10 Pixel Processing

10.1 Introduction

After a digital image has been captured, the first preprocessing steps include two classes of operations, point operations and geometric operations. Essentially, these two types of operations modify the "what" and "where" of a pixel.

Point operations modify the gray values at individual pixels depending only on the gray value and possibly on the position of the pixels. Generally, such a kind of operation is expressed by

$$G'_{mn} = P_{mn}(G_{mn}). \tag{10.1}$$

The indices at the function P denote the possible dependency of the point operation on the position of the pixel.

In contrast, *geometric operations* modify only the position of a pixel. A pixel located at the position $\boldsymbol{x}$ is relocated to a new position $\boldsymbol{x}'$. The relation between the two coordinates is given by the geometric mapping function.

$$\boldsymbol{x}' = M(\boldsymbol{x}). \tag{10.2}$$

Point and geometric operations are complementary operations. They are useful for corrections of elementary distortions of the image formation process such as nonlinear and inhomogeneous radiometric responsivity of the imaging sensors or geometric distortions of the imaging system. We apply point operations to correct and optimize the image illumination, to detect underflow and overflow, to enhance and stretch contrast, to average images, to correct for inhomogeneous illumination, or to perform radiometric calibration (Sections 10.2.3–10.3.3).

Geometric operations include two major steps. In most applications, the mapping function Eq. (10.2) is not given explicitly but must be derived from the correspondences between the original object and its image (Section 10.4.4). When an image is warped by a geometric transform, the pixels in the original and warped images almost never fall onto each other. Thus, it is required to interpolate gray values at these pixels from neighboring pixels. This important task is discussed in detail in Section 10.5 because it is not trivial to perform accurate interpolation.

Point operations and geometric operations are not only of interest for elementary preprocessing steps. They are also an integral part of

B. Jähne, Digital Image Processing
ISBN 3-540-24035-7

many complex image operations, especially for feature extraction (Chapters 11–15). Note, however, that point operations and geometric operations are not suitable to correct the effects of an optical system described by its point spread function. This requires sophisticated reconstruction techniques that are discussed in Chapter 17. Point operations and geometric operations are limited to the performance of simple radiometric and geometric corrections.

10.2 Homogeneous Point Operations

10.2.1 Definitions and Basic Properties

If a point operation is independent of the position of the pixel, we call it a *homogeneous point operation* and write

$$G'_{mn} = P(G_{mn}). \tag{10.3}$$

A point operation maps the set of gray values onto itself. Generally, point operations are not invertible, as two different gray values may be mapped onto one. Thus, a point operation generally results in an irrecoverable loss of information. The point operation

$$P(q) = \begin{cases} 0 & q < t \\ Q-1 & q \geq t \end{cases}, \tag{10.4}$$

for example, performs a simple global threshold operation. All gray values below the threshold t are set to zero (black), all above and equal to the threshold to the highest value $Q-1$ (white). Consequently, this point operation cannot be inverted. An example for an invertible point operation is image negation. This operation computes an image with an inverted gray scale as with a photographic negative according to

$$P_N(q) = Q - 1 - q. \tag{10.5}$$

The inverse operation of a negation is another negation:

$$P_N\left(P_N(q)\right) = Q - 1 - (Q - 1 - q) = q. \tag{10.6}$$

Another example of an invertible point operation is the conversion between signed and unsigned representations of gray values (Section 2.2.5).

10.2.2 Look-up Tables

The direct computation of homogeneous point operations according to Eq. (10.3) may be very costly. This is demonstrated by the following example. The 14-bit gray scale of a 1024×1024 image from a high-resolution CCD camera is to be presented in an 8-bit logarithmic gray

scale covering 4.3 decades from 1 to 16 383. The following point operation performs this conversion:

$$P(q) = 59.30 \lg q \tag{10.7}$$

A straightforward implementation would require the following operations per pixel: an integer to double conversion, computation of the logarithm, multiplication by 59.30, and a double to 8-bit integer conversion. All these operations have to be repeated over a million times for a 1024×1024 image.

The key point for a more efficient implementation lies in the observation that the definition range of any point operation consists of only a limited number of Q quantization levels. For the 14-bit to 8-bit logarithmic conversion, we have at most 16 384 different input values. This means that most of the one million computations are just repeated, on average 64 times. We can avoid the unnecessary repetition by precalculating $P(q)$ for all 16 384 possible gray values and storing the computed values in a 16 384-element table. Then, the computation of the point operation is reduced to a replacement of the gray value by the element in the table with an index corresponding to the gray value.

Such a table is called a *look-up table* or *LUT*. Hence, homogeneous point operations are equivalent to *look-up table operations*. Look-up tables are more efficient the smaller the number of quantization levels. For standard 8-bit images, the tables contain just 256 values. But it is still efficient in most cases to use 65 536 entry look-up tables with 16-bit images.

In most image processing systems and frame grabbers, look-up tables are implemented in hardware. There are two possible places for look-up tables on frame grabber boards, as illustrated in Fig. 10.1. The *input LUT* is located between the *analog-digital converter* and the frame buffer. The *output LUT* is located between the frame buffer and the digital-analog converter for output of the image in the form of an analog video signal, e. g., to a monitor. The input LUT allows a point operation to be performed *before* the image is stored in the frame buffer. With the output LUT, a point operation can be performed and observed on the monitor. In this way, we can interactively perform point operations *without* modifying the stored image. Many modern frame grabbers no longer include a frame buffer. With the advent of fast peripheral bus systems (such as the PCI bus with a peak rate of 132 MB/s, see Section 1.7), digitized images can be transferred directly to the PC memory (Fig. 10.2). With such a frame grabber, image display is performed on the graphics board of the computer. Consequently, the frame grabber includes only an input look-up table.

The use of input LUTs is limited. Nonlinear LUT functions lead to missing gray values or map two consecutive values onto one (Fig. 10.3).

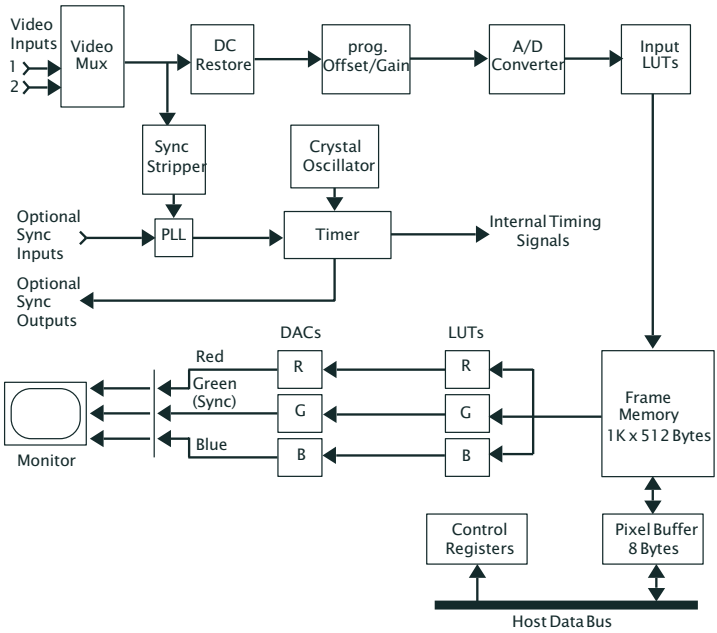

Figure 10.1: *Block diagram of the PCVISIONplus frame grabber from Imaging Technology, Inc. Look-up tables are located between the A/D converter and frame buffer (input LUT) and the frame buffer and display (output LUT).*

In this way, artifacts are introduced that yield enhanced errors in subsequent processing such as the computation of mean values and edge detection. It is obvious that especially the steepness of edges and the accuracy of gray value changes are affected.

Input LUTs would be valuable also for nonlinear point operations if the 8-bit input values were mapped to higher precision output values, e. g., 16-bit integers or 32-bit floating point numbers or if the camera signal is digitized with higher resolution, e. g., 12 bit and then output as 8-bit numbers. Then the error associated with rounding is significantly reduced. At the same time, the gray levels could be converted into a calibrated signal, e. g., a temperature for an infrared camera. Unfortunately, such generalized LUTs are not yet implemented in hardware. However, it is easy to realize them in software.

In contrast to the input LUT, the output LUT is a much more widely used tool, as it does not change the stored image. With LUT operations, we can also convert a gray value image into a *pseudo-color image*. Again, this technique is common even with the simplest frame grabber boards (Fig. 10.1). Not much additional hardware is needed. Three digital-analog converters are used for the primary colors red, green, and blue. Each channel has its own LUT with 256 entries for an 8-bit display. In this

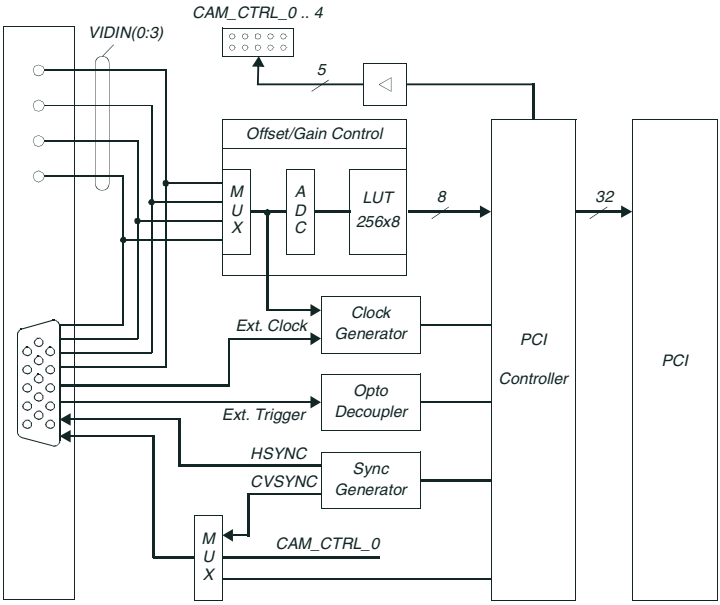

Figure 10.2: *Block diagram of the PCEYE_1 frame grabber from ELTEC Elektronik GmbH as an example of a modern PCI bus frame grabber without a frame buffer. The image data are transferred in realtime via direct memory access (DMA) to the memory of the PC for display and further processing.*

way, we can map each individual gray value q to any color by assigning a color triple to the corresponding LUT addresses $r(q)$, $g(q)$, and $b(q)$. Formally, this is a *vector point operation*

$$\boldsymbol{P}(q) = \begin{bmatrix} r(q) \\ g(q) \\ b(q) \end{bmatrix}. \tag{10.8}$$

When all three point functions $r(q)$, $g(q)$, and $b(q)$ are identical, a gray tone will be displayed. If two of the point functions are zero, the image will appear in the remaining color.

10.2.3 Interactive Gray Value Evaluation

Homogeneous point operators implemented via look-up tables are a very useful tool for inspecting images. As the look-up table operations work in real-time, images can be manipulated interactively. If only the output look-up table is changed, the original image content remains unchanged. Here, we demonstrate typical tasks.

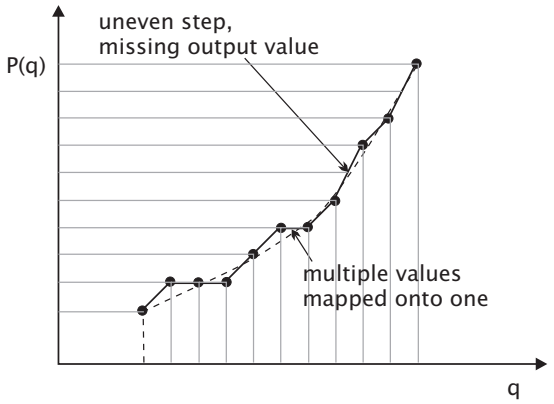

Figure 10.3: *Illustration of a nonlinear look-up table with mapping of multiple values onto one and missing output value leading to uneven steps.*

Evaluating and Optimizing Illumination. With the naked eye, we can hardly estimate the homogeneity of an illuminated area as demonstrated in Fig. 10.4a, b. A histogram reveals the gray scale distribution but not its spatial variation (Fig. 10.4c, d). Therefore, a histogram is not of much help for optimizing the illumination interactively. We need to mark gray scales such that absolute gray levels become perceivable for the human eye. If the radiance distribution is continuous, it is sufficient to use equidensities. This technique uses a staircase type of homogeneous point operation by mapping a certain range of gray scales onto one. This point operation is achieved by zeroing the p least significant bits with a logical *and* operation:

$$q' = P(q) = q \wedge \overline{(2^p - 1)}, \tag{10.9}$$

where $\wedge$ denotes the logical (bitwise) *and* and overlining denotes *negation*. This point operation limits the resolution to $Q - p$ bits and, thus, 2^{Q-p} quantization levels. Now, the jump between the remaining quantization levels is large enough to be perceived by the eye and we see contour lines of equal absolute gray scale in the image (Fig. 10.4). We can now try to homogenize the illumination by making the distance between the contour lines as large as possible.

