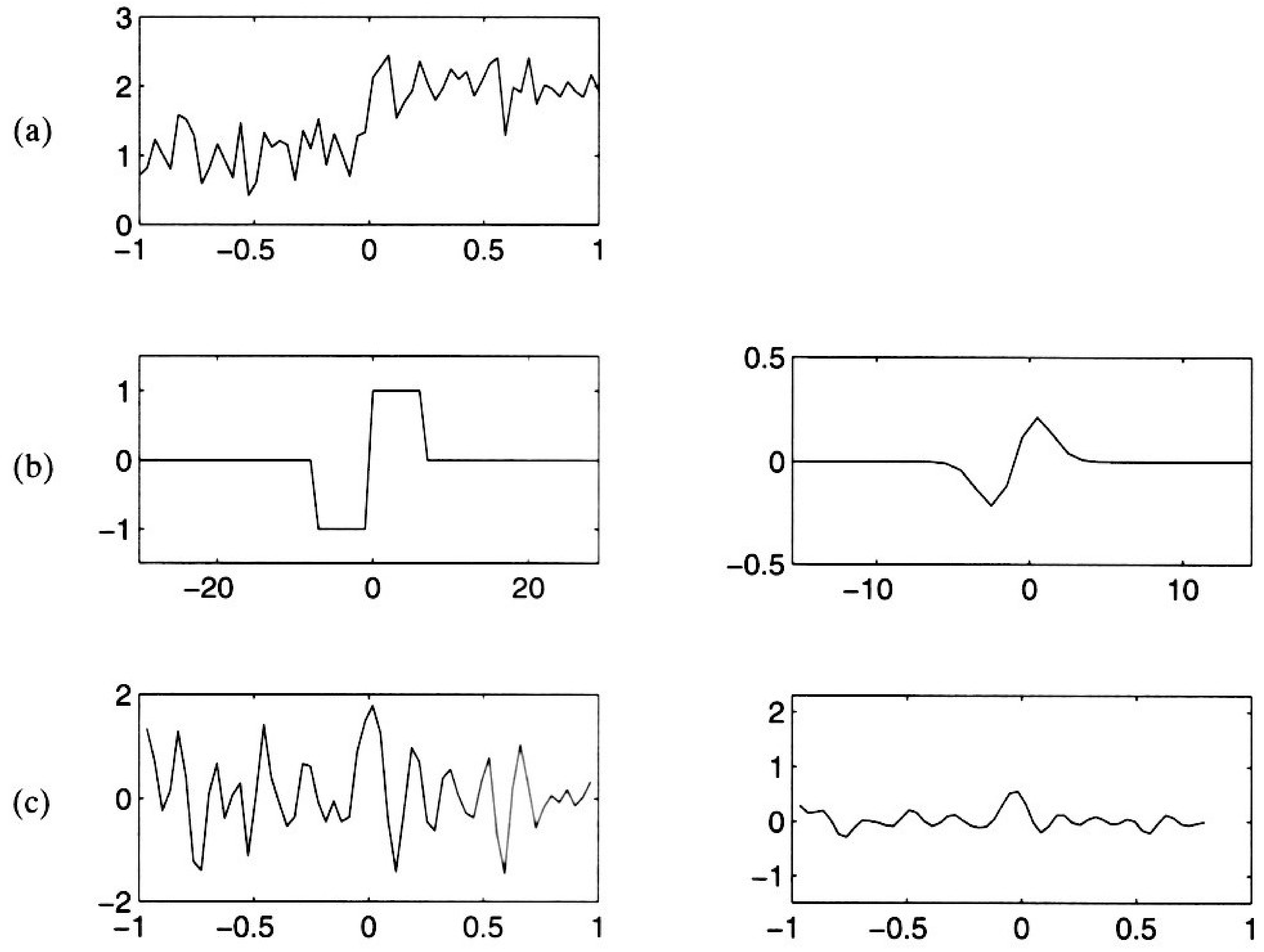


Figure 4.4 (a) Noisy, step edge, corrupted by additive Gaussian noise (standard deviation is 15% of the uncorrupted step's height). (b) The box filter (left) and the first derivative of a Gaussian (right). (c) Response to the noisy edge of the box filter (left) and of the first derivative of a Gaussian. The latter contains fewer local maxima thanks to the smoothing effect of the Gaussian. Notice the different amplitudes of the two responses due to the different amplitudes of the filters.

One consequence of our formalization is definitely worth noting. Suppose we take a spatially scaled version of the filter, that is, $f_w(x) = f(x/w)$. One can prove that

$$\Sigma(f_w) = \sqrt{w}\,\Sigma(f) \qquad \Lambda(f_w') = \frac{1}{\sqrt{w}}\Lambda(f') \tag{4.6}$$

that is, *a larger filter improves detection, but worsens localization by the same amount*, and *vice versa*. We can summarize this important result as follows.

The Localization-Detection Tradeoff

We can reach an optimal compromise between the location and detection criteria by adjusting the filter's spatial scale, but we cannot improve both criteria simultaneously.

Optimal Step Edge Enhancement. Given our edge and noise models, and the design criteria, we can obtain the optimal enhancing filter f as the function maximizing the product $\Lambda(f)\Sigma(f)$ (criteria 1 and 2), under the constraint of single response. *We have turned edge detection into a constrained optimization problem*, that is, a well-formalized mathematical problem with clear assumptions.

