2 Image Representation

2.1 Introduction

This chapter centers around the question of how to represent the information contained in images. Together with the next two chapters it lays the mathematical foundations for low-level image processing. Two key points are emphasized in this chapter.

First, the information contained in images can be represented in entirely different ways. The most important are the spatial representation (Section 2.2) and wave number representation (Section 2.3). These representations just look at spatial data from different points of view. Since the various representations are complete and equivalent, they can be converted into each other. The conversion between the spatial and wave number representation is the well-known *Fourier transform*. This transform is an example of a more general class of operations, the *unitary transforms* (Section 2.4).

Second, we discuss how these representations can be handled with digital computers. How are images represented by arrays of digital numbers in an adequate way? How are these data handled efficiently? Can fast algorithms be devised to convert one representation into another? A key example is the fast Fourier transform, discussed in Section 2.5.

2.2 Spatial Representation of Digital Images

2.2.1 Pixel and Voxel

Images constitute a spatial distribution of the *irradiance* at a plane. Mathematically speaking, the spatial irradiance distribution can be described as a continuous function of two spatial variables:

$$E(x_1, x_2) = E(\boldsymbol{x}). \tag{2.1}$$

Computers cannot handle continuous images but only arrays of digital numbers. Thus it is required to represent images as two-dimensional arrays of points. A point on the 2-D grid is called a *pixel* or *pel*. Both words are abbreviations of the word picture element. A pixel represents the irradiance at the corresponding grid position. In the simplest case, the pixels are located on a rectangular grid. The position of the pixel

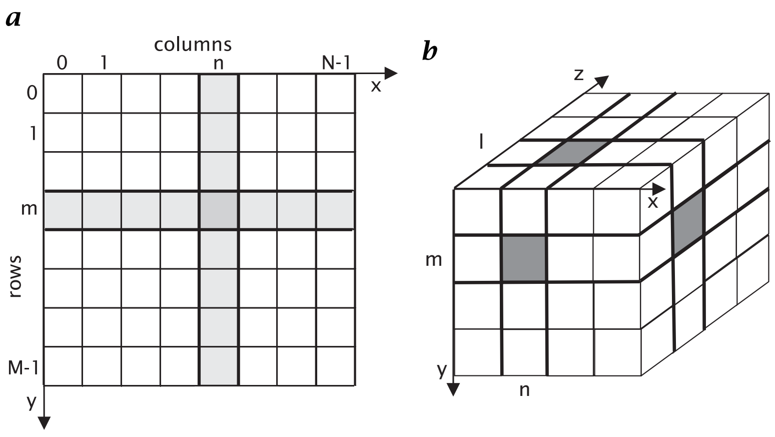

Figure 2.1: *Representation of digital images by arrays of discrete points on a rectangular grid:* ***a*** *2-D image,* ***b*** *3-D image.*

is given in the common notation for matrices. The first index, m, denotes the position of the row, the second, n, the position of the column (Fig. 2.1a). If the digital image contains $M \times N$ pixels, i. e., is represented by an $M \times N$ matrix, the index n runs from 0 to $N - 1$, and the index m from 0 to $M - 1$. M gives the number of rows, N the number of columns. In accordance with the matrix notation, the vertical axis (y axis) runs from top to bottom and not vice versa as it is common in graphs. The horizontal axis (x axis) runs as usual from left to right.

Each pixel represents not just a point in the image but rather a rectangular region, the elementary cell of the grid. The value associated with the pixel must represent the average irradiance in the corresponding cell in an appropriate way. Figure 2.2 shows one and the same image represented with a different number of pixels as indicated in the legend. With large pixel sizes (Fig. 2.2a, b), not only is the spatial resolution poor, but the gray value discontinuities at pixel edges appear as disturbing artifacts distracting us from the content of the image. As the pixels become smaller, the effect becomes less pronounced up to the point where we get the impression of a spatially continuous image. This happens when the pixels become smaller than the spatial resolution of our visual system. You can convince yourself of this relation by observing Fig. 2.2 from different distances.