Another way to mark absolute gray values is the so-called *pseudocolor image* that has already been discussed in Section 10.2.2. With this technique, a gray level q is mapped onto an RGB triple for display. As color is much better recognized by the eye, it helps reveal absolute gray levels.

Detection of Underflow and Overflow. Under- and overflows of the gray values of a digitized image often go unnoticed and cause a serious

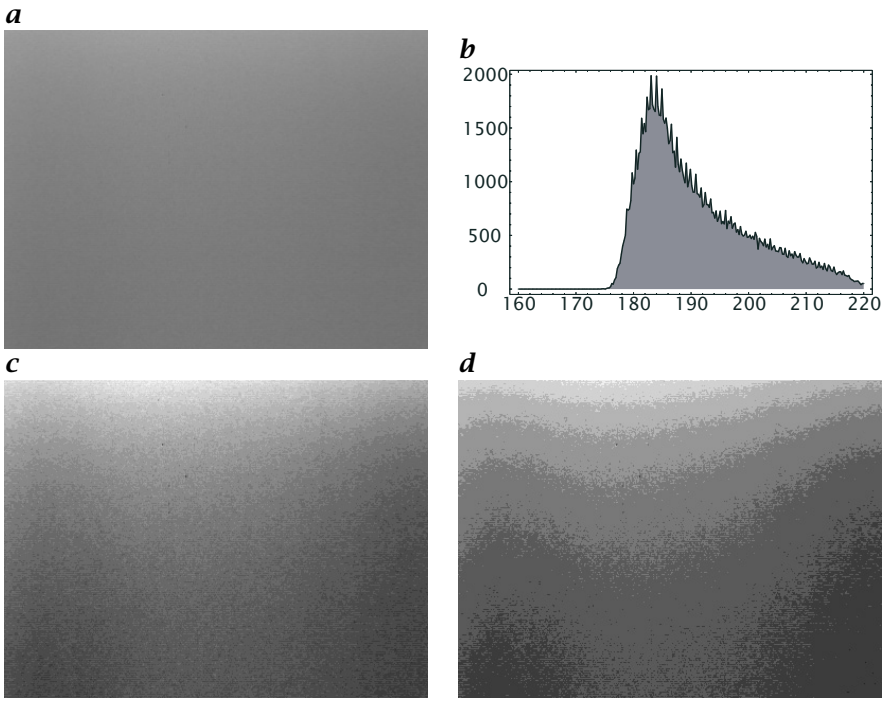

Figure 10.4: ***a*** *The irradiance is gradually decreasing from the top to the bottom, which is almost not recognized by the eye. The gray scale of this floating-point image computed by averaging over 100 images ranges from 160 to 200.* ***b*** *Histogram of* ***a****;* ***c*** *and* ***d*** *(contrast enhanced, gray scale 184–200): Edges artificially produced by a staircase LUT with a step height of 1.0 and 2.0 make contours of constant irradiance easily visible.*

bias in further processing, for instance for mean gray values of objects or the center of gravity of an object. In most cases, such areas cannot be detected directly. They may only become apparent in textured areas when the texture is bleached out. Over and underflow are detected easily in histograms by strong peaks at the minimum and/or maximum gray values (Fig. 10.5). With pseudocolor mapping, the few lowest and highest gray values could be displayed, for example, in blue and red, respectively. Then, gray values dangerously close to the limits immediately pop out of the image and can be avoided by correcting the illumination lens aperture or gain of the video input circuit of the frame grabber.

Contrast enhancement. Because of poor illumination conditions, it often happens that images are underexposed. Then, the image is too dark and of low contrast (Fig. 10.6a). The histogram (Fig. 10.6b) shows that the image contains only a small range of gray values at low gray values.

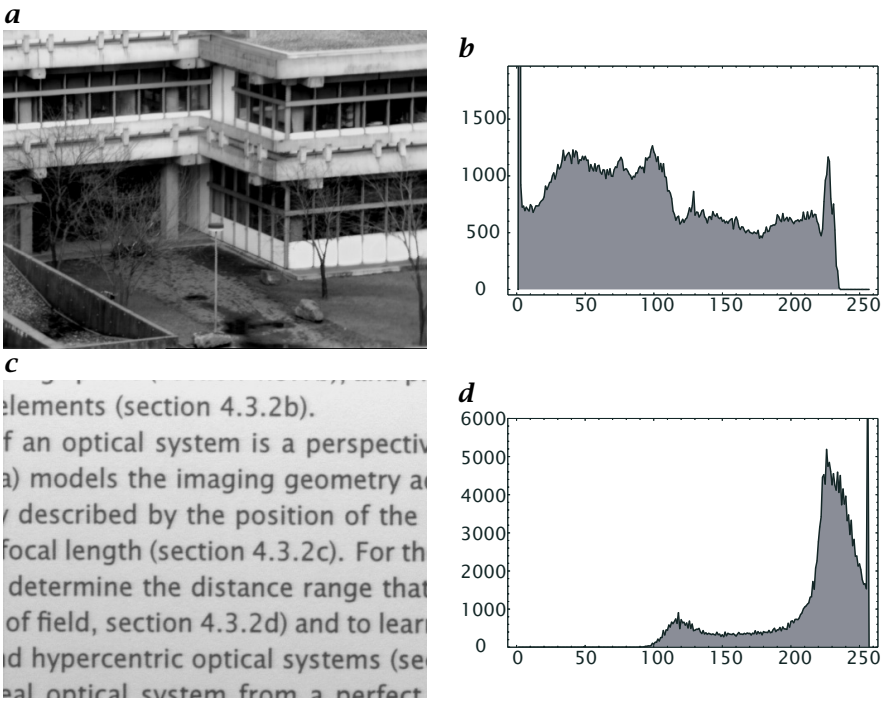

Figure 10.5: *Detection of underflow and overflow in digitized images by histograms;* ***a*** *image with underflow and* ***b*** *its histogram;* ***c*** *image with overflow and* ***d*** *its histogram.*

The appearance of the image improves considerably if we apply a point operation which maps a small grayscale range to the full contrast range (for example with this operation: $q' = 4q$ for $q < 64$, and $q' = 255$ for $q \geq 64$) (Fig. 10.6c). We only improve the appearance of the image but not the image *quality* itself. The histogram shows that the gray value resolution is still the same (Fig. 10.6d).

The image quality can be improved. The best way is to increase the object irradiance by using a more powerful light source or a better design of the illumination setup. If this is not possible, we can still increase the gain of the analog video amplifier. All modern image processing boards include an amplifier whose gain and offset can be set by software (see Figs. 10.1 and 10.2). By increasing the gain, the brightness and resolution of the image improve, but only at the expense of an increased noise level (Section 3.4.5).

Contrast Stretching. It is often of interest to analyze faint irradiance differences which are beyond the resolution of the human visual system or the display equipment used. This is especially the case if images are

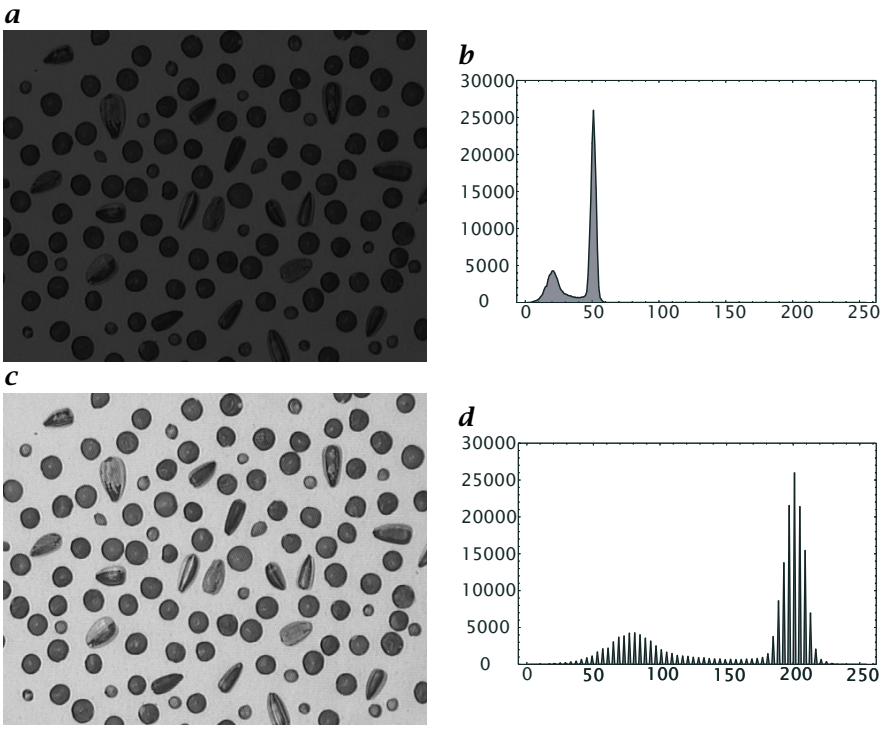

Figure 10.6: *Contrast enhancement;* ***a*** *underexposed image and* ***b*** *its histogram;* ***c*** *interactively contrast enhanced image and* ***d*** *its histogram.*

printed. In order to observe faint differences, we stretch a small gray scale range of interest to the full range available. All gray values outside this range are set to the minimum or maximum value. This operation requires that the gray values of the object of interest fall into the range selected for contrast stretching. An example of contrast stretching is shown in Fig. 10.7a,b. The wedge at the bottom of the images, ranging from 0 to 255, directly shows which part of the gray scale range is contrast enhanced.

Range Compression. In comparison to the human visual system, a digital image has a considerably smaller dynamical range. If a minimum resolution of 10% is demanded, the gray values must not be lower than 10. Therefore, the maximum dynamical range in an 8-bit image is only $255/10 \approx 25$. The low contrast range of digital images makes them appear of low quality when high-contrast scenes are encountered. Either the bright parts are bleached or no details can be recognized in the dark parts. The dynamical range can be increased by a transform that was introduced in Section 2.2.6 as the *gamma transform*. This nonlinear

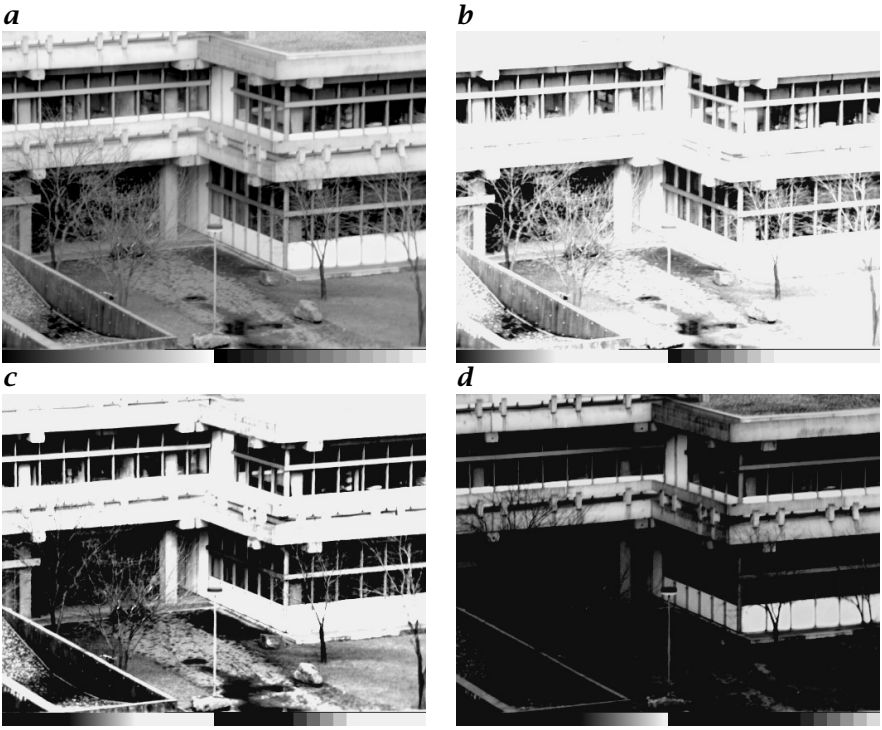

Figure 10.7: b–d *Contrast stretching of the image shown in* ***a****. The stretched range can be read from the transformation of the gray scale wedge at the bottom of the image.*

homogeneous point operation has the form

$$q' = \frac{255}{255^{\gamma}} q^{\gamma}. \tag{10.10}$$

The factors in Eq. (10.10) are chosen such that a range of $[0, 255]$ is mapped onto itself. This transformation allows a larger dynamic range to be recognized at the cost of resolution in the bright parts of the image. The dark parts become brighter and show more details. This contrast transformation is better adapted to the logarithmic characteristics of the human visual system. An image presented with different gamma factors is shown in Fig. 10.8.