The bad news is that it can be proven that the solution is unique, but it is very difficult to find closed-form solutions. We can, however, evaluate numerically criteria (4.3) and (4.4) for candidate operators f, and pick the best performer.[5] In this way, one realises that *a very good approximation of the ideal step edge detector is the first derivative of a Gaussian*, which proves only 20% worse than the ideal operator with respect to (4.3) and (4.4).[6]

So far we have worked with 1-D signals, but how about 2-D images? The key to generalization is that *the 1-D treatment still applies in the direction of the edge normal*. Since we do not know *a priori* where the normal is, we should compute, at each image point, the result of applying 1-D directional filters in all possible directions. Therefore, the optimal, 2-D edge detector would involve a set of 1-D filters spanning a large number of orientations. Such filters would be implemented by narrow masks, elongated along the edge direction; the mask's shorter cross-section is the ideal 1-D detector, and the longer (in the perpendicular direction) is Gaussian, which contributes to noise smoothing.

This laborious method can be simplified significantly. One can use a *circular* Gaussian filter, which amounts to applying Gaussian smoothing to the image followed by gradient estimation, and use the gradient to estimate all the directional derivatives required. This is less accurate than applying directional filters but adequate for most practical cases. In this way, we estimate the edge strength as

$$s(i,j) = \|\nabla(G * I)\| \tag{4.7}$$

and the edge normal as

$$\mathbf{n} = \frac{\nabla(G * I)}{\|\nabla(G * I)\|} \tag{4.8}$$

Here is an algorithm implementing a practical approximation of the optimal step edge enhancer.[7]

Algorithm CANNY_ENHANCER

The input is I, an intensity image corrupted by noise. Let G be a Gaussian with zero mean and standard deviation σ.

[5] This can be done for any edge class, not only step edges. All integrals evaluated using the step edge model change; see Canny's original article (Further Readings) for details.

[6] And 90% of the single-response criterion, although this figure is rather complicated to achieve.

[7] Notice that this algorithm *implements edge descriptors through a set of images*. An alternative would be to define a data structure gathering all the properties of an edge.

1. Apply Gaussian smoothing to I (algorithm LINEAR_FILTER of Chapter 3 with a Gaussian kernel discretising G), obtaining $J = I * G$.
2. For each pixel (i, j):

   (a) compute the gradient components, J_x and J_y (Appendix, section A.2);
   (b) estimate the edge strength

$$e_s(i, j) = \sqrt{J_x^2(i, j) + J_y^2(i, j)}$$

   (c) estimate the orientation of the edge normal

$$e_o(i, j) = \arctan \frac{J_y}{J_x}$$

The output is a *strength image*, E_s, formed by the values $e_s(i, j)$, and an *orientation image*, E_o, formed by the values $e_o(i, j)$.

☞ To make the implementation as efficient as possible, you can use the fact that $\nabla(G * I) = \nabla G * I$ (that is, convolve the image with the 1-D first derivative of a Gaussian) and Gaussian separability. You can also use lookup tables to store the square root values.

☞ The value of σ to be used depends on the length of interesting connected contours, the noise level, and the localization-detection trade off.

We must now find the position of edge centers and discard as many noisy edges as possible in the output of the enhancing filter. This is done in two steps: *nonmaximum suppression* and *thresholding*.

Nonmaximum Suppression. The strength image, E_s, output by CANNY_ENHANCER may contain wide ridges around the local maxima. Nonmaximum suppression thins such ridges to produce 1-pixel wide edges. Here is an essential algorithm, using a rather coarse quantisation of edge normal directions (4-levels).

Algorithm NONMAX_SUPPRESSION

The input is the output of CANNY_ENHANCER, that is, the edge strength and orientation images, E_s and E_o. Consider the four directions $d_1 \ldots d_4$, identified by the 0°, 45°, 90° and 135° orientations (with respect to the horizontal axis image reference frame).

For each pixel (i, j):

1. find the direction, $\hat{d}_k$, which best approximates the direction $E_o(i, j)$ (the normal to the edge);
2. if $E_s(i, j)$ is smaller than at least one of its two neighbors along $\hat{d}_k$, assign $I_N(i, j) = 0$ (suppression); otherwise assign $I_N(i, j) = E_s(i, j)$.

The output is an image, $I_N(i, j)$, of the thinned edge points (that is, $E_s(i, j)$ after suppressing nonmaxima edge points).