How many pixels are sufficient? There is no general answer to this question. For visual observation of a digital image, the pixel size should be smaller than the spatial resolution of the visual system from a nominal observer distance. For a given task the pixel size should be smaller than the finest scales of the objects that we want to study. We generally find, however, that it is the available sensor technology (see Section 1.7.1)

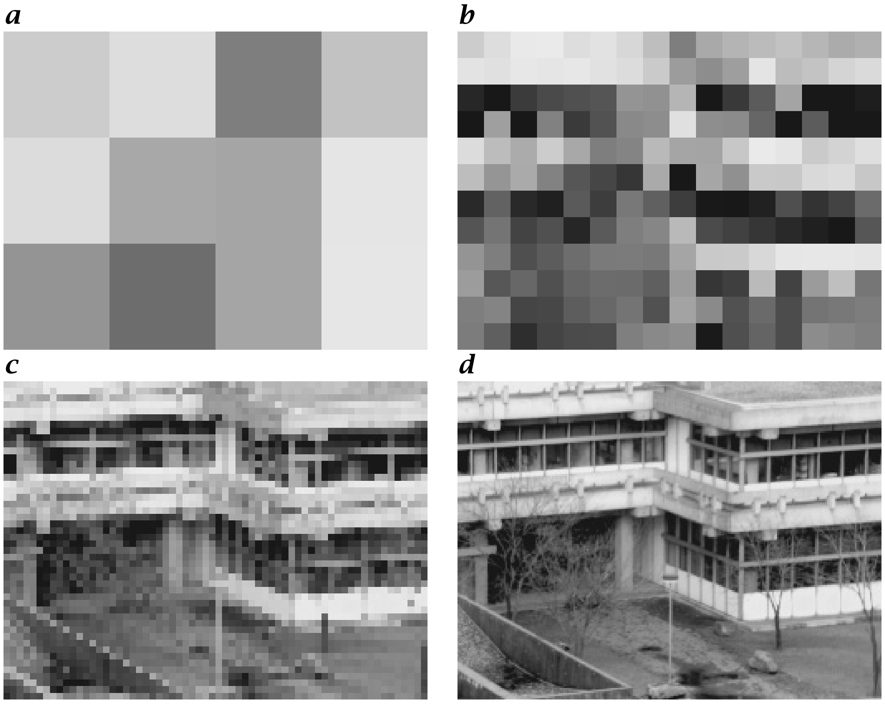

Figure 2.2: *Digital images consist of pixels. On a square grid, each pixel represents a square region of the image. The figure shows the same image with* ***a*** 3×4*,* ***b*** 12×16*,* ***c*** 48×64*, and* ***d*** 192×256 *pixels. If the image contains sufficient pixels, it appears to be continuous.*

that limits the number of pixels rather than the demands from the applications. Even a high-resolution sensor array with 1000×1000 elements has a relative spatial resolution of only 10^{-3}. This is a rather poor resolution compared to other measurements such as those of length, electrical voltage or frequency, which can be performed with relative resolutions of far beyond 10^{-6}. However, these techniques provide only a measurement at a single point, while a 1000×1000 image contains *one million* points. Thus we obtain an insight into the spatial variations of a signal. If we take image sequences, also the temporal changes and, thus, the kinematics and dynamics of the studied object become apparent. In this way, images open up a whole new world of information.

A rectangular grid is only the simplest geometry for a digital image. Other geometrical arrangements of the pixels and geometric forms of the elementary cells are possible. Finding the possible configurations is the 2-D analogue of the classification of crystal structure in 3-D space, a subject familiar to solid state physicists, mineralogists, and chemists. Crystals show periodic 3-D patterns of the arrangements of their atoms,

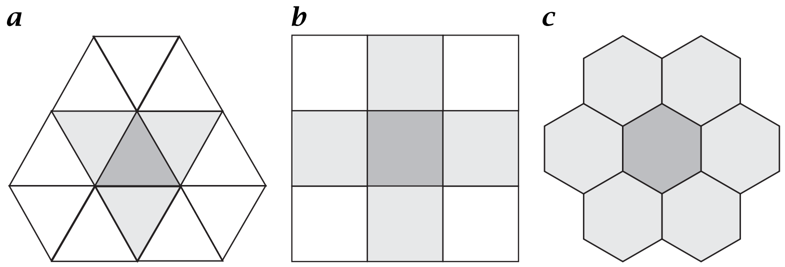

Figure 2.3: *The three possible regular grids in 2-D:* ***a*** *triangular grid,* ***b*** *square grid,* ***c*** *hexagonal grid.*