Noise Variance Equalization. From Section 3.4.5, we know that the variance of the noise generally depends on the image intensity according to

$$\sigma_g^2(g) = \sigma_0^2 + Kg. \tag{10.11}$$

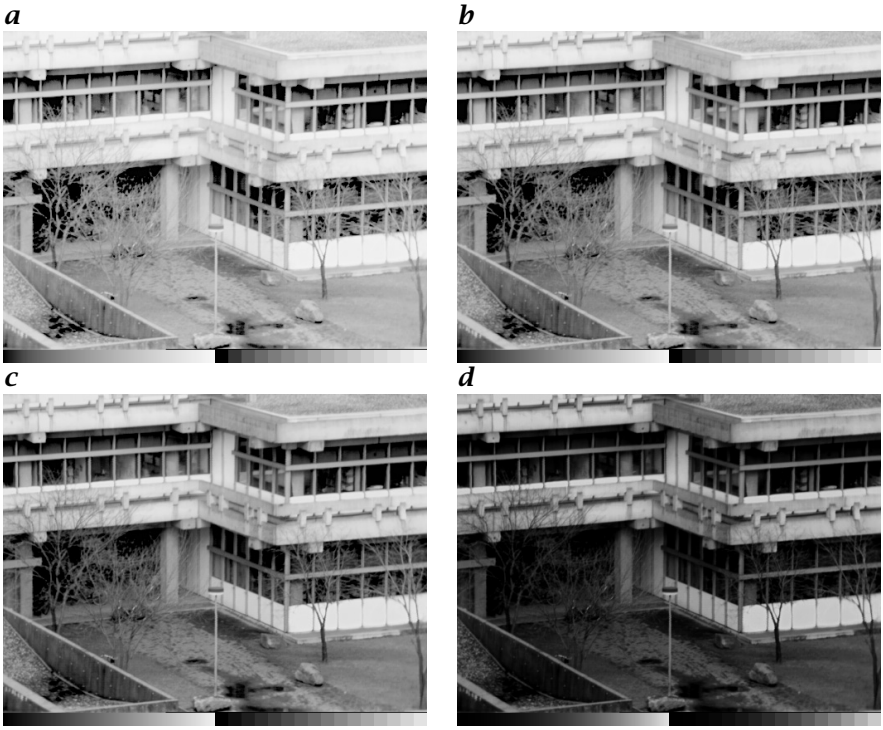

Figure 10.8: *Presentation of an image with different gamma values:* ***a*** *0.5,* ***b*** *0.7,* ***c*** *1.0, and* ***d*** *2.0.*

A statistical analysis of images and image operations is, however, much easier if the noise is independent of the gray value. Only then all the error propagation techniques discussed in Section 3.3.3 are valid.

Thus we need to apply a nonlinear gray value transform $h(g)$ in such a way that the noise variance becomes constant. To first order, the variance of $h(g)$ is

$$\sigma_h^2 \approx \left(\frac{\mathrm{d}h}{\mathrm{d}g}\right)^2 \sigma_g^2(g) \tag{10.12}$$

according to Eq. (3.36) [53]. If we set σ_h^2 to be constant, we obtain

$$\mathrm{d}h = \frac{\sigma_h}{\sqrt{\sigma^2(g)}}\mathrm{d}g.$$

Integration yields

$$h(g) = \sigma_h \int_0^g \frac{\mathrm{d}g'}{\sqrt{\sigma^2(g')}} + C. \tag{10.13}$$

With the linear variance function Eq. (10.11), the integral in Eq. (10.13) yields

$$h(g) = \frac{2\sigma_h}{K}\sqrt{\sigma_0^2 + Kg} + C. \tag{10.14}$$

We use the two free parameters σ_h and C to map the values of h into the interval $[0, \gamma g_{\max}]$. This implies the conditions $h(0) = 0$ and $h(g_{\max}) = g_{\max}$ and we obtain

$$h(g) = \gamma g_{\max}\frac{\sqrt{\sigma_0^2 + Kg} - \sigma_0}{\sqrt{\sigma_0^2 + Kg_{\max}} - \sigma_0}, \quad \sigma_h = \frac{\gamma K g_{\max}/2}{\sqrt{\sigma_0^2 + Kg_{\max}} - \sigma_0}. \tag{10.15}$$

The nonlinear transform becomes particularly simple for an ideal imaging sensor with $\sigma_0 = 0$. Then a square root transform must be applied to obtain an intensity independent noise variance

$$h(g) = \gamma\sqrt{g g_{\max}} \quad \text{and} \quad \sigma_h = \frac{\gamma}{2}\sqrt{K g_{\max}}. \tag{10.16}$$

10.3 Inhomogeneous Point Operations

Homogeneous point operations are only a subclass of point operators. In general, a point operation depends also on the position of the pixel in the image. Such an operation is called an *inhomogeneous point operation*. Inhomogeneous point operations are mostly related to calibration procedures. Generally, the computation of an inhomogeneous point operation is much more time consuming than the computation of a homogeneous point operation. We cannot use look-up tables since the point operation depends on the pixel position and we are forced to calculate the function for each pixel.

The subtraction of a background image without objects or illumination is a simple example of an inhomogeneous point operation, which is written as:

$$g'_{mn} = P_{mn}(g_{mn}) = g_{mn} - b_{mn}, \tag{10.17}$$

where b_{mn} is the background image.

10.3.1 Image Averaging

One of the simplest inhomogeneous point operations is *image averaging*. In a number of imaging applications high noise levels occur. Prominent examples include *thermal imaging* (Section 6.4.1) and all applications where only a limited number of photons are collected (see Fig. 3.2 and Section 3.4.5).

Figure 10.9a shows the temperature differences at the water surface of a wind-wave facility cooled at 1.8 m/s wind speed by evaporation. Because of a substantial noise level, the small temperature fluctuations can

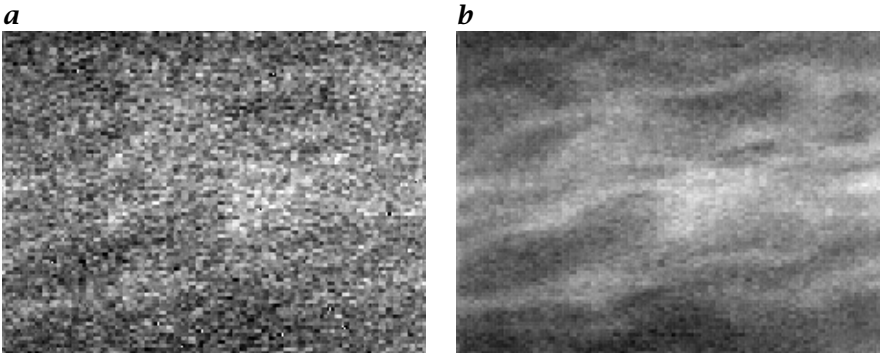

Figure 10.9: *Noise reduction by image averaging:* ***a*** *single thermal image of small temperature fluctuations on the water surface cooled by evaporation;* ***b*** *same, averaged over 16 images; the full gray value range corresponds to a temperature range of 1.1 K.*

hardly be detected. Taking the mean over several images significantly reduces the noise level (Fig. 10.9b).

The error of the mean (Section 3.3.3) taken from N samples is given by

$$\sigma_{\overline{G}}^2 \approx \frac{1}{(K-1)}\sigma_G^2 = \frac{1}{K(K-1)}\sum_{k=0}^{K-1}(\boldsymbol{G}_k - \overline{\boldsymbol{G}})^2. \tag{10.18}$$

If we take the average of K images, the noise level is reduced by $1/\sqrt{K}$ compared to a single image. Taking the mean over 16 images thus reduces the noise level by a factor of four. Equation (10.18) is only valid, however, if the standard deviation σ_g is significantly larger than the standard deviation related to the quantization (Section 9.5).

10.3.2 Correction of Inhomogeneous Illumination

Every real-world application has to contend with *uneven illumination* of the observed scene. Even if we spend a lot of effort optimizing the illumination setup, it is still very hard to obtain perfectly even object irradiance. A nasty problem is caused by small dust particles in the optical path, especially on the glass window close to the CCD sensor. Because of the distance of the window from the imager, these particles — if they are not too large — are blurred to such an extent that they are not directly visible. But they still absorb some light and thus cause a drop in the illumination level in a small area. These effects are not easily visible in a scene with high contrast and many details, but become very apparent in the case of a uniform background (Fig. 10.4a and b). Some imaging sensors, especially cheap CMOS sensors, also show a considerable uneven

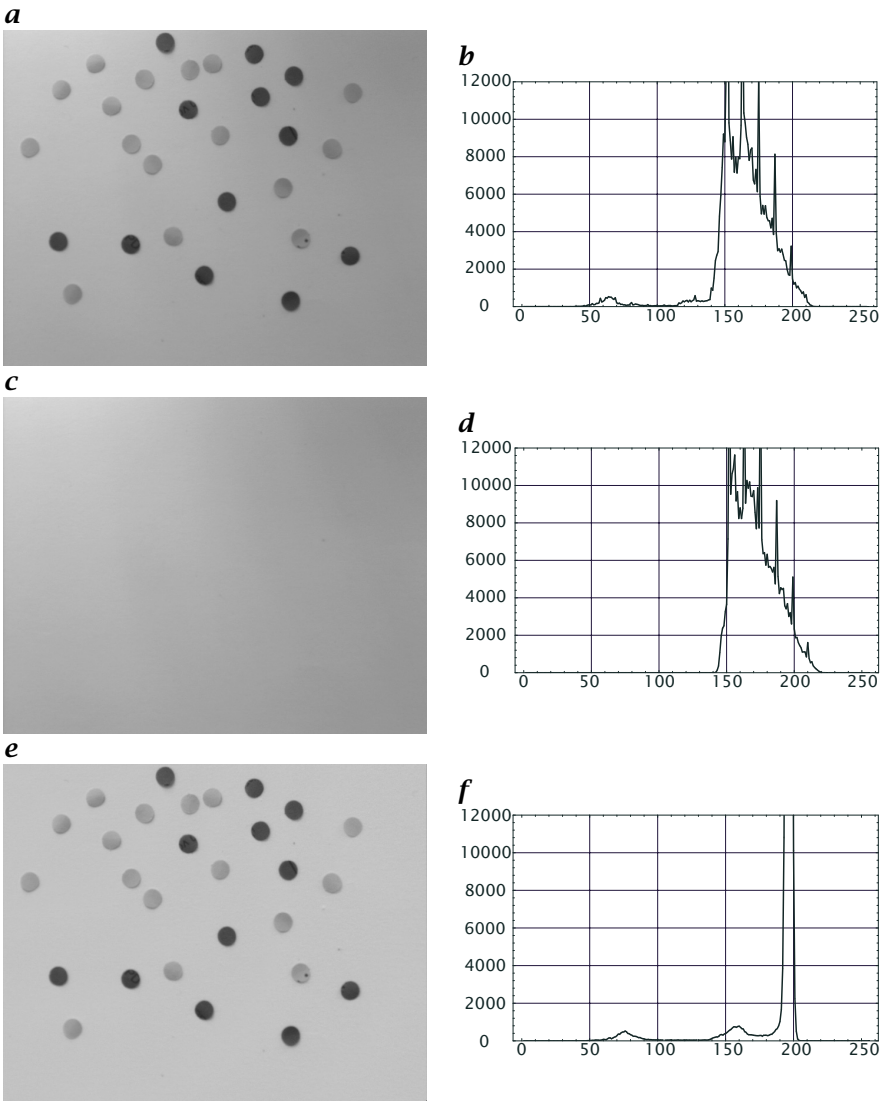

Figure 10.10: *Correction of uneven illumination with an inhomogeneous point operation:* ***a*** *original image and* ***b*** *its histogram;* ***c*** *background image and* ***d*** *its histogram;* ***e*** *division of the image by the background image and* ***f*** *its histogram.*

sensitivity of the individual photoreceptors, which adds to the nonuniformity of the image. These distortions could severely limit the quality of the images. These effects make it more difficult to separate an object from the background, and introduce systematic errors for subsequent image processing steps.

