3

Dealing with Image Noise

The mariachis would serenade
And they would not shut up till they were paid.

Tom Lehrer, *In Old Mexico*

Attenuating or, ideally, suppressing image noise is important because any computer vision system begins by processing intensity values. This chapter introduces a few, basic noise models and filtering methods, which constitute an initial but useful toolkit for many practical situations.

Chapter Overview

Section 3.1 discusses the concept of noise and how to quantify it. It also introduces *Gaussian* and *impulsive noise*, and their effects on images.

Section 3.2 discusses some essential linear and a nonlinear filtering methods, aimed to attenuate random and impulsive noise.

What You Need to Know to Understand this Chapter

- The basics of signal theory: sampling theorem (Appendix, section A.3), Fourier transforms, and linear filtering.

3.1 Image Noise

Chapter 2 introduced the concept of acquisition noise, and suggested a method to estimate it. But, in general, the term *noise* covers much more.

Noise

In computer vision, *noise* may refer to any entity, in images, data or intermediate results, that is not interesting for the purposes of the main computation.

For example, one can speak of noise in different cases:

- For image processing algorithms, like edge or line detection, noise might be the spurious fluctuations of pixel values introduced by the image acquisition system.
- For algorithms taking as input the results of some numerical computation, noise can be the errors introduced in the latter by random fluctuations or inaccuracies of input data, the computer's limited precision, round-off errors, and the like.
- For algorithms trying to group lines into meaningful objects, noise is the contours which do not belong to any meaningful object.

☞ In computer vision, what is considered noise for a task is often the interesting signal for a different task.[1]

Different types of noise are countered by different techniques, depending on the noise's nature and characteristics. This chapter concentrates on *image noise*. It must be clear that *noise filtering* is a classic topic of both signal and image processing, and the literature on the subject is *vast* (see section 3.4, Further Readings). This chapter is just meant to provide a few starting tools which prove useful in many practical situations.

It is now time to formalize better our *dramatis persona*.

Image Noise

We shall assume that the main image noise is *additive* and *random*; that is a spurious, random signal, $n(i, j)$, added to the true pixel values $I(i, j)$:

$$\hat{I}(i, j) = I(i, j) + n(i, j) \tag{3.1}$$

Noise Amount

The *amount of noise* in an image can be estimated by means of σ_n, the standard deviation of the random signal $n(i, j)$. It is important to know how strong is the noise with respect to the interesting signal. This is specified by the *signal-to-noise ratio*, or SNR:

$$SNR = \frac{\sigma_s}{\sigma_n} \tag{3.2}$$

where σ_s is the standard deviation of the signal (the pixel values $I(i, j)$). The SNR is often expressed in decibel:

$$SNR_{dB} = 10 \log_{10} \frac{\sigma_s}{\sigma_n} \tag{3.3}$$

[1] This observation was made by Jitendra Malik.

☞ Additive noise is an adequate assumption for the image acquisition systems introduced in Chapter 2, but in some cases the noise might not be additive. For instance, *multiplicative noise*, whereby $\hat{I} = nI$, models image degradation in television lines and photographs owing to grain size.

Notice that we assume that the resolution of the quantized grey levels is sufficient to sample the image appropriately; that is, to represent *all* significant variations of the image irradiance.[2] Coarse quantization can introduce spurious contours and is thus called *quantization noise*. Byte images (256 grey levels per pixel), introduced in Chapter 2, appear to be adequate for most practical purposes and are extremely popular.

3.1.1 Gaussian Noise

In the absence of information, one often assumes $n(i, j)$ to be modelled by a *white, Gaussian, zero-mean stochastic process*. For each location (i, j), this amounts to thinking of $n(i, j)$ as a random variable, distributed according to a zero mean Gaussian distribution function of fixed standard deviation, which is added to $I(i, j)$ and whose values are completely independent of each other and of the image in both space and time.

This simple model predicts that noise values are distributed symmetrically around zero and, consequently, pixel values $\hat{I}(i, j)$ around their true values $I(i, j)$; this is what you expect from good acquisition systems, which, in addition, should guarantee low noise levels. Moreover, it is easier to deal formally with Gaussian distributions than with many other statistical models. To illustrate the effect of Gaussian noise on images, Figure 3.1 (a) shows a synthetic grey-level "checkerboard" pattern and the profile of the grey levels along a horizontal scanline. Figure 3.1 (b) shows the same image corrupted by additive Gaussian noise, and the profile of the grey levels along the same scanline.

☞ The Gaussian noise model is often a convenient approximation dictated by ignorance: if we do not know and *cannot estimate* the noise characteristics, we take it to be Gaussian. Be aware, however, that white Gaussian noise is just an approximation of additive real noise! You should always try and discover as much as possible about the origin of the noise; e.g., investigating which sensor acquired the data, and design suppression methods optimally tailored to its characteristics. This is known as *image restoration*, another vast chapter of image processing.

3.1.2 Impulsive Noise

Impulsive noise, also known as *spot* or *peak noise*, occurs usually in addition to the one normally introduced by acquisition. Impulsive noise alters random pixels, making their values very different from the true values and very often from those of neighboring pixels too. Impulsive noise appears in the image as a sprinkle of dark and light spots. It can be caused by transmission errors, faulty elements in the CCD array, or external noise corrupting the analog-to-digital conversion.

---

[2] Of course, what a significant variation is depends on what you are after. This is discussed further in Chapter 4.