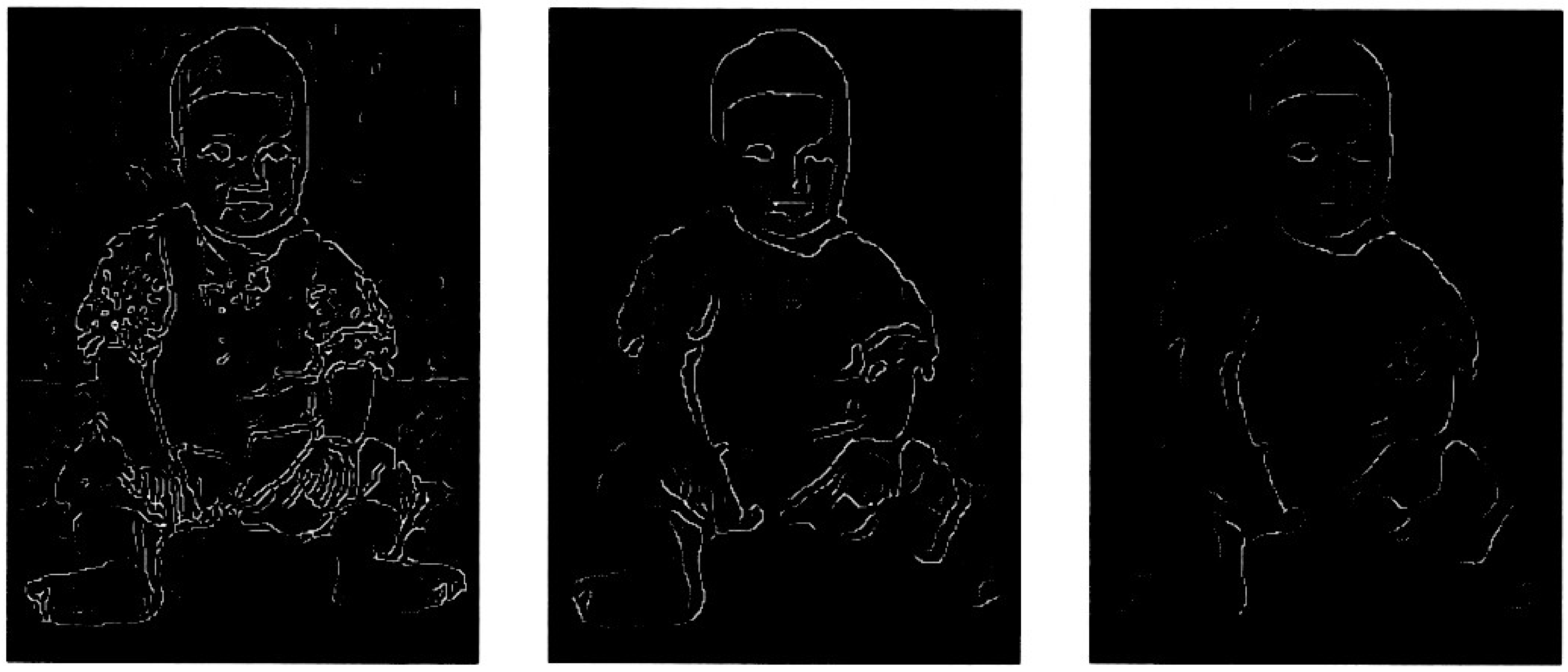


Figure 4.5 Strength images output by CANNY_ENHANCER run on Figure 4.1, after nonmaximum suppression, showing the effect of varying the filter's size, that is, the standard deviation, σ_f, of the Gaussian. Left to right: $\sigma_f = 1, 2, 3$ pixel.

Figure 4.5 shows the output of our implementation of CANNY_ENHANCER, after nonmaximum suppression, when run on the image in Figure 4.1 (left)[8] with three values of standard deviation of the filtering Gaussian. Notice how smaller filters capture shorter edges, but several of these do not belong to any interesting contour in the image (e.g., background edges).

Thresholding. The image output by NONMAX_SUPPRESSION, I_N, still contains the local maxima created by noise. How do we get rid of these? We can try to discard all pixels of value less than a threshold, but this has two problems:

- if we set a low threshold in the attempt of capturing true but weak edges, some noisy maxima will be accepted too (*false contours*);
- the values of true maxima along a connected contours may fluctuate above and below the threshold, fragmenting the resulting edge (*streaking*).

A solution is *hysteresis thresholding*.

Algorithm HYSTERESIS_THRESH

The input is I_N, the output of NONMAX_SUPPRESSION, E_o, the edge orientation image, and τ_l, τ_h, two thresholds such that $\tau_l < \tau_h$.

For all the edge points in I_N, and scanning I_N in a fixed order:

1. Locate the next unvisited edge pixel, $I_N(i, j)$, such that $I_N(i, j) > \tau_h$;

[8] Without the highlighted scanline, obviously!

2. Starting from $I_N(i, j)$, follow the chains of connected local maxima, in both directions perpendicular to the edge normal, as long as $I_N > \tau_l$. Mark all visited points, and save a list of the locations of all points in the connected contour found.

The output is a set of lists, each describing the position of a connected contour in the image, as well as the strength and the orientation images, describing the properties of the edge points.

Hysteresis thresholding reduces the probability of false contours, as they must produce a response higher than τ_h to occur, as well as the probability of streaking, which requires now much larger fluctuations to occur than in the single-threshold case.

☞ If τ_h is large, τ_l can also be set to 0.

Notice that HYSTERESIS_THRESH performs *edge tracking*: it finds chains of connected edge maxima, or connected contours. The descriptors for such chains, saved by HYSTERESIS_THRESH in addition to edge point descriptors, can be useful for curve detection.

☞ Notice that Y-junctions will be split by NONMAX_SUPPRESSION. How serious this is depends on what the edges are computed for. A possible solution is to modify HYSTERESIS_THRESH so that it recognizes Y-junctions, and interrupt all edges.

Figure 4.6 shows the output of our implementation of NONMAX_SUPPRESSION and HYSTERESIS_THRESH when run on the images in Figure 4.5. All contours are one pixel wide, as desired.