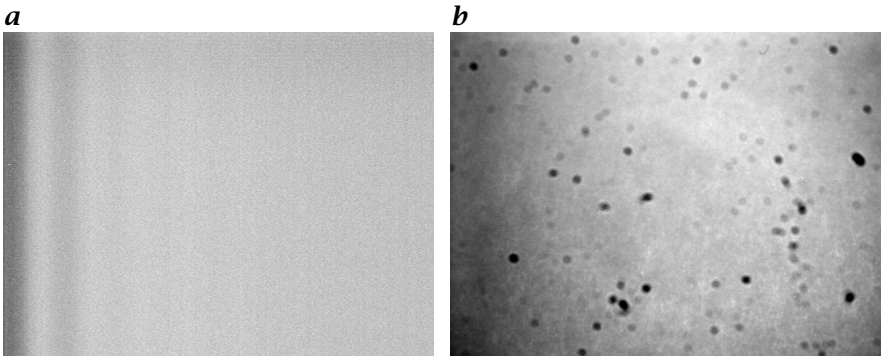

Figure 10.11: *Contrast-enhanced* ***a*** *dark image and* ***b*** *reference image for a two-point radiometric calibration of a CCD camera with analog video output.*

Nevertheless, it is possible to correct these effects if we know the nature of the distortion and can take suitable reference images. In the following, we study two cases. In the first, we assume that the gray value in the image is a product of the inhomogeneous irradiance and the reflectivity or transmissivity of the object. Furthermore, we assume that we can take a reference image without absorbing objects or with an object of constant reflectivity. A reference image can also be computed, when small objects are randomly distributed in the image. Then, it is sufficient to compute the average image from many images with the objects. The inhomogeneous illumination can then be corrected by dividing the image by the reference image:

$$\boldsymbol{G}' = c \cdot \boldsymbol{G}/\boldsymbol{R}. \tag{10.19}$$

The constant c is required to represent the normalized image with integer numbers again. If the objects absorb light, the constant c is normally chosen to be close to the maximum integer value. Figure 10.10e demonstrates that an effective suppression of inhomogeneous illumination is possible using this simple method.

10.3.3 Two-Point Radiometric Calibration

The simple ratio imaging described above is not applicable if also a nonzero inhomogeneous background has to be corrected for, as caused, for instance, by the fixed pattern noise of a CCD sensor. In this case, two reference images are required. This technique is also applied for a simple *two-point radiometric calibration* of an imaging sensor with a linear response. Some image measuring tasks require an absolute or relative radiometric calibration. Once such a calibration is obtained, we can infer the radiance of the objects from the measured gray values.

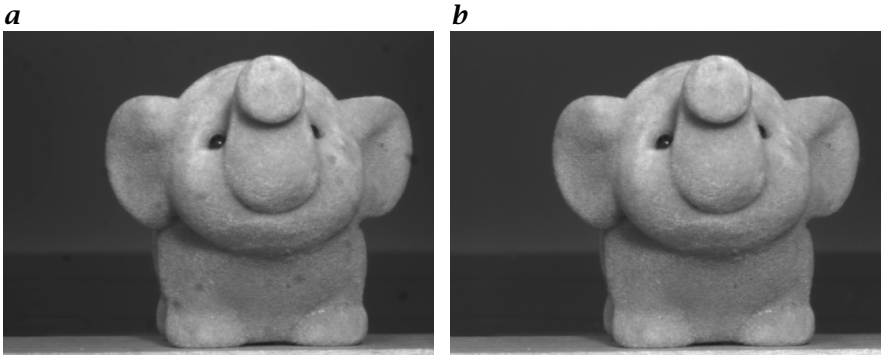

Figure 10.12: *Two-point radiometric calibration with the dark and reference image from Fig. 10.11:* ***a*** *original image and* ***b*** *calibrated image; in the calibrated image the dark spots caused by dust are no longer visible.*

First, we take a dark image $\boldsymbol{B}$ without any illumination. Second, we take a reference image $\boldsymbol{R}$ with an object of constant radiance, e. g., by looking into an *integrating sphere*. Then, a normalized image corrected for both the fixed pattern noise and inhomogeneous sensitivity is given by

$$\boldsymbol{G}' = c\frac{\boldsymbol{G}-\boldsymbol{B}}{\boldsymbol{R}-\boldsymbol{B}}. \tag{10.20}$$

Fig. 10.11 shows a contrast-enhanced dark image and reference image of a CCD camera with analog output. Typical signal distortions can be observed. The signal oscillation at the left edge of the dark image results from an electronic interference, while the dark blobs in the reference image are caused by dust on the glass window in front of the sensor. The improvement due to the radiometric calibration can clearly be seen in Fig. 10.12.

10.3.4 Nonlinear Radiometric Calibration

Sometimes, the quantity to be measured by an imaging sensor is related in a nonlinear way to the measured gray value. An obvious example is thermography. In such cases a *nonlinear radiometric calibration* is required. Here, the temperature of the emitted object is determined from its radiance using Planck's equations (Section 6.4.1).

We will give a practical calibration procedure for ambient temperatures. Because of the nonlinear relation between radiance and temperature, a simple two-point calibration with linear interpolation is not sufficient. Haußecker [71] showed that a quadratic relation is accurate enough for a small temperature range, say from 0 to 40° centigrade. Therefore, three calibration temperatures are required, which are provided by a temperature-regulated blackbody calibration unit.

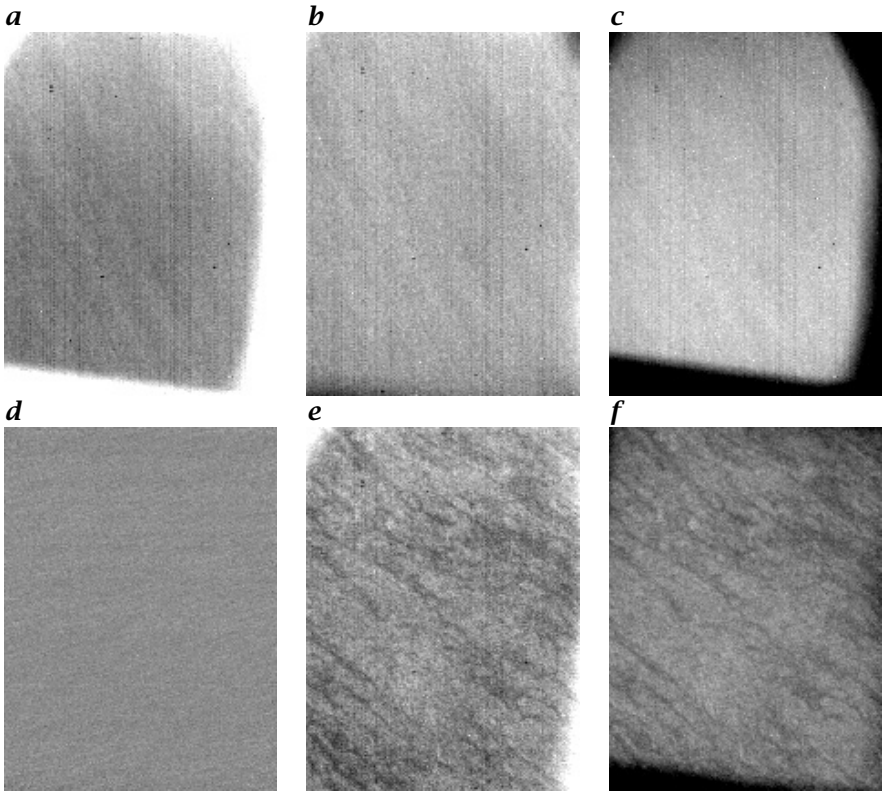

Figure 10.13: *Three-point calibration of infrared temperature images:* ***a–c*** *show images of calibration targets made out of aluminum blocks at temperatures of 13.06, 17.62, and 22.28°centigrade. The images are stretched in contrast to a narrow range of the 12-bit digital output range of the infrared camera:* ***a****: 1715–1740,* ***b****: 1925–1950,* ***c****: 2200–2230, and show some residual inhomogeneities, especially vertical stripes.* ***d*** *Calibrated image using the three images* ***a–c*** *with quadratic interpolation.* ***e*** *Original and* ***f*** *calibrated image of the temperature microscale fluctuations at the ocean surface (area about* $0.8 \times 1.0\,m^2$*).*

The calibration delivers three calibration images $\boldsymbol{G}_1$, $\boldsymbol{G}_2$, and $\boldsymbol{G}_3$ with known temperatures T_1, T_2, and T_3. The temperature image $\boldsymbol{T}$ of an arbitrary image $\boldsymbol{G}$ can be computed by quadratic interpolation as

$$\boldsymbol{T} = \frac{\Delta\boldsymbol{G}_2 \cdot \Delta\boldsymbol{G}_3}{\Delta\boldsymbol{G}_{21} \cdot \Delta\boldsymbol{G}_{31}} T_1 - \frac{\Delta\boldsymbol{G}_1 \cdot \Delta\boldsymbol{G}_3}{\Delta\boldsymbol{G}_{21} \cdot \Delta\boldsymbol{G}_{32}} T_2 + \frac{\Delta\boldsymbol{G}_1 \cdot \Delta\boldsymbol{G}_2}{\Delta\boldsymbol{G}_{31} \cdot \Delta\boldsymbol{G}_{32}} T_3, \tag{10.21}$$

with

$$\Delta\boldsymbol{G}_k = \boldsymbol{G} - \boldsymbol{G}_k \qquad \text{and} \qquad \Delta\boldsymbol{G}_{kl} = \boldsymbol{G}_k - \boldsymbol{G}_l. \tag{10.22}$$

The symbol $\cdot$ indicates pointwise multiplication of the images in order to distinguish it from matrix multiplication. Figure 10.13a, b, and c shows

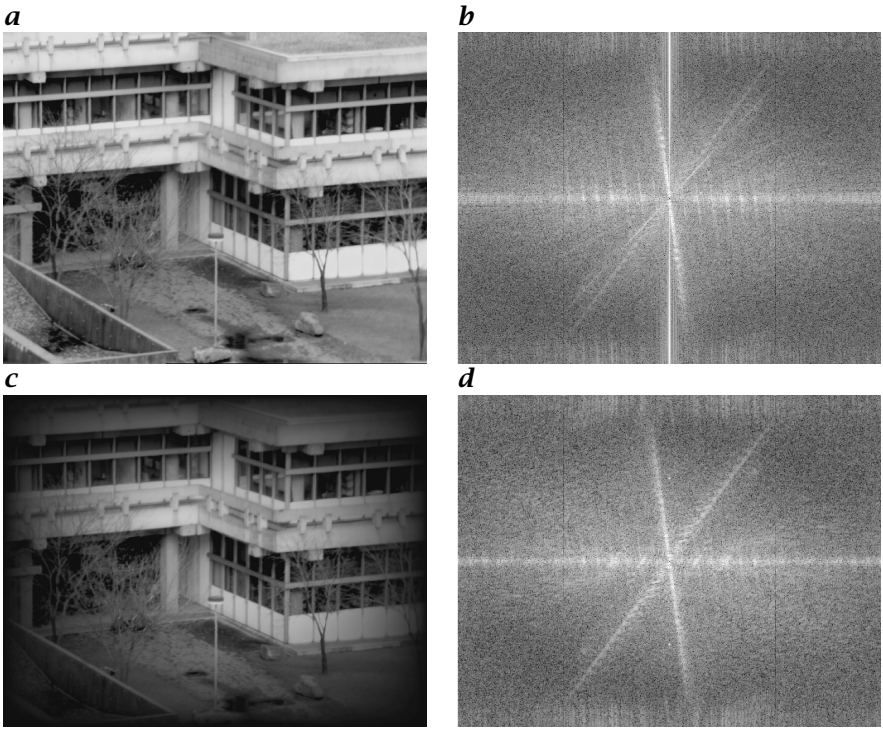

Figure 10.14: *Effect of windowing on the discrete Fourier transform:* ***a*** *original image;* ***b*** *DFT of* ***a*** *without using a window function;* ***c*** *image multiplied with a cosine window;* ***d*** *DFT of* ***c*** *using a cosine window.*

three calibration images. The infrared camera looks at the calibration target via a mirror, which limits the field of view at the edges of the images. This is the reason for the sharp temperature changes seen at the image borders in Fig. 10.13a, c. The calibration procedure removes the residual inhomogeneities (Fig. 10.13d, f) that show up in the original images.

10.3.5 Windowing

Another important application of inhomogeneous point operations is an operation known as *windowing*. Before we can calculate the DFT of an image, the image must be multiplied with a *window function*. If we omit this step, the spectrum will be distorted by the convolution of the image spectrum with the Fourier transform of the box function, the sinc function (see Section 2.3, ≻R5), which causes spectral peaks to become star-like patterns along the coordinate axes in Fourier space (Fig. 10.14b). We can also explain these distortions with the periodic

repetition of finite-area images, an effect that is discussed in conjunction with the sampling theorem in Section 9.2.3. The periodic repetition leads to discontinuities at the horizontal and vertical edges of the image which cause correspondingly high spectral densities along the x and y axes in the Fourier domain.

In order to avoid these distortions, we must multiply the image with a window function that gradually approaches zero towards the edges of the image. An optimum window function should preserve a high spectral resolution and show minimum distortions in the spectrum, that is, its DFT should fall off as fast as possible. These are contradictory requirements. A good spectral resolution requires a broad window function. Such a window, however, falls off steeply at the edges, causing a slow fall-off of the side lobes of its spectrum.

A carefully chosen window is crucial for a spectral analysis of time series [131, 148]. However, in digital image processing it is less critical because of the much lower dynamic range of the gray values. A simple cosine window

$$W_{mn} = \sin\left(\frac{\pi m}{M}\right)\sin\left(\frac{\pi n}{N}\right), \quad 0 \le m < M,\ 0 \le n < N \tag{10.23}$$

performs this task well (Fig. 10.14c,d).