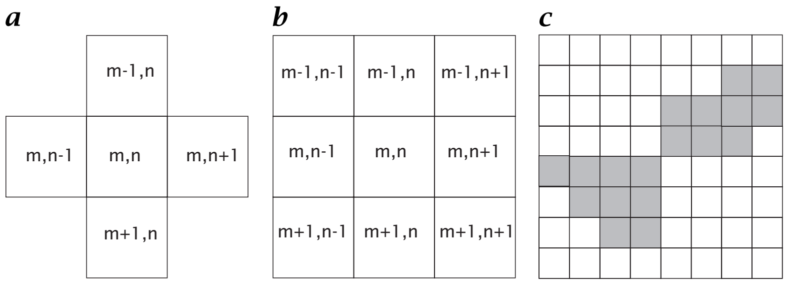

Figure 2.4: *Neighborhoods on a rectangular grid:* ***a*** *4-neighborhood and* ***b*** *8-neighborhood.* ***c*** *The black region counts as one object (connected region) in an 8-neighborhood but as two objects in a 4-neighborhood.*

ions, or molecules which can be classified by their symmetries and the geometry of the elementary cell. In 2-D, classification of digital grids is much simpler than in 3-D. If we consider only regular polygons, we have only three possibilities: triangles, squares, and hexagons (Fig. 2.3).

The 3-D spaces (and even higher-dimensional spaces) are also of interest in image processing. In three-dimensional images a pixel turns into a *voxel*, an abbreviation of *volume element*. On a rectangular grid, each voxel represents the mean gray value of a cuboid. The position of a voxel is given by three indices. The first, k, denotes the depth, m the row, and n the column (Fig. 2.1b). A Cartesian grid, i. e., hypercubic pixel, is the most general solution for digital data since it is the only geometry that can easily be extended to arbitrary dimensions.

2.2.2 Neighborhood Relations

An important property of discrete images is their *neighborhood relations* since they define what we will regard as a *connected region* and therefore as a *digital object*. A *rectangular grid* in two dimensions shows the unfortunate fact, that there are two possible ways to define neighboring pixels (Fig. 2.4a, b). We can regard pixels as neighbors either when they

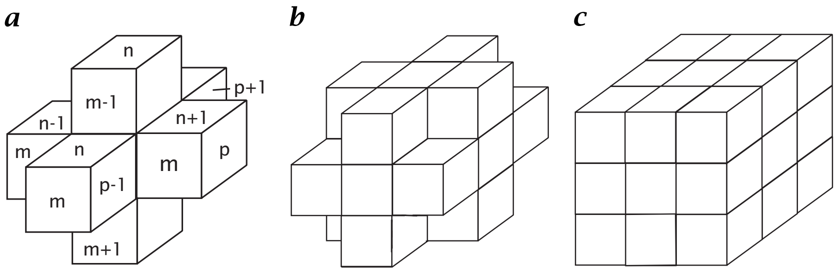

Figure 2.5: *The three types of neighborhoods on a 3-D cubic grid.* ***a*** *6-neighborhood: voxels with joint faces;* ***b*** *18-neighborhood: voxels with joint edges;* ***c*** *26-neighborhood: voxels with joint corners.*

have a joint edge or when they have at least one joint corner. Thus a pixel has four or eight neighbors and we speak of a *4-neighborhood* or an *8-neighborhood*.

Both types of neighborhood are needed for a proper definition of objects as connected regions. A region or an object is called connected when we can reach any pixel in the region by walking from one neighboring pixel to the next. The black object shown in Fig. 2.4c is one object in the 8-neighborhood, but constitutes two objects in the 4-neighborhood. The white background, however, shows the same property. Thus we have either two connected regions in the 8-neigborhood crossing each other or two separated regions in the 4-neighborhood. This inconsistency can be overcome if we declare the objects as 4-neighboring and the background as 8-neighboring, or vice versa.