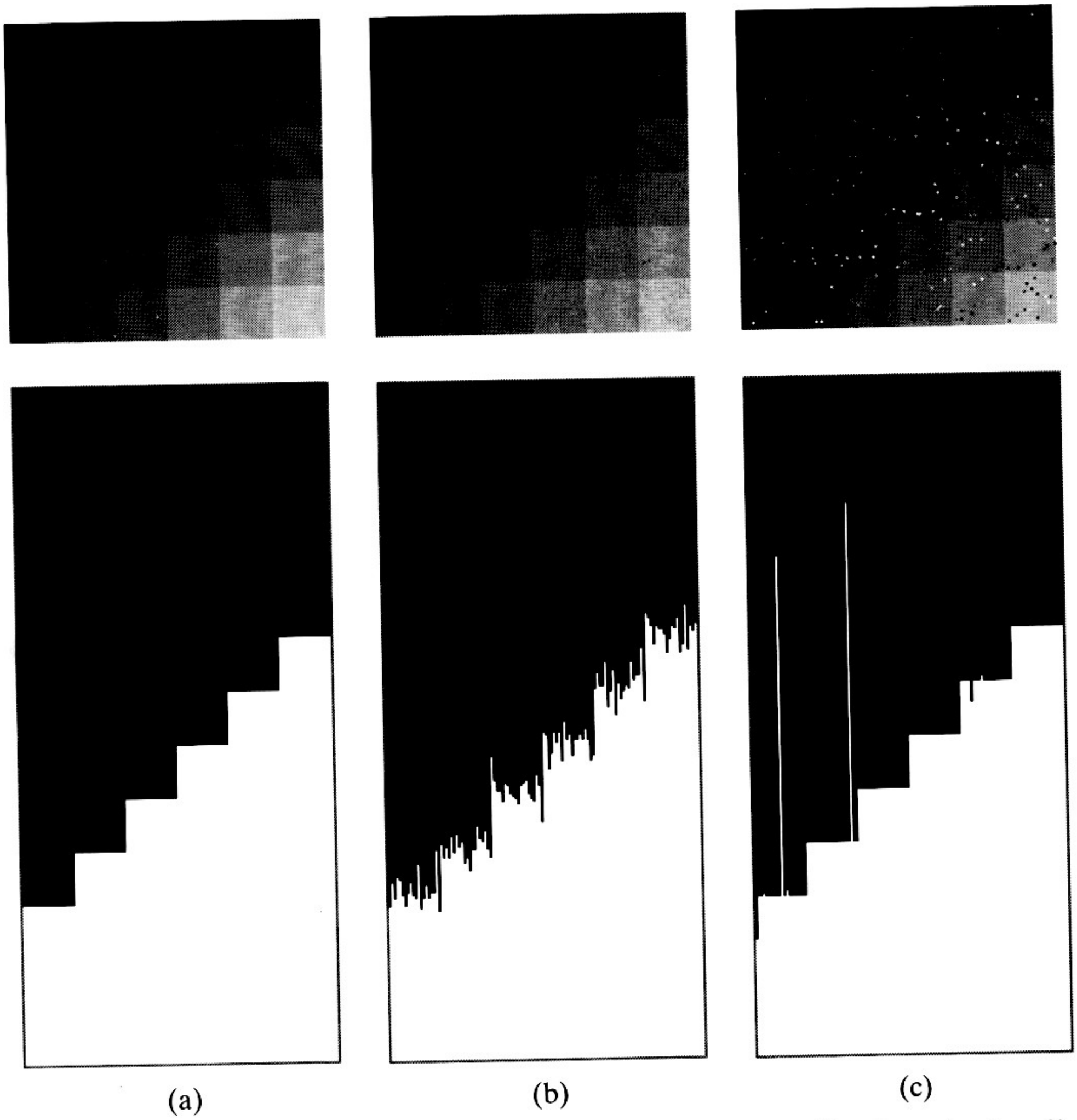


Figure 3.1 (a) Synthetic image of a 120 × 120 grey-level "checkerboard" and grey-level profile along a row. (b) After adding zero-mean Gaussian noise ($\sigma = 5$). (c) After adding salt and pepper noise (see text for parameters).

Salt-and-pepper noise is a model adopted frequently to simulate impulsive noise in synthetic images. The noisy image values $I_{sp}(h, k)$ are given by

$$I_{sp}(h,k) = \begin{cases} I(h,k) & x < l \\ i_{min} + y(i_{max} - i_{min}) & x \geq l \end{cases} \tag{3.4}$$

where I is the true image, $x, y \in [0, 1]$ are two uniformly distributed random variables, l is a parameter controlling how much of the image is corrupted, and i_{min}, i_{max} how severe

is the noise. You can obtain *saturated salt-and-pepper noise* turning y into a two-valued variable ($y = 0$ or $y = 1$), and setting $i_{min} = 0$ and $i_{max} = 255$.

To illustrate the effects of salt and pepper noise on images, Figure 3.1 (right) shows the "checkerboard" pattern and the same scanline of Figure 3.1 (left) corrupted by salt and pepper noise with $i_{min} = 0$, $i_{max} = 255$, and $l = .99$.

3.2 Noise Filtering

Problem Statement: Noise Suppression, Smoothing, and Filtering

Given an image I corrupted by noise n, attenuate n as much as possible (ideally, eliminate it altogether) without altering I significantly.

Attenuating or, if possible, suppressing image noise is important as the result of most computations on pixel values might be distorted by noise. An important example is computing image derivatives, which is the basis of many algorithms: any noise in the signal can result in serious errors in the derivatives (see Exercise 3.3). A common technique for noise smoothing is *linear filtering*, which consists in convolving the image with a constant matrix, called *mask* or *kernel*.[3] As a reminder, here is the basic linear filtering algorithm.

Algorithm LINEAR_FILTER

Let I be a $N \times M$ image, m an odd number smaller than both N and M, and A the kernel of a linear filter, that is a $m \times m$ mask. The filtered version I_A of I at each pixel (i, j) is given by the discrete convolution

$$I_A(i, j) = I * A = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} A(h, k) I(i - h, j - k) \tag{3.5}$$

where $*$ indicates discrete convolution, and $m/2$ integer division (e.g., $3/2 = 1$).