A direct implementation of the windowing operation is very time consuming because we would have to calculate the cosine function $2MN$ times. It is much more efficient to perform the calculation of the window function once, to store the window image, and to use it then for the calculation of many DFTs. The storage requirements can be reduced by recognizing that the window function Eq. (10.23) is separable, i. e., a product of two functions $W_{m,n} = {}^{c}w_m \cdot {}^{r}w_n$. Then, we need to calculate only the M plus N values for the column and row functions ${}^{c}w_m$ and ${}^{r}w_n$, respectively. As a result, it is sufficient to store only the row and column functions. The reduced storage space comes at the expense of an additional multiplication per pixel for the window operation.

10.4 Geometric Transformations

In the remaining part of this chapter, we discuss geometric operations as the complementary operations to point operations. First we discuss elementary geometric transforms such as the affine transform (Section 10.4.2), the perspective transform (Section 10.4.3), and how to obtain the transformation parameters by point matching methods. Then we focus in Section 10.5 on interpolation, which arises as the major problem for a fast and accurate implementation of geometric operations on discrete images. Finally, in Section 10.6.3 we briefly discuss fast algorithms for geometric transforms.

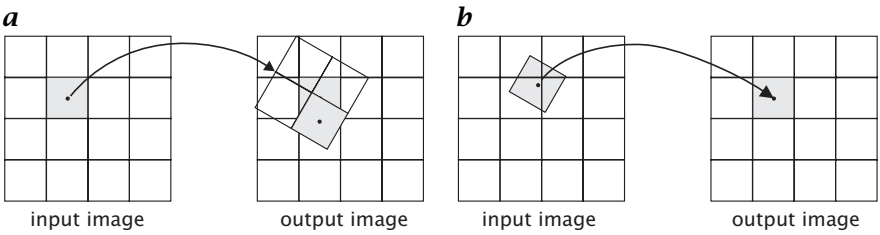

Figure 10.15: *Illustration of* ***a*** *forward mapping and* ***b*** *inverse mapping for spatial transformation of images.*

10.4.1 Forward and Inverse Mapping

Geometric transforms define the relationship between the points in two images. This relation can be expressed in two ways. Either the coordinates of the output image, $\boldsymbol{x}'$, can be specified as a function of the input coordinates, $\boldsymbol{x}$, or vice versa:

$$\boldsymbol{x}' = M(\boldsymbol{x}) \quad \text{or} \quad \boldsymbol{x} = M^{-1}(\boldsymbol{x}'), \tag{10.24}$$

where M specifies the mapping function and M^{-1} its inverse. The two expressions in Eq. (10.24) give rise to two principal kinds of spatial transformation: *forward mapping* and *inverse mapping*.

With *forward mapping*, a pixel of the input image is mapped onto the output image (Fig. 10.15a). Generally, a pixel of the input image lies in-between the pixels of the output image. With forward mapping, it is not appropriate just to assign the value of the input pixel to the nearest pixel in the output image (point-to-point or nearest neighbor mapping). Then, it may happen that the transformed image contains holes as a value is never assigned to a pixel in the output image or that a value is assigned more than once to a point in the output image. An appropriate technique distributes the value of the input pixel to several output pixels. The easiest procedure is to regard pixels as squares and to take the fraction of the area of the input pixel that covers the output pixel as the weighting factor. Each output pixel accumulates the corresponding fractions of the input pixels which — if the mapping is continuous — add up to cover the whole output pixel.

With *inverse mapping*, the coordinates of a point in the output image are mapped back onto the input image (Fig. 10.15b). It is obvious that this scheme avoids holes and overlaps in the output image as all pixels are scanned sequentially. Now, the interpolation problem occurs in the input image. The coordinates of the output image in general do not hit a pixel in the input image but lie in between the pixels. Thus, its correct value must be interpolated from the surrounding pixels. Generally, inverse mapping is a more flexible technique, as it is easier to implement various types of interpolation techniques.

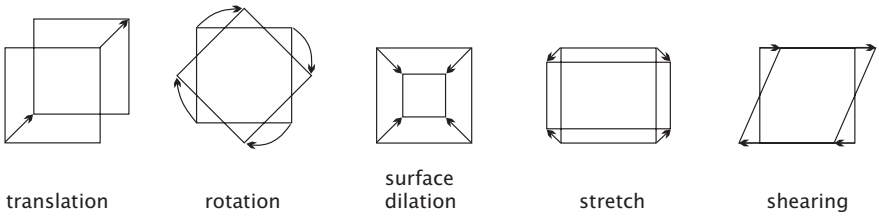

Figure 10.16: *Elementary geometric transforms for a planar surface element: translation, rotation, dilation, stretching, and shearing.*

10.4.2 Affine Transform

An affine transform is a linear coordinate transformation that includes the elementary transformations *translation*, *rotation*, *scaling*, *stretching*, and *shearing* (Fig. 10.16) and can be expressed by vector addition and matrix multiplication.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \tag{10.25}$$

With *homogeneous coordinates* (Section 7.7), the affine transform is written with a single matrix multiplication as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{10.26}$$

An affine transform has six degrees of freedom: two for translation (t_x, t_y) and one each for rotation, scaling, stretching, and shearing (a_{11}, a_{12}, a_{21}, and a_{22}). The affine transform maps a triangle into a triangle and a rectangle into a parallelogram. Therefore, it is also referred to as *three-point mapping*. Thus, it is obvious that the use of the affine transform is restricted. More general distortions such as the mapping of a rectangle into an arbitrary quadrilateral are not affine transforms.

10.4.3 Perspective Transform

Perspective projection is the basis of optical imaging as discussed in Section 7.3. The affine transform corresponds to parallel projection and can only be used as a model for optical imaging in the limit of a small field of view. The general perspective transform is most conveniently written with *homogeneous coordinates* (Section 7.7) as

$$\begin{bmatrix} w'x' \\ w'y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \quad \text{or} \quad X' = PX. \tag{10.27}$$

The two additional coefficients, a_{31} and a_{32}, not present in the affine transform Eq. (10.26), describe the perspective projection (compare Eq. (7.61) in Section 7.7).

Written in standard coordinates, the perspective transform according to Eq. (10.27) reads

$$\begin{aligned} x' &= \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + 1} \\ y' &= \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + 1}. \end{aligned} \tag{10.28}$$

In contrast to the affine transform, the perspective transform is nonlinear. However, it is reduced to a linear transform by using homogeneous coordinates. A perspective transform maps lines into lines but only lines parallel to the projection plane remain parallel. A rectangle is mapped into an arbitrary quadrilateral. Therefore, the perspective transform is also referred to as *four-point mapping*.

10.4.4 Determination of Transform Coefficients by Point Matching

Generally, the coefficients of a transform, as described in Sections 10.4.2 and 10.4.3, are not known. Instead we have a set of corresponding points between the object and image space. In this section, we learn how to infer the coefficients of a transform from sets of corresponding points. For an affine transform, we need three non-collinear points (to map a triangle into a triangle). With these three points, Eq. (10.26) results in the following linear equation system:

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \tag{10.29}$$

or

$$\boldsymbol{P}' = \boldsymbol{A}\boldsymbol{P} \tag{10.30}$$

from which $\boldsymbol{A}$ can be computed as

$$\boldsymbol{A} = \boldsymbol{P}'\boldsymbol{P}^{-1}. \tag{10.31}$$

The inverse of the matrix $\boldsymbol{P}$ exists when the three points $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ are linearly independent. This means geometrically that they must not lie on one line.

With more than three corresponding points, the parameters of the affine transform can be solved by the following equation system in a

least square sense (Section 17.4):

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{P}'\boldsymbol{P}^T(\boldsymbol{P}\boldsymbol{P}^T)^{-1} \quad \text{with} \\ \boldsymbol{P}'\boldsymbol{P}^T &= \begin{bmatrix} \sum x'_n x_n & \sum x'_n y_n & \sum x'_n \\ \sum y'_n x_n & \sum y'_n y_n & \sum y'_n \\ \sum x_n & \sum y_n & N \end{bmatrix} \\ \boldsymbol{P}\boldsymbol{P}^T &= \begin{bmatrix} \sum x_n^2 & \sum x_n y_n & \sum x_n \\ \sum x_n y_n & \sum y_n^2 & \sum y_n \\ \sum x_n & \sum y_n & N \end{bmatrix}. \end{aligned} \tag{10.32}$$

The inverse of an affine transform is itself affine. The transformation matrix of the inverse transform is given by the inverse of A^{-1}.

The determination of the coefficients for the perspective projection is slightly more complex. Given four or more corresponding points, the coefficients of the perspective transform can be determined. To that end, we rewrite Eq. (10.28) as

$$\begin{aligned} x' &= a_{11}x + a_{12}y + a_{13} - a_{31}xx' - a_{32}yx' \\ y' &= a_{21}x + a_{22}y + a_{23} - a_{31}xy' - a_{32}yy'. \end{aligned} \tag{10.33}$$

For N points, this leads to a linear equation system with $2N$ equations and 8 unknowns of the form

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_N \\ y'_N \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ & & & & \vdots & & & \\ x_N & x_N & 1 & 0 & 0 & 0 & -x_Nx'_N & -y_Nx'_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -x_Ny'_N & -y_Ny'_N \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

that can be solved as a least square problem.

10.5 Interpolation

10.5.1 General

The other important aspect of discrete geometric operations besides the transform is *interpolation*. Interpolation is required as the transformed grid points of the input image in general no longer coincide with the grid points of the output image and vice versa.

The basis of interpolation is the sampling theorem (Section 9.2.2). This theorem states that the digital image *completely* represents the

continuous image provided the sampling conditions are met. In short it means that each periodic structure that occurs in the image must be sampled at least twice per wavelength. From this basic fact it is easy — at least in principle — to devise a general framework for interpolation: reconstruct the continuous image first and then perform a new sampling at the new grid points. This procedure only works as long as the new grid has equal or narrower grid spacing. If it is wider, aliasing will occur. In this case, the image must be pre-filtered before it is resampled.

Although these procedures sound simple and straightforward, they are not at all. The problem is related to the fact that the reconstruction of the continuous image from the sampled image in practice is quite involved and can be performed only approximately. Thus, we need to consider how to optimize the interpolation given certain constraints. In this section, we will first see why ideal interpolation is not possible and then discuss various practical approaches in Sections 10.5.2–10.6.2.

In Section 9.3.1, we stated that reconstruction of a continuous function from sampled points can be considered as a convolution operation,

$$\boxed{g_r(\boldsymbol{x}) = \sum_{m,n} g(\boldsymbol{x}_{m,n}) h(\boldsymbol{x} - \boldsymbol{x}_{m,n}),} \tag{10.34}$$

where the continuous interpolation mask h is the sinc function

$$h(\boldsymbol{x}) = \frac{\sin \pi x_1/\Delta x_1}{\pi x_1/\Delta x_1} \frac{\sin \pi x_2/\Delta x_2}{\pi x_2/\Delta x_2}. \tag{10.35}$$

The transfer function of the point spread function in Eq. (10.35) is a box function with widths $2\tilde{k}_w = 1/\Delta x_w$ (Eqs. (9.8) and (9.14)):

$$\hat{h}(\boldsymbol{k}) = \Pi(\tilde{k}_1/2, \tilde{k}_2/2) \quad \text{with} \quad \tilde{k}_w = 2k_w \Delta x_w. \tag{10.36}$$

The interpolatory nature of the convolution kernel Eq. (10.35) can be inferred from the following properties. The interpolated values in Eq. (10.34) at grid points $\boldsymbol{x}_{mn}$ should reproduce the grid points and not depend on any other grid point. From this condition, we can infer the *interpolation condition*:

$$h(\boldsymbol{x}_{m,n}) = \begin{cases} 1 & m = 0, n = 0 \\ 0 & \text{otherwise} \end{cases}. \tag{10.37}$$

The interpolation mask in Eq. (10.35) meets this interpolation condition. Any interpolation mask must, therefore, have zero crossings at all grid points except the zero point where it is 1.

The fact that interpolation is a convolution operation and thus can be described by a transfer function in Fourier space Eq. (10.36) gives us a handy tool to rate the errors associated with an interpolation technique. The box-type transfer function for the ideal interpolation function

simply means that all wave numbers within the range of possible wave numbers $|k_w| \leq 1/(2\Delta x_w)$ experience neither a phase shift nor amplitude damping. Also, no wave numbers beyond the allowed interval are present in the interpolated signal, because the transfer function is zero there.

The ideal interpolation function in Eq. (10.34) is separable. Therefore, interpolation can as easily be formulated for higher-dimensional images. We can expect that all solutions to the interpolation problem will also be separable. Consequently, we need only discuss the 1-D interpolation problem. Once it is solved, we also have a solution for the n-dimensional interpolation problem.