These complications occur not only with a *rectangular grid*. With a *triangular grid* we can define a 3-neighborhood and a 12-neighborhood where the neighbors have either a common edge or a common corner, respectively (Fig. 2.3a). On a hexagonal grid, however, we can only define a *6-neighborhood* because pixels which have a joint corner, but no joint edge, do not exist. Neighboring pixels always have one joint edge and two joint corners. Despite this advantage, hexagonal grids are hardly used in image processing, as the imaging sensors generate pixels on a rectangular grid. The photosensors on the retina in the human eye, however, have a more hexagonal shape [210].

In three dimensions, the neighborhood relations are more complex. Now, there are three ways to define a neighbor: voxels with joint faces, joint edges, and joint corners. These definitions result in a 6-neighborhood, an 18-neighborhood, and a 26-neighborhood, respectively (Fig. 2.5). Again, we are forced to define two different neighborhoods for objects and the background in order to achieve a consistent definition of connected regions. The objects and background must be a 6-neighborhood and a 26-neighborhood, respectively, or vice versa.

2.2.3 Discrete Geometry

The discrete nature of digital images makes it necessary to redefine elementary geometrical properties such as distance, slope of a line, and coordinate transforms such as translation, rotation, and scaling. These quantities are required for the definition and measurement of geometric parameters of object in digital images.

In order to discuss the discrete geometry properly, we introduce the *grid vector* that represents the position of the pixel. The following discussion is restricted to rectangular grids. The grid vector is defined in 2-D, 3-D, and 4-D spatiotemporal images as

$$\boxed{\boldsymbol{r}_{m,n} = \begin{bmatrix} n\Delta x \\ m\Delta y \end{bmatrix}, \; \boldsymbol{r}_{l,m,n} = \begin{bmatrix} n\Delta x \\ m\Delta y \\ l\Delta z \end{bmatrix}, \; \boldsymbol{r}_{k,l,m,n} = \begin{bmatrix} n\Delta x \\ m\Delta y \\ l\Delta z \\ k\Delta t \end{bmatrix}.} \quad (2.2)$$

To measure distances, it is still possible to transfer the *Euclidian distance* from continuous space to a discrete grid with the definition

$$d_e(\boldsymbol{r},\boldsymbol{r}') = \|\boldsymbol{r} - \boldsymbol{r}'\| = \left[(n-n')^2\Delta x^2 + (m-m')^2\Delta y^2\right]^{1/2}. \quad (2.3)$$

Equivalent definitions can be given for higher dimensions. In digital images two other metrics have often been used. The *city block distance*

$$d_b(\boldsymbol{r},\boldsymbol{r}') = |n-n'| + |m-m'| \quad (2.4)$$

gives the length of a path, if we can only walk in horizontal and vertical directions (4-neighborhood). In contrast, the *chess board distance* is defined as the maximum of the horizontal and vertical distance

$$d_c(\boldsymbol{r},\boldsymbol{r}') = \max(|n-n'|, |m-m'|). \quad (2.5)$$

For practical applications, only the Euclidian distance is relevant. It is the only metric on digital images that preserves the isotropy of the continuous space. With the city block distance, for example, distances in the direction of the diagonals are longer than the Euclidean distance. The curve with equal distances to a point is not a circle but a diamond-shape curve, a square tilted by 45°.

Translation on a discrete grid is only defined in multiples of the pixel or voxel distances

$$\boldsymbol{r}'_{m,n} = \boldsymbol{r}_{m,n} + \boldsymbol{t}_{m',n'}, \quad (2.6)$$

i. e., by addition of a grid vector $\boldsymbol{t}_{m',n'}$.

Likewise, *scaling* is possible only for integer multiples of the scaling factor by taking every qth pixel on every pth line. Since this discrete

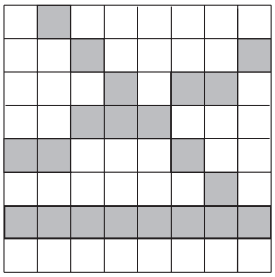

Figure 2.6: *A discrete line is only well defined in the directions of axes and diagonals. In all other directions, a line appears as a staircase-like jagged pixel sequence.*

scaling operation subsamples the grid, it remains to be seen whether the scaled version of the image is still a valid representation.