A linear filter replaces the value $I(i, j)$ with a weighted sum of I values in a neighborhood of (i, j); the weights are the entries of the kernel. The effects of a linear filter on a signal can be better appreciated in the frequency domain. Through the *convolution theorem*, the Fourier transform of the convolution of I and A is simply the product of their Fourier transforms $\mathcal{F}(I)$ and $\mathcal{F}(A)$. Therefore, the result of convolving a signal with A is to attenuate (or suppress) the signal frequencies corresponding to low (or zero) values of $|\mathcal{F}(A)|$, spectrum of the filter A.

[3] The name *linear* is due to the fact that the convolution with a constant kernel models space- and time-invariant linear systems. From this point of view, the kernel is the *impulse response* of the filter.

3.2.1 Smoothing by Averaging

If all entries of A in (3.5) are non-negative, the filters performs *average smoothing*. The simplest smoothing kernel is the *mean filter*, which replaces a pixel value with the mean of its neighborhood; for instance, with $m = 3$

$$A_{avg} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.6}$$

☞ If the sum of all kernel entries is not one, as it happens for averaging kernels, $I_A(i, j)$ must be divided by the sum of the entries, to avoid that the filtered image becomes brighter than the original.

Why does such a filter attenuate noise? Intuitively, averaging takes out small variations: Averaging m^2 noisy values around pixel (i, j) divides the standard deviation of the noise by $\sqrt{m^2} = m$.

Frequency Behavior of the Mean Filter. In the frequency domain we have that the Fourier transform of a 1-D mean filter kernel of width $2W$ is the "sinc" function

$$sinc(\omega) = \frac{2\sin(\omega W)}{\omega},$$

(Figure 3.3 shows an example in 2-D). Since the signal frequencies falling inside the main lobe are weighted more than the frequencies falling in the secondary lobes, the mean filter can be regarded as an approximate "low-pass" filter.

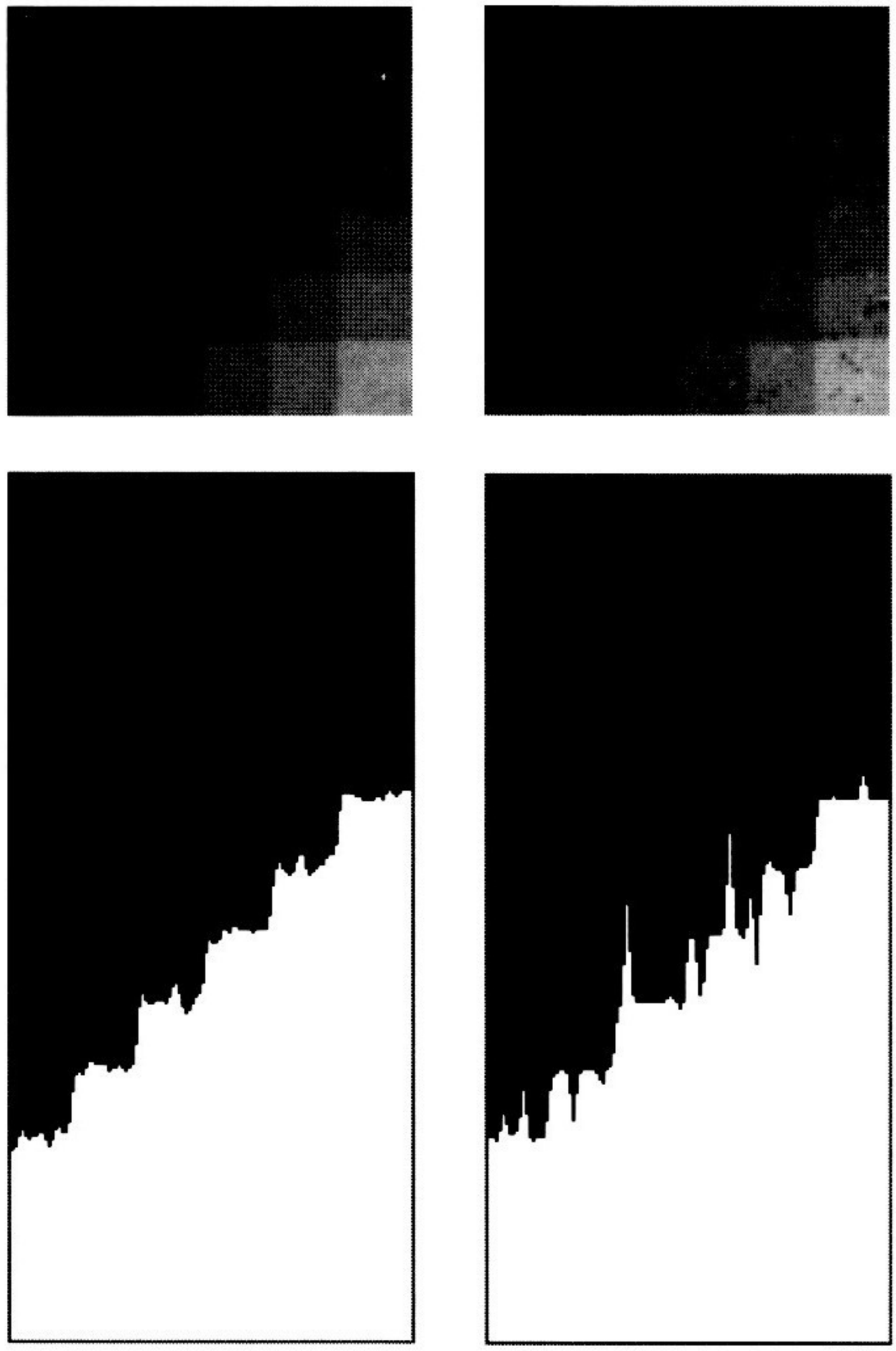
Limitations of Averaging. Averaging is simple but has problems, including at least the following.