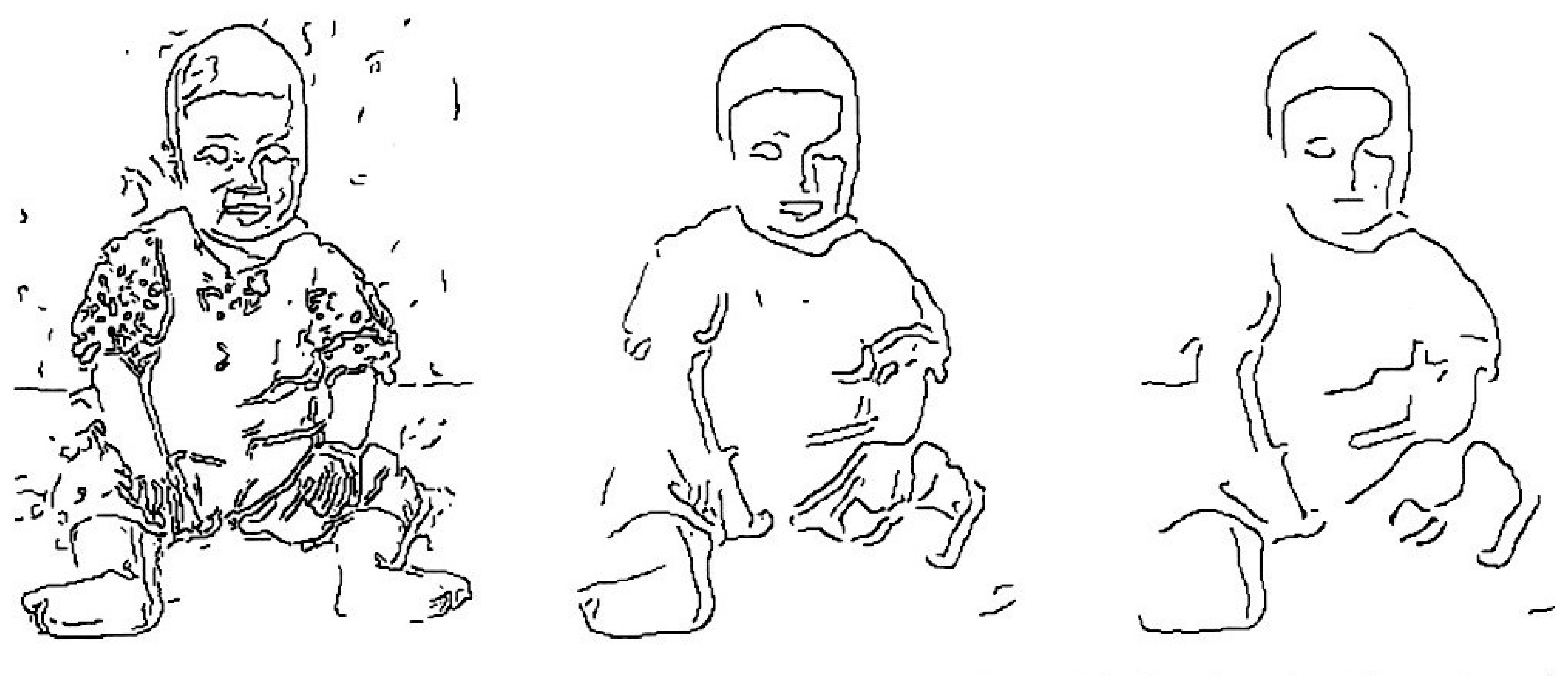


Figure 4.6 Output of HYSTERESIS_THRESH run on Figure 4.5, showing the effect of varying the filter's size. Left to right: $\sigma_f = 1, 2, 3$ pixel. The grey levels has been inverted (black on white) for clarity.

4.2.3 Other Edge Detectors

Early edge detection algorithms were less formalized mathematically than Canny's. We sketch two examples, the *Roberts* and the *Sobel* edge detectors, which are easily implemented in their essential form.

Algorithm ROBERTS_EDGE_DET

The input is formed by an image, I, and a threshold, τ.

1. apply noise smoothing as appropriate (for instance, Gaussian smoothing in the absence of information on noise: see Chapter 3), obtaining a new image I_s;
2. filter I_s (algorithm LINEAR_FILTER, Chapter 3) with the masks
$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$
obtaining two images I_1 and I_2;
3. estimate the gradient magnitude at each pixel (i, j) as
$$G(i,j) = \sqrt{I_1^2(i,j) + I_2^2(i,j)},$$
obtaining an image of magnitude gradients, G;
4. mark as edges all pixels (i, j) such that $G(i, j) > \tau$.

The output is the location of edge points obtained in the last step.

Algorithm SOBEL_EDGE_DET

Same as for ROBERTS_EDGE_DET, but replace step 2. with the following.

2. filter I_s (algorithm LINEAR_FILTER, Chapter 3) with the masks
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
obtaining two images I_1 and I_2.

Notice that the element special to these two detectors is the edge-enhancing filter (see Review Questions). Figure 4.7 shows an example of Sobel edge detection.

4.2.4 Concluding Remarks on Edge Detection

Evaluating Edge Detectors. The ultimate evaluation for edge detectors which are part of larger vision systems is whether or not a particular detector improves the performance of the global system, other conditions being equal. For instance, within an