An important special case is the interpolation to intermediate grid points halfway between the existing grid points. This scheme doubles the resolution and image size in all directions in which it is applied. Then, the continuous interpolation kernel reduces to a discrete convolution mask. As the interpolation kernel Eq. (10.35) is separable, we can first interpolate the intermediate points in a row in the horizontal direction before we apply vertical interpolation to the intermediate rows. In three dimensions, a third 1-D interpolation is added in the z or t direction. The interpolation kernels are the same in all directions. We need the continuous kernel $h(x)$ only at half-integer values for $x/\Delta x$. From Eq. (10.35) we obtain the discrete ideal interpolation kernel

$$h = \left[\cdots \frac{(-1)^{m-1} 2}{(2m-1)\pi} \cdots -\frac{2}{3\pi} \frac{2}{\pi} \frac{2}{\pi} -\frac{2}{3\pi} \cdots \frac{(-1)^{m-1} 2}{(2m-1)\pi} \cdots \right] \tag{10.38}$$

with coefficients of alternating sign.

10.5.2 Interpolation in Fourier Space

Interpolation reduces to a simple operation in the Fourier domain. As shown by Eq. (10.36), the transfer function of an ideal interpolation kernel is a rectangular (box) function which is zero outside the wave numbers that can be represented. This basic fact suggests the following interpolation procedure in Fourier space:

1. Enlarge the matrix of the Fourier transformed image. If an $M \times M$ matrix is increased to an $M' \times M'$ matrix, the image in the spatial domain is also increased to an $M' \times M'$ image. Because of the reciprocity of the Fourier transform, the image size remains unchanged. Only the spacing between pixels is decreased, resulting in a higher spatial resolution:

$$M\Delta k \to M'\Delta k \quad \circ\!\!-\!\!\bullet \quad \Delta x = \frac{1}{M\Delta k} \to \Delta x' = \frac{1}{M'\Delta k} \tag{10.39}$$

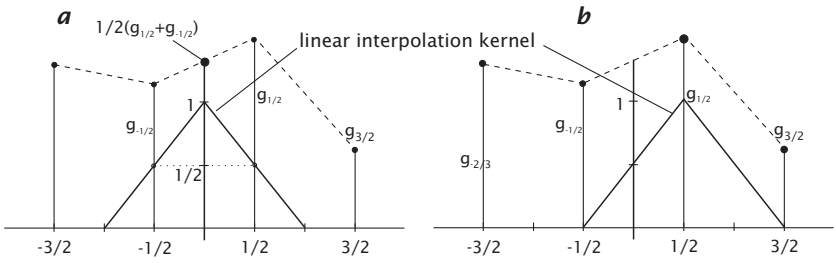

Figure 10.17: *Illustration of linear interpolation:* **a** *at* $\tilde{x} = 0$*, the mean of* $g_{1/2}$ *and* $g_{-1/2}$ *is taken,* **b** *at* $\tilde{x} = 1/2$*,* $g_{1/2}$ *is replicated.*

2. Fill the padded area in the Fourier space with zeroes and compute an inverse Fourier transform.

Theoretically, this procedure results in a perfectly interpolated image. Unfortunately, it has three drawbacks.

1. The Fourier transform of a finite image implies a cyclic repetition of the image in the spatial and Fourier domain. Thus, the convolution performed by the Fourier transform is cyclic. This means that at the edge of the image, convolution continues with the image at the opposite side. As the real world is not periodic and interpolation masks are large, this may lead to significant distortions of the interpolation even at quite large distances from the edges of the image.
2. The Fourier transform can be computed efficiently only for a specified number of values for M'. Best known are the fast radix-2 algorithms that can be applied only to images of the size $M' = 2^{N'}$ (Section 2.5.2). Therefore, the Fourier transform-based interpolation is slow for numbers M' that cannot be expressed as a product of many small factors.
3. As the Fourier transform is a global transform, it can be applied only to scaling. According to the generalized similarity theorem (Theorem 2.1, p. 53), it could also be applied to rotation and affine transforms. But then the interpolation problem is only shifted from the spatial domain to the wave number domain.

10.5.3 Linear Interpolation

Linear interpolation is the classic approach to interpolation. The interpolated points lie on pieces of straight lines connecting neighboring grid points. In order to simplify the expression, we use in the following normalized spatial coordinates $\tilde{x} = x/\Delta x$. We locate the two grid points at $-1/2$ and $1/2$. This yields the interpolation equation

$$g(\tilde{x}) = \frac{g_{1/2} + g_{-1/2}}{2} + (g_{1/2} - g_{-1/2})\,\tilde{x} \quad \text{for} \quad |\tilde{x}| \leq 1/2. \tag{10.40}$$

By comparison of Eq. (10.40) with Eq. (10.34), we can conclude that the continuous interpolation mask for linear interpolation is

$$h_1(\tilde{x}) = \begin{cases} 1 - |\tilde{x}| & |\tilde{x}| \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{10.41}$$

Its interpolatory nature is illustrated in Fig. 10.17. The transfer function of the interpolation mask for linear interpolation, the triangle function $h_1(x)$ in Eq. (10.41), is the squared sinc function (> R5)

$$\hat{h}_1(\tilde{k}) = \frac{\sin^2 \pi\tilde{k}/2}{(\pi\tilde{k}/2)^2}. \tag{10.42}$$

A comparison with the ideal transfer function for interpolation Eq. (10.36) shows that two distortions are introduced by linear interpolation:

1. While low wave numbers (and especially the mean value $\tilde{k} = 0$) are interpolated correctly, high wave numbers are slightly reduced in amplitude, resulting in some degree of smoothing. At $\tilde{k} = 1$, the transfer function is reduced to about 40%: $\hat{h}_1(1) = (2/\pi)^2 \approx 0.4$.
2. As $\hat{h}_1(\tilde{k})$ is not zero at wave numbers $\tilde{k} > 1$, some spurious high wave numbers are introduced. If the continuously interpolated image is resampled, this yields moderate aliasing. The first side lobe has an amplitude of $(2/3\pi)^2 \approx 0.045$.

If we interpolate only the intermediate grid points at $\tilde{x} = 0$, the continuous interpolation function Eq. (10.41) reduces to a discrete convolution mask with values at $\tilde{x} = [\ldots\ -3/2\ -1/2\ 1/2\ 3/2\ \ldots]$. As Eq. (10.41) is zero for $|\tilde{x}| \geq 1$, we obtain the simple interpolation mask $H = 1/2[11]$ with the transfer function

$$\hat{h}_1(\tilde{k}) = \cos \pi\tilde{k}/2. \tag{10.43}$$

The transfer function is real, so no phase shifts occur. The significant amplitude damping at higher wave numbers, however, shows that structures with high wave numbers are not correctly interpolated. Phase shifts do occur at all other values except for the intermediate grid points at $\tilde{x} = 0$. We investigate the phase shift and amplitude attenuation of linear interpolation at arbitrary fractional integer shifts $\epsilon \in [-1/2, 1/2]$. The interpolation mask for a point at ϵ is then $[1/2 - \epsilon, 1/2 + \epsilon]$. The mask contains a symmetric part $[1/2, 1/2]$ and an antisymmetric part $[-\epsilon, \epsilon]$. Therefore, the transfer function is complex and has the form

$$\hat{h}_1(\epsilon, \tilde{k}) = \cos \pi\tilde{k}/2 + 2\mathrm{i}\epsilon \sin \pi\tilde{k}/2. \tag{10.44}$$

In order to estimate the error in the phase shift, it is useful to compensate for the linear phase shift $\Delta\varphi = \epsilon\pi\tilde{k}$ caused by the displacement ϵ.

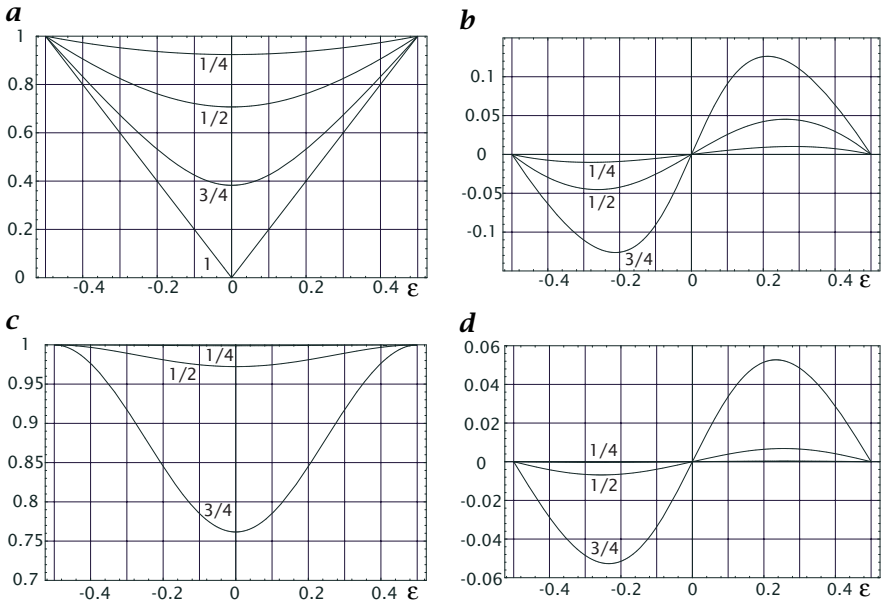

Figure 10.18: *Amplitude attenuation (left column) and phase shift expressed as a position shift $\Delta x = \Delta\varphi\lambda/2\pi$ (right column) in radians for wave numbers $\tilde{k} = 1/4, 1/2, 3/4$, as indicated, displayed as a function of the fractional position from $-1/2$ to $1/2$ for linear interpolation (**a** and **b**) and cubic B-spline interpolation (**c** and **d**).*

According to the shift theorem (Theorem 2.3, p. 54, ≻R4), it is required to multiply Eq. (10.44) by $\exp(-\mathrm{i}\epsilon\pi\tilde{k})$: Then we obtain

$$\hat{h}_1(\epsilon,\tilde{k}) = (\cos\pi\tilde{k}/2 + 2\mathrm{i}\epsilon\sin\pi\tilde{k}/2)\exp(-\mathrm{i}\epsilon\pi\tilde{k}). \tag{10.45}$$

Only for $\epsilon = 0$ and $\epsilon = 1/2$ is the transfer function real: $\hat{h}_1(0,\tilde{k}) = \cos\pi\tilde{k}/2$, $h_1(1/2,\tilde{k}) = 1$; but at all other fractional shifts, a non-zero phase shift remains, as illustrated in Fig. 10.18. The phase shift $\Delta\varphi$ is expressed as the position shift Δx of the corresponding periodic structure, i. e., $\Delta x = \Delta\varphi\lambda/2\pi = \Delta\varphi/(\pi\tilde{k})$.

10.5.4 Polynomial Interpolation

Given the significant limitations of linear interpolation as discussed in Section 10.5.3, we ask whether higher-order interpolation schemes perform better. The basic principle of linear interpolation was that a straight line was drawn to pass through two neighboring points. In the same way, we can use a polynomial of degree P that must pass through $P+1$ points with $P+1$ unknown coefficients a_p:

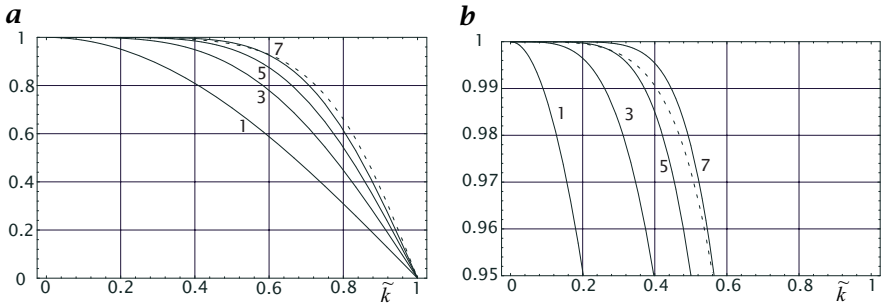

Figure 10.19: *Transfer function of discrete polynomial interpolation filters to interpolate the value between two grid points. The degree of the polynomial (1 = linear, 3 = cubic, etc.) is marked on the graph. The dashed line marks the transfer function for cubic B-spline interpolation (Section 10.6.1).* ***a*** *The full range,* ***b*** *a 5% margin below the ideal response* $\hat{h}(\tilde{k}) = 1$.