1. Signal frequencies shared with noise are lost; this implies that sharp signal variations are filtered out by averaging, and the image is blurred. As we shall see in Chapter 4, blurring affects the accuracy of feature localization.
2. Impulsive noise is only attenuated and diffused, not removed.
3. The secondary lobes in the Fourier transform of the mean filter's let noise into the filtered image.

3.2.2 Gaussian Smoothing

Gaussian smoothing is a particular case of averaging, in which the kernel is a 2-D Gaussian. Its effect is illustrated by Figure 3.2, which shows the results of Gaussian smoothing applied to the noisy "checkerboards" of Figure 3.1(center and right), corrupted by Gaussian and impulsive noise respectively. Notice that impulsive noise has only been attenuated; in fact, each spike has also been spread in space.

Frequency Behavior of the Gaussian Kernel. The Fourier transform of a Gaussian is still a Gaussian and, hence, has no secondary lobes. This makes the Gaussian kernel



(a) (b)

Figure 3.2 (a) Results of applying Gaussian filtering (kernel width 5 pixel, $\sigma = 1$) to the "checkerboard" image corrupted by Gaussian noise, and grey-level profile along a row. (b) Same for the "checkerboard" image corrupted by salt and pepper noise.

a better low-pass filter than the mean filter. A comparison of the mean and Gaussian filters in both the spatial and frequency domain in 2-D is shown in Figure 3.3.

Separability of the Gaussian Kernel. Gaussian smoothing can be implemented efficiently thanks to the fact that the kernel is *separable*:

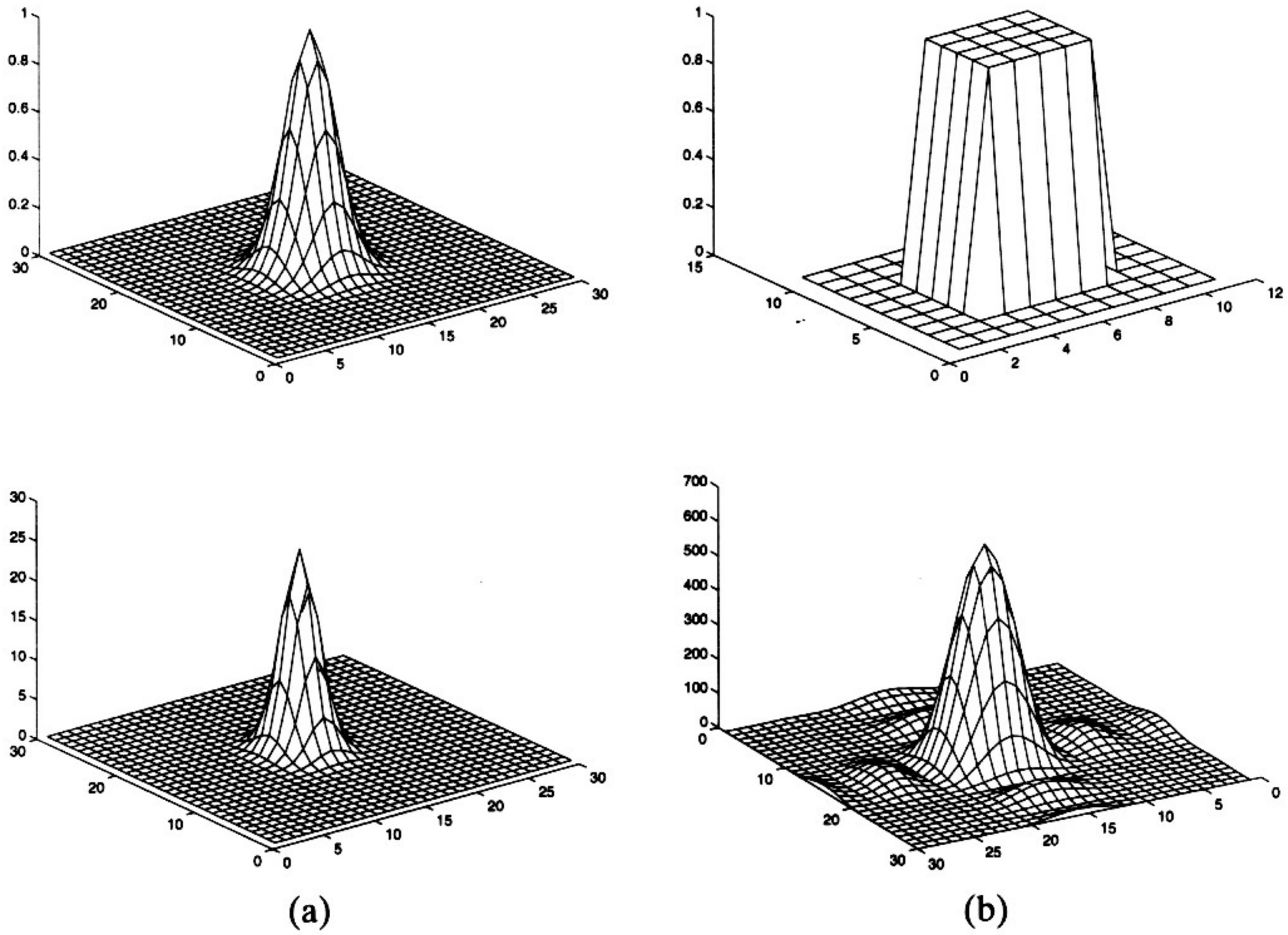


Figure 3.3 (a) The plot of a 5×5 Gaussian kernel of width 5 (top) and its Fourier transform (bottom). (b) The same for a mean-filter kernel.

$$
\begin{aligned}
I_G = I * G = \\
&= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} G(h,k) I(i-h, j-k) = \\
&= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{h^2+k^2}{2\sigma^2}} I(i-h, j-k) = \\
&= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{h^2}{2\sigma^2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{k^2}{2\sigma^2}} I(i-h, j-k) \qquad (3.7)
\end{aligned}
$$

This means that convolving an image I with a 2-D Gaussian kernel G is the same as convolving first all rows, then all columns with a 1-D Gaussian having the same σ. The advantage is that time complexity increases linearly with mask size, instead of quadratically (see Exercise 3.4). The next box gives the obvious algorithm for a separable kernel implemented in the special case of the Gaussian kernel.