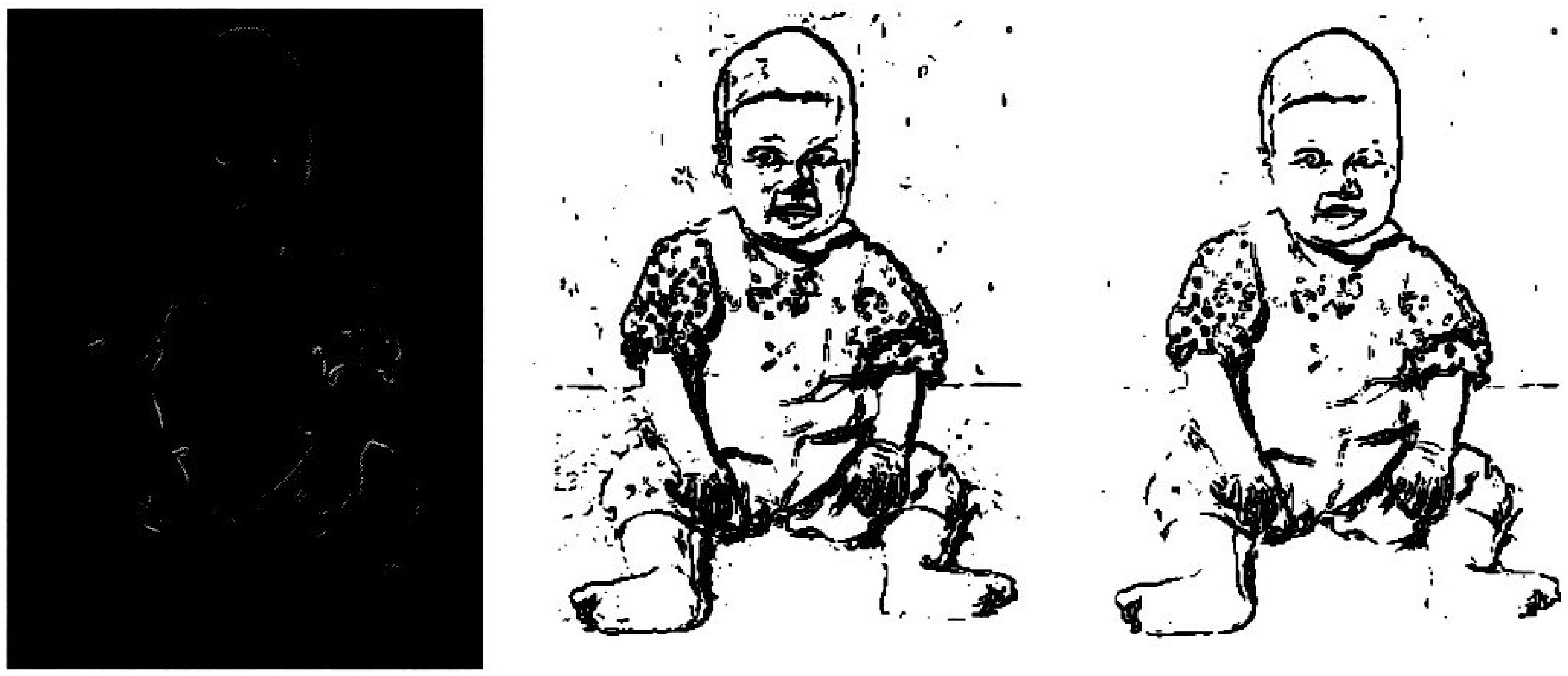


Figure 4.7 Left: output of Sobel edge enhancer run on Figure 4.1. Middle: edges detected by thresholding the enhanced image at 35. Right: same, thresholding at 50. Notice that some contours are thicker than one pixel (compare with Figure 4.5).

inspection system, the detector leading to the best accuracy in the target measurements, and acceptably fast, is to be preferred.

However, it is useful to evaluate edge detectors *per se* as well. We run edge detectors in the hope of finding the contours of interesting scene elements; therefore, one could think that an edge detector is good "if it finds object contours", and in a sense this is true. But it is also imprecise and unfair, because edge detectors do not know about "objects contours" at all; they look for intensity variations, and we must evaluate algorithms for what they actually know and do. Specific edge detectors can be evaluated

- *theoretically* (e.g., for edge enhancement filters, using Canny's criteria in section 4.2.2);
- *experimentally*, estimating various *performance indices* in a large number of experiments with synthetic images (Appendix, section A.1)), in which all edge and noise parameters are perfectly known and vary in realistic ranges. Performance indices include
  - the number of spurious edges (which is an estimate *a posteriori* of the probability of false detection),
  - the number of true edges missed (which is an estimate *a posteriori* of the probability of misses),
  - the average and RMS errors of estimates of edge position and orientation.

Subpixel-Precision Edge Detection. All the edge detectors in this chapter identify the pixel *which contains* the center of the true edge center. In fact, the center could be anywhere within the pixel, so that the average accuracy is 0.5 pixel. In precision applications, half a pixel may correspond to unacceptably large errors in millimiters, and it is important to locate the edge position with *subpixel precision*. For instance, in commercial laser scanners (Chapter 2) an accuracy of about 0.25mm is a common target, and half a pixel may correspond to less than 0.5mm. The easiest way to achieve subpixel resolution is to locate the peak of a parabola interpolating three values in the output of CANNY_ENHANCER, namely at the edge pixel and at its two neighbours along the edge normal. Of course, any information available on edge profiles and noise should be exploited to improve on the parabola method.

4.3 Point Features: Corners

Although the mathematics of edge detection may seem involved, edges can be characterized intuitively in geometric terms: they are the projection of object boundaries, surface marks, and other interesting elements of a scene. We now give an example of image features that can be characterized more easily than edges in mathematical terms, but do not correspond necessarily to any geometric entities of the observed scene. These features can be interpreted as *corners*, but not only in the sense of intersections of image lines; they capture corner structures in patterns of intensities. Such features prove stable across sequences of images, and are therefore interesting to track objects across sequences (Chapter 8).

How do we detect corner features? Consider the spatial image gradient, $[E_x, E_y]^\top$ (the subscripts indicate partial differentiation, e.g., $E_x = \frac{\partial E}{\partial x}$). Consider a generic image point, p, a neighbourhood Q of p, and a matrix, C, defined as

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}, \tag{4.9}$$

where the sums are taken over the neighbourhood Q. This matrix characterizes the *structure* of the grey levels. How?