$$g_r(\tilde{x}) = \sum_{p=0}^{P} a_p \tilde{x}^p. \tag{10.46}$$

For symmetry reasons, in case of an even number of grid points, we set their position at half-integer values

$$\tilde{x}_p = \frac{2p - P}{2}. \tag{10.47}$$

From the interpolation condition at the grid points $g_r(\tilde{x}_p) = g_p$, we obtain a linear equation system with $P+1$ equations and $P+1$ unknowns a_P of the following form when P is odd:

$$\begin{bmatrix} g_0 \\ \vdots \\ g_{(P-1)/2} \\ g_{(P+1)/2} \\ \vdots \\ g_P \end{bmatrix} = \begin{bmatrix} 1 & -P/2 & P^2/4 & -P^3/8 & \cdots \\ \vdots & & & & \\ 1 & -1/2 & 1/4 & -1/8 & \cdots \\ 1 & 1/2 & 1/4 & 1/8 & \cdots \\ \vdots & & & & \\ 1 & P/2 & P^2/4 & P^3/8 & \cdots \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ \vdots \\ a_P \end{bmatrix} \tag{10.48}$$

from which we can determine the coefficients of the polynomial. For a cubic polynomial ($P = 3$), the equations system is

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & -3/2 & 9/4 & -27/8 \\ 1 & -1/2 & 1/4 & -1/8 \\ 1 & 1/2 & 1/4 & 1/8 \\ 1 & 3/2 & 9/4 & 27/8 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{10.49}$$

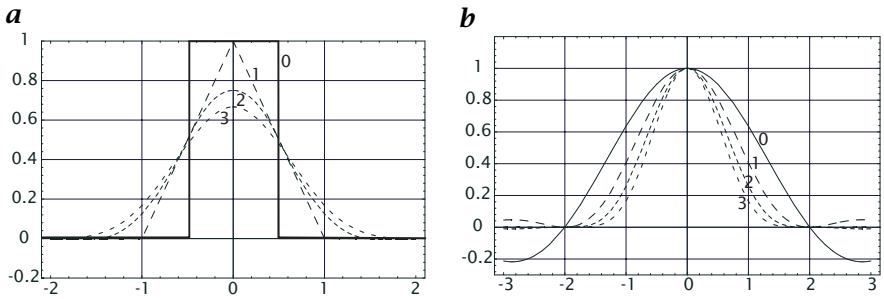

Figure 10.20: ***a*** *B-spline interpolation kernels generated by cascaded convolution of the box kernel of order 0 (nearest neighbor), 1 (linear interpolation), 2 (quadratic B-spline), and 3 (cubic B-spline);* ***b*** *corresponding transfer functions.*

with the solution

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \frac{1}{48} \begin{bmatrix} -3 & 27 & 27 & -3 \\ 2 & -54 & 54 & -2 \\ 12 & -12 & -12 & 12 \\ -8 & 24 & -24 & 8 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}. \tag{10.50}$$

From this solution, we can infer, for example, that the point at $\tilde{x} = 0$ is interpolated by $g_r(0) = a_0 = -1/16g_0 + 9/16g_1 + 9/16g_2 - 1/16g_3$ corresponding to the interpolation mask $1/16[-1, 9, 9, -1]$.

Figure 10.19 shows the transfer functions for a polynomial interpolation of various degrees. With increasing degree P of the interpolating polynomial, the transfer function approaches the box function better. However, convergence is slow. For an accurate interpolation, we must take a large interpolation mask.

10.6 *Optimized Interpolation*

10.6.1 *Spline-Based Interpolation*

Besides its limited accuracy, polynomial interpolation has another significant disadvantage. The interpolated curve is not continuous at the grid points already in its first derivative. This is due to the fact that for each interval between grid points another polynomial is taken. Thus, only the interpolated function is continuous at the grid points but not the derivatives.

Splines avoid this disadvantage by additional constraints for the continuity of derivatives at the grid points. From the wide classes of splines, we will here discuss only one class, the *B-splines*. As B-splines are separable, it is sufficient to discuss the properties of 1-D B-splines. From the background of image processing, the easiest access to B-splines is their convolution property. The kernel of a P-order B-spline curve is generated by convolving the box function $P+1$ times

with itself (Fig. 10.20a):

$$\beta_P(\tilde{x}) = \underbrace{\Pi(\tilde{x}) * \ldots * \Pi(\tilde{x})}_{(P+1)\text{-mal}} \quad \circ\!\!-\!\!\bullet \quad \hat{\beta}_P(\hat{k}) = \left(\frac{\sin \pi\tilde{k}/2}{(\pi\tilde{k}/2)}\right)^{P+1}. \tag{10.51}$$

The transfer function of the box function is the sinc function (≻ R5). Therefore, the transfer function of the P-order B-spline is

$$\hat{\beta}_P(\hat{k}) = \left(\frac{\sin \pi\tilde{k}/2}{(\pi\tilde{k}/2)}\right)^{P+1}. \tag{10.52}$$

Figure 10.20b shows that the B-spline function does not make a suitable interpolation function. The transfer function decreases too early, indicating that B-spline interpolation performs too much averaging. Moreover, the B-spline kernel does not meet the interpolation condition Eq. (10.37) for $P > 1$.

B-splines can only be used for interpolation if first the discrete grid points are transformed in such a way that a following convolution with the B-spline kernel restores the original values at the grid points. This transformation is known as the B-spline transformation and constructed from the following condition:

$$g_p(x) = \sum_n c_n \beta_P(x - x_n) \quad \text{with} \quad g_p(x_n) = g(x_n). \tag{10.53}$$

If centered around a grid point, the B-spline interpolation kernel is unequal to zero only for three grid points. The coefficients $\beta_3(-1) = \beta_{-1}, \beta_3(0) = \beta_0$, and $\beta_3(1) = \beta_1$ are 1/6, 2/3, and 1/6. The convolution of this kernel with the unknown B-spline transform values c_n should result in the original values g_n at the grid points. Therefore,

$$\boldsymbol{g} = \boldsymbol{c} * \boldsymbol{\beta_3} \quad \text{or} \quad g_n = \sum_{n'=-1}^{1} c_{n+n'} \beta_{n'}. \tag{10.54}$$

Equation (10.54) constitutes the sparse linear equation system

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 4 & 1 & 0 & \ddots & 0 & 1 \\ 1 & 4 & 1 & 0 & \ddots & 0 \\ 0 & 1 & 4 & 1 & 0 & \ddots \\ \ddots & & & & \ddots & \ddots \\ \ddots & \ddots & 1 & 4 & 1 & 0 \\ 0 & \ddots & 0 & 1 & 4 & 1 \\ 1 & 0 & \ddots & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix} \tag{10.55}$$

using cyclic boundary conditions. The determination of the B-spline transformation thus requires the solution of a linear equation system with N unknowns.

The special form of the equation system as a convolution operation, however, allows for a more efficient solution. In Fourier space, Eq. (10.54) reduces to

$$\hat{\boldsymbol{g}} = \hat{\boldsymbol{\beta}}_3 \hat{\boldsymbol{c}}. \tag{10.56}$$

The transfer function of $\boldsymbol{\beta}_3$ is $\hat{\beta}_3(\tilde{k}) = 2/3 + 1/3\cos(\pi\tilde{k})$. As this function has no zeroes, we can compute $\boldsymbol{c}$ by inverse filtering (Section 4.4.2), i. e., convoluting $\boldsymbol{g}$ with a mask that has the transfer function

$$\hat{\beta}_3^{-1}(\tilde{k}) = \hat{\beta}_T(\tilde{k}) = \frac{1}{2/3 + 1/3\cos\pi\tilde{k}}. \tag{10.57}$$

Such a transfer function is a kind of recursive filter (Section 4.4.2) that is applied first in the forward and then in the backward direction with the following recursion [204]:

$$\begin{aligned} g'_n &= g_n - (2-\sqrt{3})(g'_{n-1} - g_n) \\ c'_n &= g'_n - (2-\sqrt{3})(c_{n+1} - g'_n). \end{aligned} \tag{10.58}$$

The whole operation takes only two multiplications and four additions.

The B-spline interpolation is applied after the B-spline transformation. In the continuous case, using Eq. (10.51)bsplinerectf, this yields the effective transfer function

$$\hat{\beta}_I(\tilde{k}) = \frac{\sin^4(\pi\tilde{k}/2)/(\pi\tilde{k}/2)^4}{(2/3 + 1/3\cos\pi\tilde{k})}. \tag{10.59}$$

Essentially, the B-spline transformation performs an amplification of high wave numbers (at $\tilde{k} = 1$ by a factor 3). This compensates the smoothing of the B-spline interpolation to a large extent.

We investigate this compensation at the grid points and at the intermediate points. From the equation of the cubic B-spline interpolating kernel Eq. (10.51) (see also Fig. 10.20a) the interpolation coefficients for the grid points and intermediate grid points are

$$\begin{aligned} &1/6\,[1\ 4\ 1] \quad \text{and} \\ &1/48\,[1\ 23\ 23\ 1], \end{aligned} \tag{10.60}$$

respectively. Therefore, the transfer functions are

$$\begin{aligned} &2/3 + 1/3\cos\pi\tilde{k} \quad \text{and} \\ &23/24\cos(\pi\tilde{k}/2) + 1/24\cos(3\pi\tilde{k}/2), \end{aligned} \tag{10.61}$$

respectively. At the grid points, the transfer function exactly compensates — as expected — the application of the B-spline transformation Eq. (10.57). Thus, the interpolation curve goes through the values at the grid points. At the intermediate points the effective transfer function for the cubic B-spline interpolation is then

$$\hat{\beta}_I(1/2, \tilde{k}) = \frac{23/24\cos(\pi\tilde{k}/2) + 1/24\cos(3\pi\tilde{k}/2)}{2/3 + 1/3\cos\pi\tilde{k}}. \tag{10.62}$$

The amplitude attenuation and the phase shifts expressed as a position shift in pixel distances are shown in Fig. 10.18c, d. Note that the shift is related to the intermediate grid. The shift and amplitude damping is zero at the grid points $[-0.5, 0.5]^T$. While the amplitude damping is maximal for the intermediate point, the position shift is also zero at the intermediate point because of

symmetry reasons. Also, at the wave number $\tilde{k} = 3/4$ the phase shift is unfortunately only about two times smaller than for linear interpolation (Fig. 10.18b). It is still significant with a maximum of about 0.13.

This value is much too high for algorithms that ought to be accurate in the 1/100 pixel range. If no better interpolation technique can be applied, this means that the maximum wave number should be lower than 0.5. Then, the maximum shift is lower than 0.01 and the amplitude damping less than 3%.

Note that these comments on phase shifts only apply for arbitrary fractional shifts. For pixels on the intermediate grid, no position shift occurs at all. In this special case — which often occurs in image processing, for instance for pyramid computations (Chapter 5) — optimization of interpolation filters is quite easy because only the amplitude damping must be minimized over the wave number range of interest.

10.6.2 *Least-Squares Approach to Interpolation*

Filter design for interpolation — like any filter design problem — can be treated in a mathematically more rigorous way as an optimization problem. The general idea is to vary the filter coefficients in such a way that the derivation from the ideal transfer function reaches a minimum. For non-recursive filters, the transfer function is linear in the cofficients h_r:

$$\hat{h}(\tilde{k}) = \sum_{r=1}^{R} h_r \hat{f}_r(\tilde{k}). \tag{10.63}$$

Let the ideal transfer function be $\hat{h}_I(\tilde{k})$. Then the optimization procedure should minimize the integral

$$\int_0^1 w(\tilde{k}) \left| \left(\sum_{r=1}^{R} h_r \hat{f}_r(\tilde{k}) \right) - \hat{h}_I(\tilde{k}) \right|^n \mathrm{d}\tilde{k}. \tag{10.64}$$

In this expression, a weighting function $w(\tilde{k})$ has been introduced which allows control over the optimization for a certain wave number range. In equation Eq. (10.64) an arbitrary L_n-norm is included. Mostly the L_2-norm is taken, which minimizes the sum of squares.