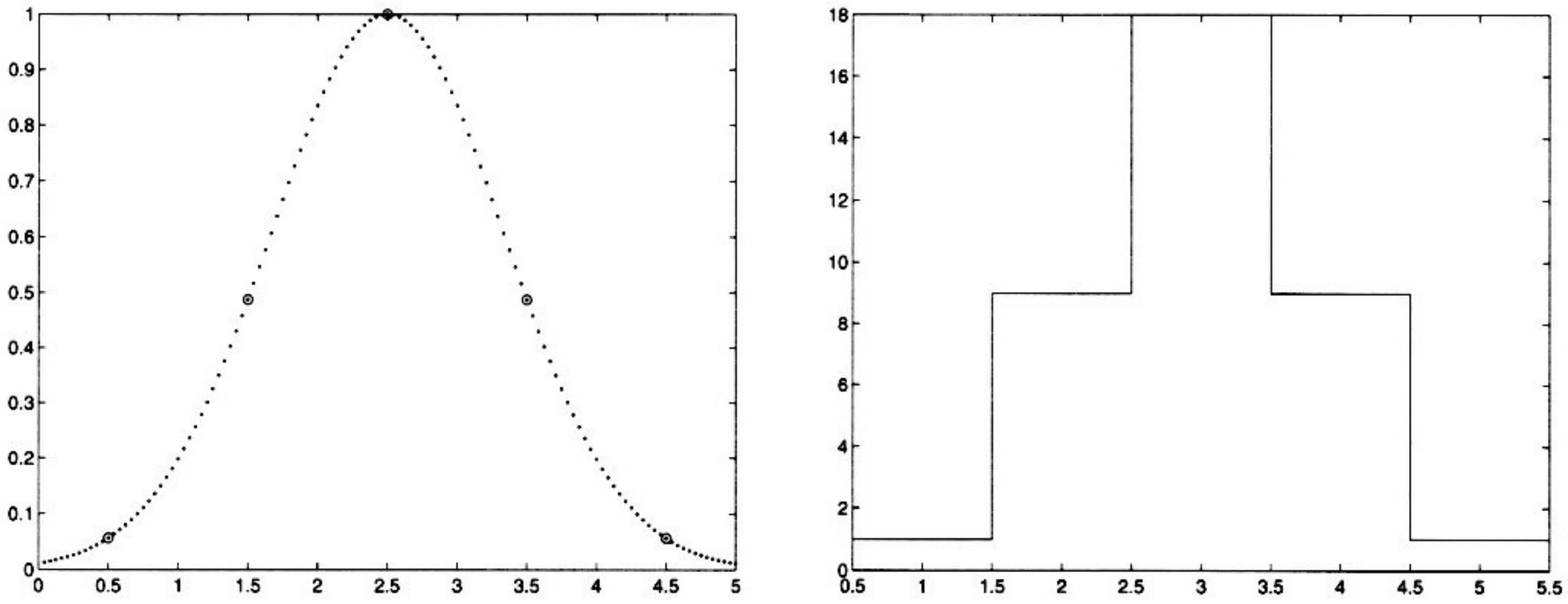


Figure 3.4 (a) 1-D Gaussian (dotted) and real samples (circles) for 5×5 kernel. (b) Plot of corresponding integer kernel.

Algorithm SEPAR_FILTER

To convolve an image I with a $m \times m$ 2-D Gaussian kernel G with $\sigma = \sigma_G$

1. Build a 1-D Gaussian mask g, of width m, with $\sigma_g = \sigma_G$;
2. Convolve each row of I with g, yielding a new image I_r;
3. Convolve each column of I_r with g.

Building Gaussian Kernels. Thanks to the separability of the Gaussian kernel, we can consider only 1-D masks. To build a discrete Gaussian mask, one has to sample a continuous Gaussian. To do so, we must determine the mask width given the Gaussian kernel we intend to use, or, conversely, the σ of the continuous Gaussian given the desired mask width. A relation between σ and the mask width w (typically an odd number) can be obtained imposing that w subtends most of the area under the Gaussian. An adequate choice is $w = 5\sigma$, which subtends 98.76% of the area. Fitting this portion of the Gaussian between the endpoints of the mask, we find that a 3-pixel mask corresponds to $\sigma_3 = 3/5 = 0.6$ pixel; a 5-pixel mask to $\sigma_5 = 5/5 = 1$ pixel; in general,

$$\sigma_w = \frac{w}{5} \tag{3.8}$$

Sampling a continuous Gaussian yields real kernel entries. Filtering times can be greatly reduced by *approximated integer kernels* so that image values being integers too, no floating point operations are necessary at all. To build an integer kernel, you simply normalize the real kernel to make its smallest entry 1, round off the results, and divide by the sum of the entries. Figure 3.4 shows the plot of a 1-D Gaussian profile, the real samples taken, and the 5×5 integer kernel ([1, 9, 18, 9, 1]).

Algorithm INT_GAUSS_KER

To build an approximate, integer kernel G_i:

1. Compute a floating point kernel $G(h,k)$ the same size as G_i; let $g_{min} = G(0,0)$ be the minimum value of G.
2. Determine the normalization factor $f = 1/g_{min}$.
3. Compute the entries of the non-normalized filter as $G_i(h,k) = int[fG(h,k)]$, where *int* indicates closest integer.