The key to the answer is in the eigenvalues of C and their geometric interpretation. Notice that C is symmetric, and can therefore be diagonalized by a rotation of the coordinate axes; thus, with no loss of generality, we can think of C as a diagonal matrix:

$$C = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}.$$

The two eigenvalues, λ_1 and λ_2, are both nonnegative (why?); let us assume $\lambda_1 \geq \lambda_2$. The geometric interpretation of λ_1 and λ_2 can be understood through a few particular cases. First, consider a perfectly uniform Q: the image gradient vanishes everywhere, C becomes the null matrix, and we have $\lambda_1 = \lambda_2 = 0$. Second, assume that Q contains an ideal black and white step edge: we have $\lambda_2 = 0$, $\lambda_1 > 0$, and the eigenvector associated with λ_1 is parallel to the image gradient. Note that C is rank deficient in both cases, with rank 0 and 1 respectively. Third, assume that Q contains the corner of a black square against a white background: as there are two principal directions in Q, we expect $\lambda_1 \geq \lambda_2 > 0$, and the larger the eigenvalues, the stronger (higher contrast) their corresponding image lines. At this point, you have caught on with the fact that *the eigenvectors encode edge directions, the eigenvalues edge strength*. A corner is identified

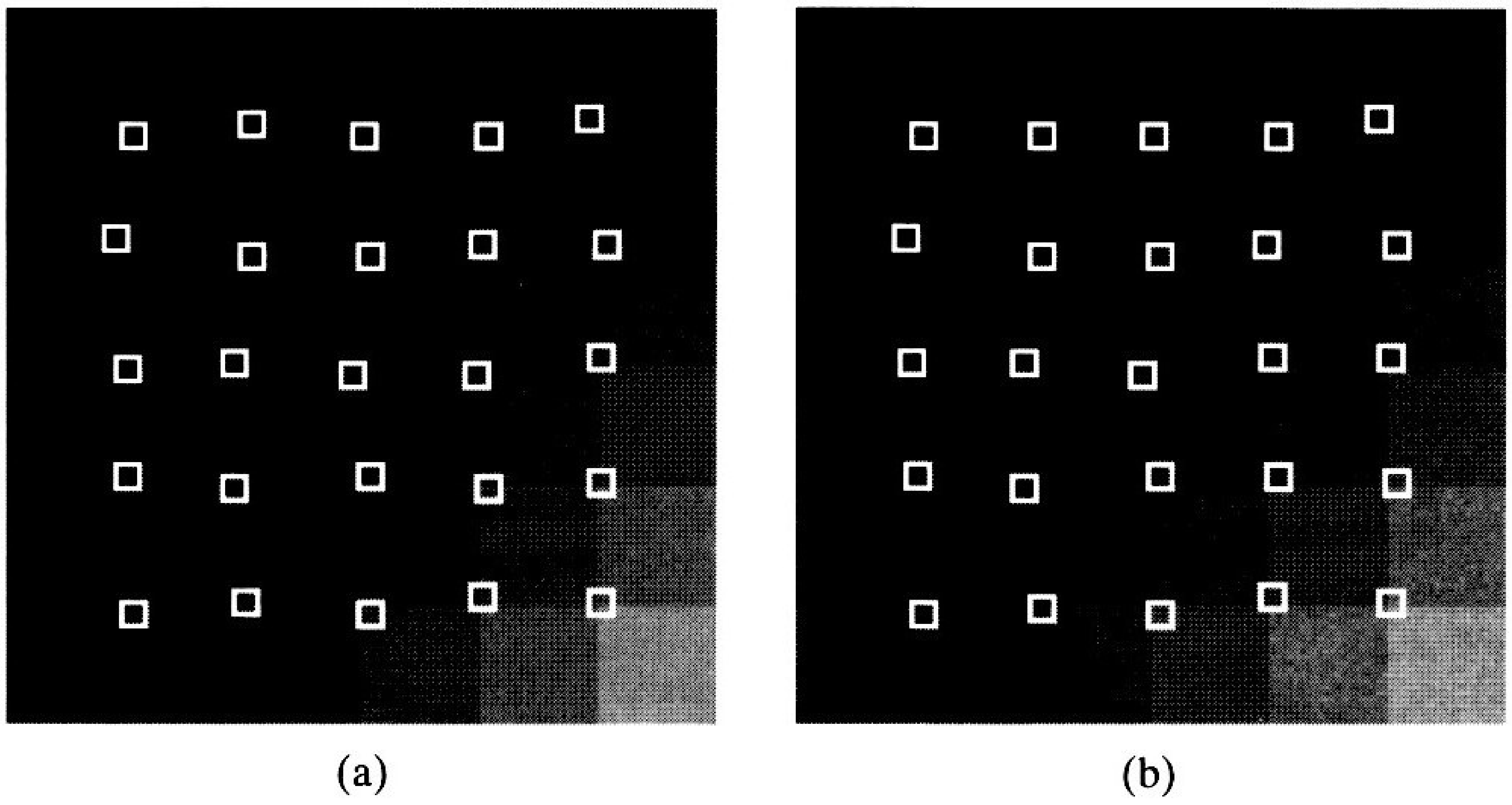


Figure 4.8 Corners found in a 8-bit, synthetic checkerboard image, corrupted by two realizations of synthetic Gaussian noise of standard deviation 2. The corner is the bottom right point of each 15 × 15 neighbourhood (highlighted).

by two strong edges; therefore, as $\lambda_1 \geq \lambda_2$, *a corner is a location where the smaller eigenvalue, λ_2, is large enough.*

Time for examples. Figure 4.8 shows the corners found in a synthetic image of a checkerboard, with and without additive noise. Figure 4.9 shows the corners found in the image of a building, and the histogram of the λ_2 values. The shape of this histogram is rather typical for most natural images. If the image contains uniform regions, or many almost ideal step edges, the histogram has a second peak at $\lambda_2 = 0$. The tail (right) of the histogram is formed by the points for which λ_2 is large, which are precisely the points (or, equivalently, the neighbourhoods) we are interested in. Figure 4.10 shows another example with a road scene.