For the L_2-norm, the minimization problem results in a linear equation system for the R coefficients of the filter which can readily be solved:

$$\boldsymbol{M}\boldsymbol{h} = \boldsymbol{d} \tag{10.65}$$

with

$$\boldsymbol{d} = \begin{bmatrix} \overline{h_I \hat{f}_1} \\ \overline{h_I \hat{f}_2} \\ \vdots \\ \overline{h_I \hat{f}_R} \end{bmatrix} \quad \text{and} \quad \boldsymbol{M} = \begin{bmatrix} \overline{\hat{f}_1^2} & \overline{\hat{f}_1 \hat{f}_2} & \cdots & \overline{\hat{f}_1 \hat{f}_R} \\ \overline{\hat{f}_1 \hat{f}_2} & \overline{\hat{f}_2^2} & \cdots & \overline{\hat{f}_2 \hat{f}_R} \\ \vdots & & \ddots & \vdots \\ \overline{\hat{f}_1 \hat{f}_R} & \overline{\hat{f}_2 \hat{f}_R} & \cdots & \overline{\hat{f}_R^2} \end{bmatrix}$$

where the abbreviation

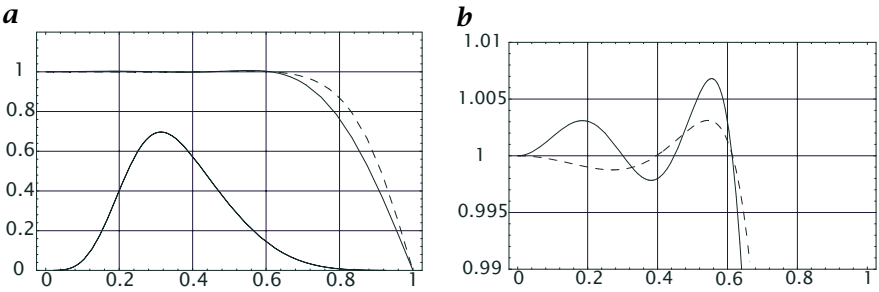

Figure 10.21: *Transfer function of interpolation kernels optimized with the weighted least squares technique of Eq. (10.67) and Eq. (10.68) with R = 3 (solid line) and of Eq. (10.69) for R = 2 (dashed line). The weighting function used for the optimization is shown in* ***a*** *as a thin solid line;* ***b*** *shows a narrow sector of the plot in* ***a*** *for a better estimation of small deviations from ideal values.*

$$\overline{\hat{e}(\tilde{k})} = \int_0^1 w(\tilde{k}) \cdot \hat{e}(\tilde{k}) \mathrm{d}\tilde{k} \tag{10.66}$$

for an arbitrary function $e(\tilde{k})$ has been used.

The flexibility of the least squares optimization technique for filter design is given by the free choice of the weighting function $w(\tilde{k})$ and the careful consideration of the symmetry properties and other features of a filter by the choice of the transfer function in Eq. (10.63). For illustration, we discuss the following two approaches:

$$\hat{h}(\tilde{k}) = \sum_{r=1}^{R} h_r \cos\left(\frac{2r-1}{2}\pi\tilde{k}\right) \tag{10.67}$$

and

$$\hat{h}(\tilde{k}) = \cos\left(\frac{1}{2}\pi\tilde{k}\right) + \sum_{r=2}^{R} h_r \left[\cos\left(\frac{2r-3}{2}\pi\tilde{k}\right) - \cos\left(\frac{1}{2}\pi\tilde{k}\right)\right]. \tag{10.68}$$

Both filters result in a symmetric mask by the choice of the cosine function. Equation Eq. (10.68) ensures that $\hat{h}(0) = 1$, i. e., the mean gray values are preserved by the interpolation. This is done by forcing the first coefficient, h_1, to be one minus the sum of all others. Equation Eq. (10.67) does not apply this constraint. Figure 10.21 compares the optimal transfer functions with both approaches for $R = 3$.

The filters are significantly better than those obtained by polynomial and cubic B-spline interpolation (Fig. 10.19). The additional degree of freedom for Eq. (10.67) leads to significantly better solutions for the wave number range where the weighting function is maximal.

Even better interpolation masks can be obtained by using a combination of nonrecursive and recursive filters, as with the cubic B-spline interpolation:

$$\hat{h}(\tilde{k}) = \frac{\cos\left(1/2\ \pi\tilde{k}\right) + \sum_{r=2}^{R} h_r \left[\cos\left((2r-3)/2\ \pi\tilde{k}\right) - \cos\left(1/2\ \pi\tilde{k}\right)\right]}{1 - \alpha + \alpha\cos\left(\pi\tilde{k}\right)}. \tag{10.69}$$

With recursive filters, the least squares optimization becomes nonlinear because $\hat{h}(\tilde{k})$ in Eq. (10.69) is nonlinear in the parameter α of the recursive filter. Then, iterative techniques are required to solve the optimization problem. Figure 10.21c, d shows the transfer functions for $R = 2$. A more detailed discussion of interpolation filters including tables with optimized filters can be found in Jähne [89].

10.6.3 *Fast Algorithms for Geometric Transforms*

With the extensive discussion on interpolation we are well equipped to devise fast algorithms for the different geometric transforms. Basically, all fast interpolation algorithms use the following two principles: efficient computation employing the interpolation coefficients and partition into 1-D geometric transforms.

First, many computations are required to compute the interpolation coefficients for fractional shifts. For each shift, different interpolation coefficients are required. Thus we must devise the transforms in such a way that we need only constant shifts for a certain pass of the transform. If this is not possible, it might still be efficient to precompute the interpolation coefficients for different fractional shifts and to save them for later usage.

Second, we learnt in Section 10.5.1 that interpolation is a separable procedure. Taking advantage of this basic fact considerably reduces the number of operations. In most cases it is possible to separate the two- and higher dimensional geometric transforms into a series of 1-D transforms.

10.7 *Multichannel Point Operations*

10.7.1 *Definitions*

Point operations can be generalized to multichannel point operations in a straightforward way. The operation still depends only on the values of a single pixel. The only difference is that it depends on a vectorial input instead of a scalar input. Likewise, the output image can be a multichannel image. For homogeneous point operations that do not depend on the position of the pixel in the image, we can write

$$\boldsymbol{G}' = \boldsymbol{P}(\boldsymbol{G}) \quad \text{with} \quad \begin{aligned} \boldsymbol{G}' &= [\boldsymbol{G}'_0,\ \boldsymbol{G}'_1,\ \ldots,\ \boldsymbol{G}'_l,\ \ldots,\ \boldsymbol{G}'_{L-1}], \\ \boldsymbol{G} &= [\boldsymbol{G}_0,\ \boldsymbol{G}_1,\ \ldots,\ \boldsymbol{G}_k,\ \ldots,\ \boldsymbol{G}_{K-1}], \end{aligned} \tag{10.70}$$

where $\boldsymbol{G}'_l$ and $\boldsymbol{G}_k$ are the components l and k of the multichannel images $\boldsymbol{G}'$ and $\boldsymbol{G}$ with L and K channels, respectively.

Linear operators are an important subclass of multicomponent point operators. This means that each component of the multichannel image $\boldsymbol{G}'$ is a linear combination of the components of the multichannel image $\boldsymbol{G}$:

$$\boldsymbol{G}'_l = \sum_{k=0}^{K-1} P_{lk} \boldsymbol{G}_k \tag{10.71}$$

where P_{lk} are constant coefficients. Therefore, a general linear multicomponent point operation is given by a matrix of coefficients $\boldsymbol{P}$. Then, we can write Eq. (10.71) in matrix notation as

$$\boldsymbol{G}' = \boldsymbol{P}\boldsymbol{G}. \tag{10.72}$$

If the components of the multichannel images in a point operation are not interrelated to each other, all coefficients in $\boldsymbol{P}$ except those on the diagonal become zero. For K-channel input and output images, just K different point operations remain, one for each channel. The matrix of point operations finally reduces to a standard scalar point operation when the same point operation is applied to each channel of a multi-component image.

For an equal number of output and input images, linear point operations can be interpreted as coordinate transformation. If the matrix of the coefficients in Eq. (10.72) has a rank $R < K$, the multichannel point operation projects the K-dimensional space to an R-dimensional subspace.

Generally, linear multichannel point operations are quite easy to handle as they can be described in a straightforward way with the concepts of linear algebra. For square matrices, for instance, we can easily give the condition when an inverse operation to a multichannel operation exists and compute it. For nonlinear multicomponent point operations, the linear coefficients in Eqs. (10.71) and (10.72) have to be replaced by nonlinear functions:

$$\boldsymbol{G}'_l = P_l(\boldsymbol{G}_0, \boldsymbol{G}_1, \ldots, \boldsymbol{G}_{K-1}) \tag{10.73}$$

Nonlinear multicomponent point operations cannot be handled in a general way, unlike linear operations. Thus, they must be considered individually. The complexity can be reduced significantly if it is possible to separate a given multichannel point operation into its linear and nonlinear parts.

10.7.2 *Dyadic Point Operations*

Operations in which only two images are involved are termed *dyadic point operations*. Dyadic homogeneous point operations can be implemented as LUT operations. Generally, any dyadic image operation can be expressed as

$$G'_{mn} = P(G_{mn}, H_{mn}). \tag{10.74}$$

If the gray values of the two input images take Q different values, there are Q^2 combinations of input parameters and, thus, different output values. Thus, for 8-bit images, 64K values need to be calculated. This is still a quarter less than with a direct computation for each pixel in a 512×512 image. All possible results of the dyadic operation can be stored in a large LUT L with $Q^2 = 64\text{K}$ entries in the following manner:

$$L(2^8 p + q) = P(p, q), \quad 0 \le p, q < Q. \tag{10.75}$$

The high and low bytes of the LUT address are given by the gray values in the images G and H, respectively.

Some image processing systems contain a 16-bit LUT as a modular processing element. Computation of a dyadic point operation either with a hardware or

software LUT is often significantly faster than a direct implementation, especially if the operation is complex. In addition, it is easier to control exceptions such as division by zero or underflow and overflow.

A dyadic point operation can be used to perform two point operations simultaneously. The phase and magnitude (r, i) of a complex-valued image, for example, can be computed simultaneously with one dyadic LUT operation if we restrict the output to 8 bits as well:

$$L(2^8 r + i) = 2^8 \sqrt{r^2 + i^2} + \frac{128}{\pi} \arctan\left(\frac{i}{r}\right), \quad 0 \le r, i < Q. \tag{10.76}$$

The magnitude is returned in the high byte and the phase, scaled to the interval $[-128, 127]$, in the low byte.

10.8 *Exercises*

Problem 10.1: Contrast enhancement

Interactive demonstration of contrast enhancement by lookup tables (dip6ex10.01)

Problem 10.2: Inspection of inhomogeneous illumination

Interactive illustration of the possibilities to objectively inspect inhomogeneous illumination using homogeneous point operations (dip6ex10.02)

Problem 10.3: Overflow detection

Interactive demonstration of the detection of underflow and overflow using histograms (dip6ex10.03)

Problem 10.4: Homogeneous point operations

Interactive demonstration of homogeneous point operations (dip6ex10.04)

Problem 10.5: * Lookup tables

Lookup tables can be used for fast computation of homogeneous point operations. Determine the equations for the computation of lookup tables for the following point operations. The images have $Q = 2^P$ discrete values. Also answer the question whether the point operation can be inverted.

1. Negative image (white becomes black and vice versa)
2. A lookup table that indicates the underflow and overflow of gray values. Underflow should be marked in blue, overflow in red. Hint: color output requires three lookup tables, one each for red, green, and blue (additive color mixing).
3. Contrast enhancement: a small range of S gray values should be mapped to the full gray value range of 2^P gray values.

Problem 10.6: *Correction of nonlinear calibration curves

With lookup tables nonlinear calibration curves can be corrected.

1. Write down the complete lookup table for the following calibration curve:

$$g' = a_0 + a_1 g + a_2 g^2,$$

   where $a_0 = 0$, $a_1 = 0.7$, and $a_2 = 0.02$ with 16 different gray values (4 bit, gray values 0 to 15). Please note that there are different possibilities for rounding: a) truncation (next lower integer) and b) round-to-nearest (integer that is closest to the floating point number).
2. What types of errors are caused by rounding?
3. How are the rounding errors reduced if the sensor digitizes the signal g internally with 6 bits (gray values 0 to 63) and outputs g' as 4-bit values? Write down a modified lookup table that covers this case.

Problem 10.7: **Computation of polar coordinates with a lookup table

Dyadic functions (functions with two input values) can efficiently be computed with lookup tables.

1. Determine the equations to compute a lookup table that computes the polar coordinates form Cartesian coordinates with P bits resolution:

$$r = (x^2 + y^2)^{1/2}, \quad \phi = (2^{P-1}/\pi) \arctan(y/x)$$

2. How many elements has the lookup table?

Problem 10.8: Averaging of noisy image sequences

Interactive demonstration of the averaging of noisy image sequences; computation of the variance image (dip6ex10.05)

Problem 10.9: Correction of inhomogeneous illumination

Interactive demonstration of the correction of inhomogeneous illumination using inhomogeous point operations (dip6ex10.06)

Problem 10.10: Window functions with Fourier transform

Interactive demonstration of the use of window functions with the Fourier transform (dip6ex10.07)

Problem 10.11: Interpolation

Interactive demonstration of the accuracy of different interpolation methods with subpixel-accurate scaling, shifting, and rotation of images (dip6ex10.08)

Problem 10.12: *Linear and cubic interpolation

A cosine signal is sampled either four or eight times per wavelength. Which signal form is generated when a continuous signal is reconstructed from these sampled signals by either linear or cubic interpolation?

10.9 *Further Readings*

Holst [78, 80] and Biberman [8] deal with radiometric calibration of sensors and cameras in the visible and infrared. A detailed discussion of interpolation filters including tables with filter coefficients and efficient algorithms for geometric transforms can be found in Jähne [89, Chap. 8]. Readers interested in the mathematical background of interpolation are referred to Davis [29] and Lancaster and Salkauskas [115]. Wolberg [219] expounds geometric transforms.

Part III

Feature Extraction