3.2.3 Are our Samples Really Gaussian?

You must be aware that problems lurk behind the straightforward recipe we gave for building discrete Gaussian kernels. It is instructive to take a closer look at least at one, sampling. The pixelization imposes a fixed sampling interval of 1 pixel. If the pixel width is taken as unit, by virtue of the sampling theorem (see Appendix, section A.3), we cannot reconstruct completely from its samples a signal containing frequencies higher than 0.5 pixel^{-1}, as any significant component at $|\omega| > \omega_c = 2\pi(0.5) = \pi$ is lost. Notice that ω_c is fixed *only* by the pixelization step, not by the signal.

What are the consequences for building discrete Gaussian kernels? In the *continuum*, the Fourier transform of the Gaussian

$$g(x,\sigma) = e^{-\frac{x^2}{2\sigma^2}}$$

is the Gaussian $g(\omega,\sigma')$ with $\sigma' = 1/\sigma$. As $g(\omega,\sigma')$ is not bandlimited, sampling $g(x,\sigma)$ on the pixel grid implies *necessarily* the loss of all components with $|\omega| > \omega_c$. For relatively small σ this means that the Fourier transform of $g(x,\sigma)$, $g(\omega,\sigma')$, is substantially different from zero well outside the interval $[-\pi,\pi]$, as shown in Figure 3.5. To avoid aliasing, the best we can do is to try keeping most of the energy of $g(\omega,\sigma')$ within the interval $[-\pi,\pi]$. Applying the "98.86% of the area" criterion in the frequency domain, we find

$$5\sigma' = \frac{5}{\sigma} \leq 2\pi$$

or

$$\sigma \geq \frac{5}{2\pi} = 0.796.$$

The preceding inequality tells you that you cannot sample appropriately a Gaussian kernel whose σ is less than 0.8 (in pixel units) no matter how many spatial samples you keep!

We can also interpret this result in terms of the minimum size for a Gaussian kernel. Since $\sigma = w/5$, for $w = 3$ we have $\sigma = 0.6$. Therefore, you cannot build a faithful Gaussian kernel with just 3 samples. For $w = 5$, instead, we have $\sigma = 1$ which means that 5 samples are enough. What happens if you ignore all this? Figure 3.6 shows that the inverse FFT of the FFT of the original Gaussian $g(x,\sigma)$ is significantly different from $g(x,\sigma)$ for $\sigma = 0.6$ $(w = 3)$. In accordance with our prediction, a much smaller difference is found for $\sigma = 1$ $(w = 5)$.

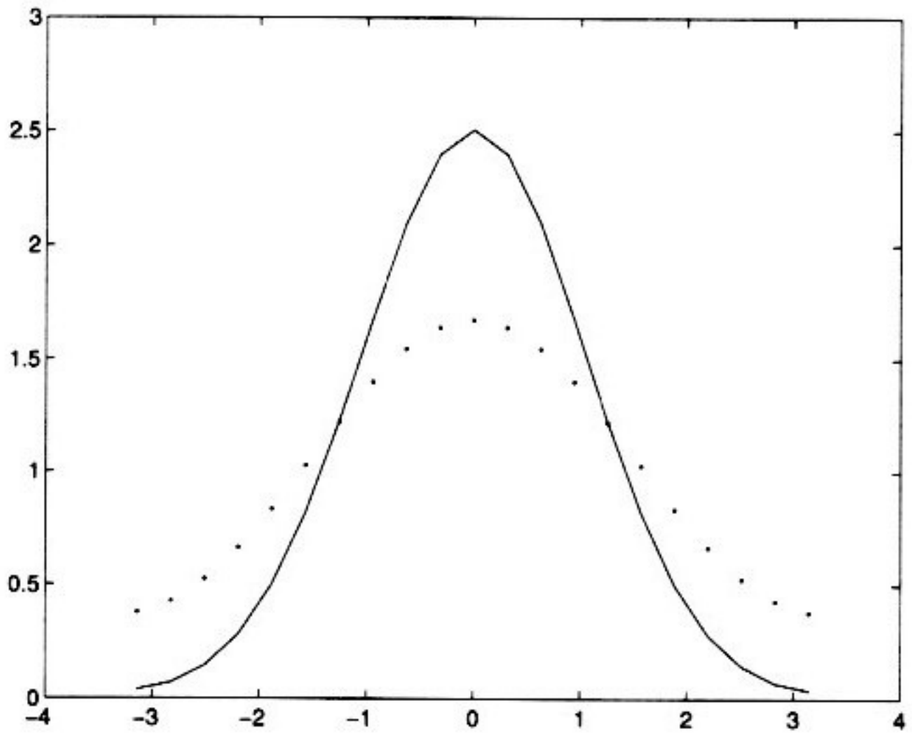


Figure 3.5 The Fourier transforms of two sampled Gaussians, for $w = 3$ ($\sigma = 0.6$, dotted line) and $w = 5$ ($\sigma = 1$, solid line). Notice that a smaller portion of the transform corresponding to $\sigma = 1$ is lost between $-\pi$ and π.

Gaussian Smoothing by Repeated Averaging. *Repeated averaging* (RA) is a simple and efficient way to approximate Gaussian smoothing. It is based on the fact that, by virtue of the central limit theorem, convolving a 3×3 averaging mask n times with an image I approximates the convolution of I with a Gaussian mask of $\sigma = \sqrt{n/3}$ and size $3(n+1) - n = 2n + 3$.

☞ Notice that RA leads to a different relation between σ and n from the one we obtained from the area criterion (Exercise 3.6).