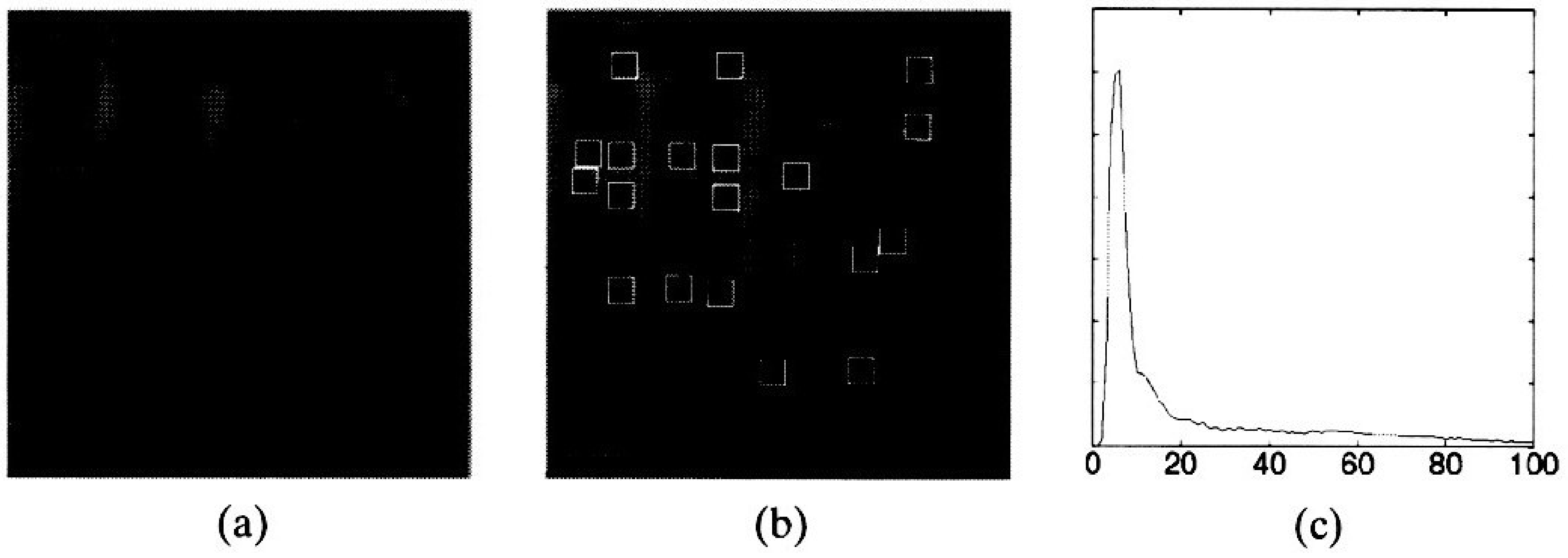


Figure 4.9 (a): original image of a building. (b): the 15 × 15 pixel neighbourhoods of some of the image points for which $\lambda_2 > 20$. (c): histogram of λ_2 values across the image.