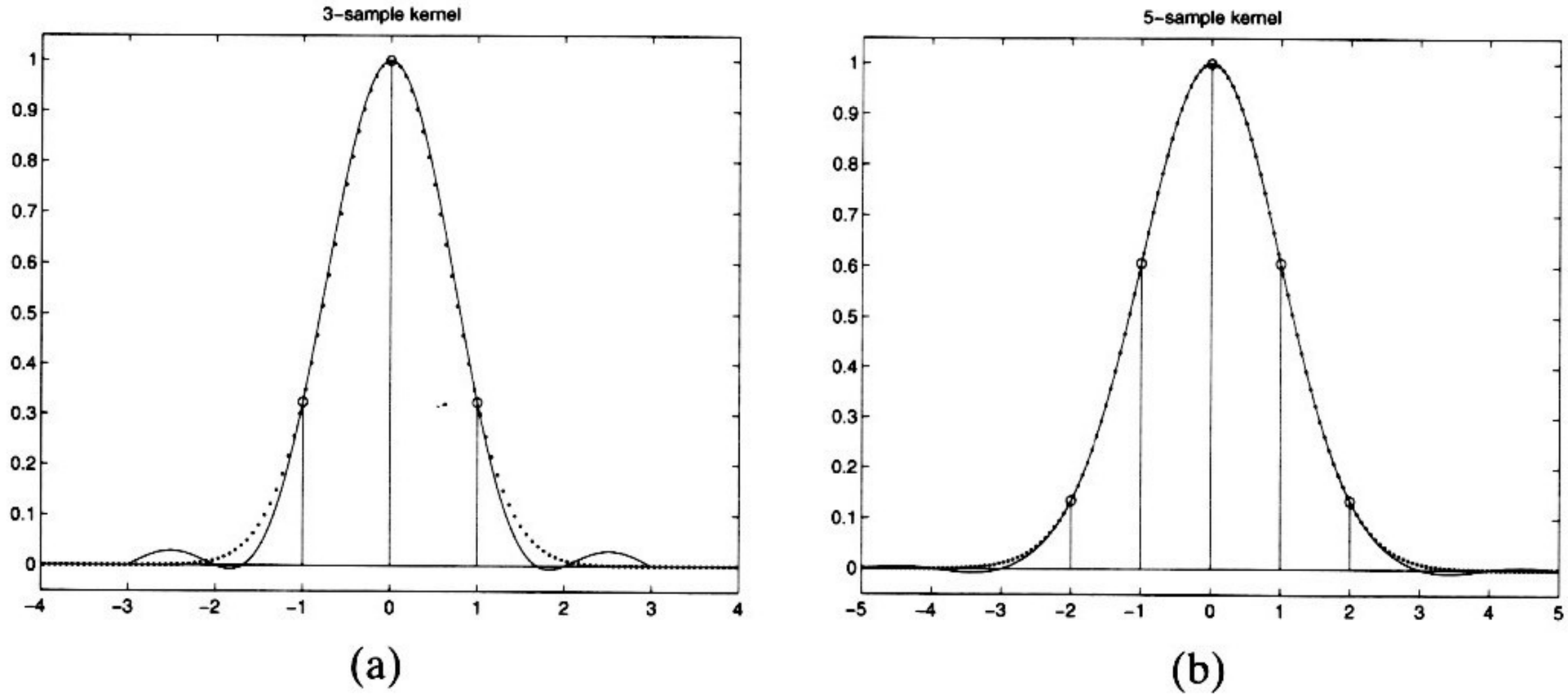


Figure 3.6 Continuous Gaussian kernels (dotted), sampled real kernels, and continuous kernels reconstructed from samples (solid), for $\sigma = 0.6$ ($w = 3$) (a) and $\sigma = 1$ ($w = 5$) (b) respectively.

Algorithm REP_AVG

Let $A * B$ indicate the convolution of matrices A and B. Let I be the input image. Define the 3×3 RA mask

$$R = \frac{1}{24}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.9}$$

To convolve I with an approximated Gaussian kernel of $\sigma = \sqrt{n/3}$:

1. $I_{res} = I$
2. For $i = 1, \ldots, n$, $I_{res} = I_{res} * R$

You might be tempted to combine separability and repeated averaging, as this would yield a very efficient algorithm indeed. But are you sure that the kernel defined in REP_AVG is separable? Using separability with a nonseparable kernel means that the result of REP_AVG is different from the application of the 2-D mask, which may result in errors in further processing; image differentiation is once again an apt example.

A safe way to combine separability and repeated averaging is *cascading*. The idea is that smoothing with Gaussian kernels of increasingly large standard deviations can also be achieved by convolving an image repeatedly with the same Gaussian kernel. In this way, each filtering pass of REP_AVG is surely separable (see Exercise 3.7).

3.2.4 Nonlinear Filtering

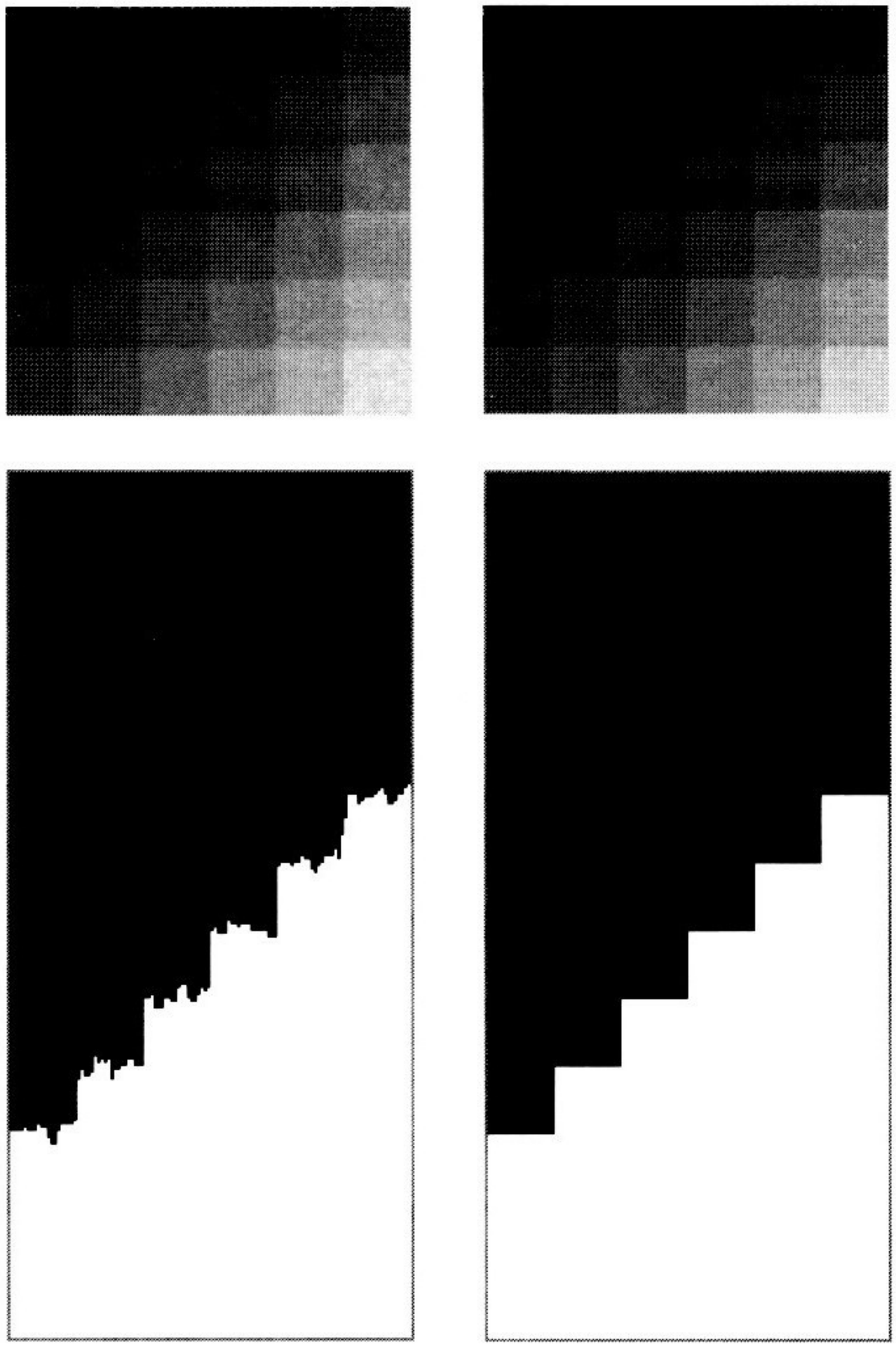
In section 3.2.1, we listed the main problems of the averaging filter: blur, poor feature localization, secondary lobes in the frequency domain, and incomplete suppression of peak noise. Gaussian filters solve only the third one, as the Fourier transform of a Gaussian has no secondary lobes. The remaining problems are tackled efficiently by *nonlinear filtering*; that is, filtering methods that cannot be modelled by convolution.

The *median filter* is a useful representative of this class. A median filter just replaces each pixel value $I(i, j)$ with the median of the values found in a local neighborhood of (i, j). As with averaging, the larger the neighborhood, the smoother the result.

Algorithm MED_FILTER

Let I be the input image, I_m the filtered image, and n an odd number.
For each pixel (i, j):

1. Compute the median $m(i, j)$ of the values in a $n \times n$ neighborhood of (i, j), $\{I(i + h, j + k), h, k \in [-n/2, n/2]\}$, where $n/2$ indicates integer division.
2. Assign $I_m(i, j) = m(i, j)$



(a) (b)

Figure 3.7 (a) Results of applying median filtering (3-pixel wide) to the "checkerboard" image corrupted by Gaussian noise, and grey-level profile along the same row of Figure 3.2. (b) Same for the "checkerboard" image corrupted by impulsive noise.

Figure 3.7 shows the effects of median filtering on the "checkerboard" image corrupted by Gaussian and impulsive noise (Figure 3.1 center and right, respectively). Compare these results with those obtained by Gaussian smoothing (Figure 3.2): Median filtering has suppressed impulsive noise completely. Contours are also blurred less by the median than by the Gaussian filter; therefore, a median filter preserves discontinuities better than linear, averaging filters.

3.3 Summary

After working through this chapter you should be able to:

- explain the concept of noise, image noise, and why noise smoothing is important for computer vision
- design noise-smoothing algorithms using Gaussian and median filtering
- decide whether it is appropriate to use linear or median smoothing filters in specific situations

3.4 Further Readings

Noise filtering and image restoration are classic topics of noise and image processing. Detailed discussions of image processing methods are found in several books; for instance, [4, 3, 10, 8]. Papoulis [7] is a good reference text for Fourier transforms.

Repeated averaging for computer vision was first reported by Brady *et al.* [1]. Cai [2] discusses several linear filtering methods in the context of diffusion smoothing. Witkin [11] and Lindeberg [5] provide a good introduction to *scale-space representations*, the study of image properties when smoothing with Gaussians of increasing standard deviation (the *scale parameter*). One reason for keeping multiple scales is that some image features may be lost after filtering with large kernels, but small kernels could keep in too much noise. Alternative methods for representing signals at multiple scales include pyramids [9] and wavelets [6] (see also references therein).

3.5 Review

Questions

- *3.1* Explain the concept of image noise, how it can be quantified, and how it can affect computer vision computations.
- *3.2* How would you estimate the quantization noise in a range image in terms of mean and standard deviation? Notice that this allows you to compare directly quantization and acquisition noise.
- *3.3* Explain why a non-negative kernel works as a low-pass filters, and in which assumptions it can suppress noise.
- *3.4* What is a separable kernel? What are the advantages of separability?
- *3.5* What are the problems of the mean filter for noise smoothing? Why and in what sense is Gaussian smoothing better?
- *3.6* Explain why the sampling accuracy of a 1-D Gaussian filter with $\sigma = 0.6$ cannot be improved using more than three spatial samples.
- *3.7* What is repeated averaging? What are its effects and benefits?
- *3.8* What is the difference between cascading and repeated averaging?
- *3.9* Can you think of any disadvantage of cascading? (*Hint:* Which standard deviations do you achieve?)