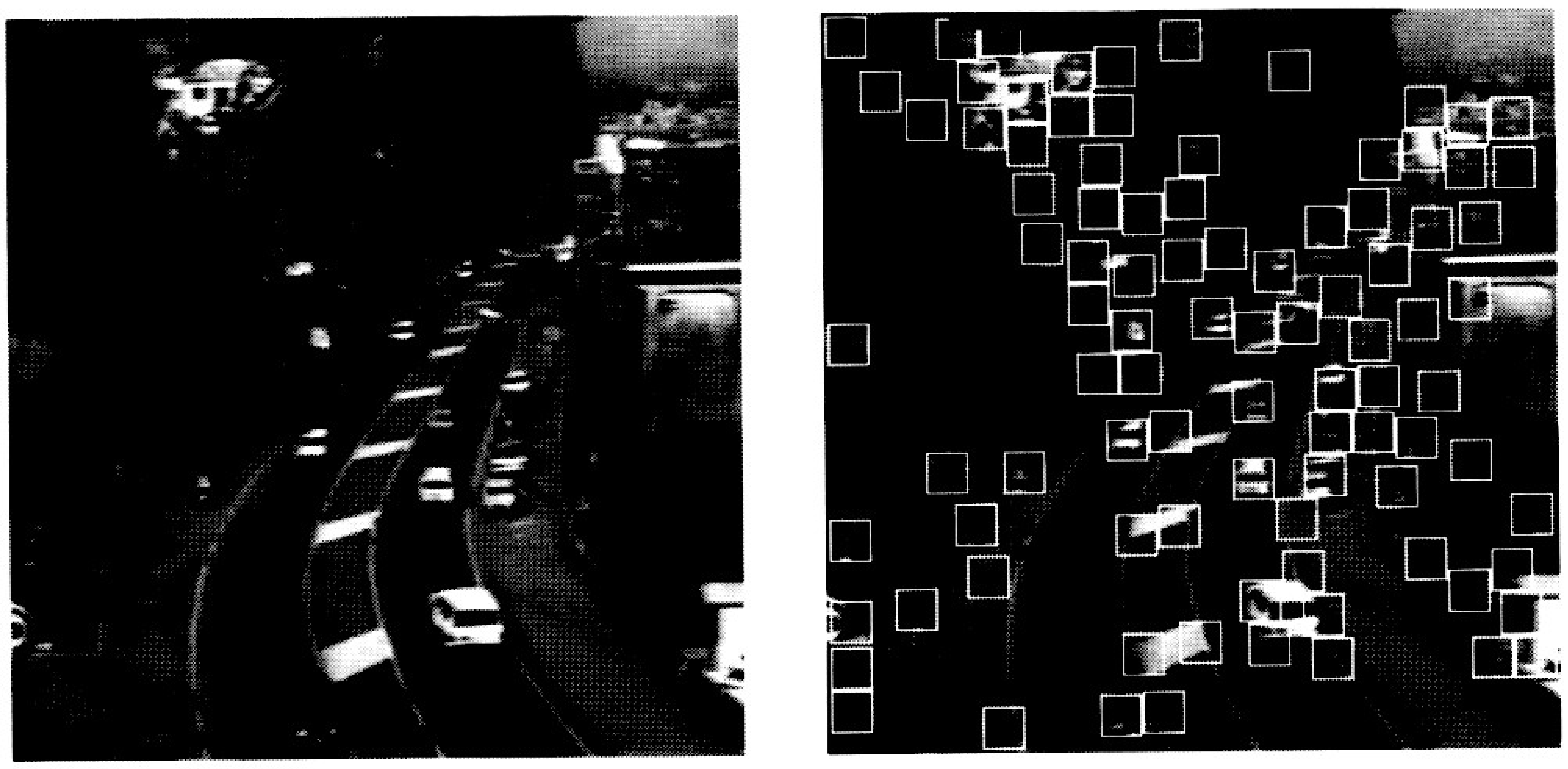


Figure 4.10 (a): image of an outdoor scene. The corner is the bottom right point of each 15×15 neighbourhood (highlighted). (b): corners found using a 15×15 neighbourhood.

We reiterate that our feature points include high-contrast image corners and T-junctions generated by the intersection of object contours (as the corners in Figure 4.8), but also corners of the local intensity pattern not corresponding to obvious scene features (as some of the corners in Figure 4.10). In general terms, at corners points *the intensity surface has two well-pronounced, distinctive directions, associated to eigenvalues of C both significantly larger than zero.*

We now summarize the procedure for locating this new type of image features.

Algorithm CORNERS

The input is formed by an image, I, and two parameters: the threshold on λ_2, τ, and the linear size of a square window (neighbourhood), say $2N + 1$ pixels.

1. Compute the image gradient over the entire image I;
2. For each image point p:
   (a) form the matrix C of (4.9) over a $(2N + 1) \times (2N + 1)$ neighbourhood Q of p;
   (b) compute λ_2, the smaller eigenvalue of C;
   (c) if $\lambda_2 > \tau$, save the coordinates of p into a list, L.
3. Sort L in decreasing order of λ_2.
4. Scanning the sorted list top to bottom: for each current point, p, delete all points appearing further on in the list which belong to the neighbourhood of p.

The output is a list of feature points for which $\lambda_2 > \tau$ and whose neighbourhoods do not overlap.

Algorithm CORNERS has two main parameters: the threshold, τ, and the size of the neighbourhood, $(2N + 1)$. The threshold, τ, can be estimated from the histogram of λ_2 (Exercise 4.6), as the latter has often an obvious valley near zero (Figure 4.9).

☞ Notice that such valley is not *always* present (Exercise 4.7).

Unfortunately, there is no simple criterion for the estimation of the optimal size of the neighbourhood. Experience indicates that choices of N between 2 and 10 are adequate in most practical cases.

☞ In the case of corner points, the value of N is linked to the location of the corner within the neighbourhood. As you can see from Figure 4.9, for relatively large values of N the corner tends to move away from the neighbourhood center (see Exercise 4.8 for a quantitative analysis of this effect).

4.4 Surface Extraction from Range Images

Many 3-D objects, especially man-made, can be conveniently described in terms of the shape and position of the surfaces they are made of. For instance, you can describe a cone as an object formed by two surface patches, one conical and one planar, the latter perpendicular to the axis of the former. Surface-based descriptions are used for object classification, pose estimation, and reverse engineering, and are ubiquitous in computer graphics.

As we have seen in Chapter 2, range images are basically a sampled version of the visible surfaces in the scene. Therefore, ignoring the distortions introduced by sensor imperfections, *the shape of the image surface*[9] *and the shape of the visible scene surfaces are the same*, and any geometric property holding for one holds for the other too. This section presents a well-known method to find patches of various shapes composing the visible surface of an object. The method, called *HK segmentation*, partitions a range image into regions of homogeneous shape, called *homogeneous surface patches*, or just *surface patches* for short.[10] The method is based on differential geometry; Appendix, section A.5 gives a short summary of the basic concepts necessary.

☞ The solution to several computer vision problems involving 3-D object models are simpler when using 3-D features than 2-D features, as image formation must be taken into account for the latter.

[9] That is, the image values regarded as a surface defined on the image plane.

[10] Notice that surface patches are the basic ingredients for building a surface-based CAD model of an object automatically.