Rotation on a discrete grid is not possible except for some trivial angles. The condition is that all points of the rotated grid coincide with the grid points. On a rectangular grid, only rotations by multiples of 180° are possible, on a square grid by multiples of 90°, and on a hexagonal grid by multiples of 60°.

Generally, the correct representation even of simple geometric objects such as lines and circles is not clear. Lines are well-defined only for angles with values of multiples of 45°, whereas for all other directions they appear as jagged, staircase-like sequences of pixels (Fig. 2.6).

All these limitations of digital geometry cause errors in the position, size, and orientation of objects. It is necessary to investigate the consequences of these errors for subsequent processing carefully (Chapter 9).

2.2.4 Quantization

For use with a computer, the measured irradiance at the image plane must be mapped onto a limited number Q of discrete gray values. This process is called *quantization*. The number of required quantization levels in image processing can be discussed with respect to two criteria.

First, we may argue that no gray value steps should be recognized by our visual system, just as we do not see the individual pixels in digital images. Figure 2.7 shows images quantized with 2 to 16 levels of gray values. It can be seen clearly that a low number of gray values leads to false edges and makes it very difficult to recognize objects that show slow spatial variation in gray values. In printed images, 16 levels of gray values seem to be sufficient, but on a monitor we would still be able to see the gray value steps.

Generally, image data are quantized into 256 gray values. Then each pixel occupies 8 bits or one byte. This bit size is well adapted to the

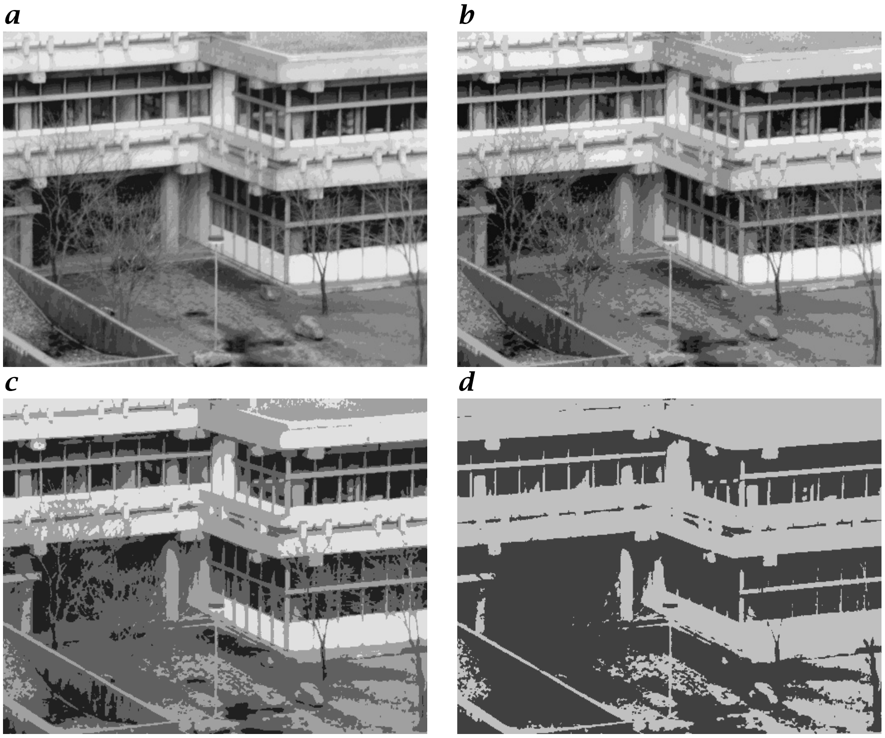

Figure 2.7: *Illustration of quantization. The same image is shown with different quantization levels:* ***a*** *16,* ***b*** *8,* ***c*** *4,* ***d*** *2. Too few quantization levels produce false edges and make features with low contrast partly or totally disappear.*

architecture of standard computers that can address memory bytewise. Furthermore, the resolution is good enough to give us the illusion of a continuous change in the gray values, since the relative intensity resolution of our visual system is no better than about 2 %.

The other criterion is related to the imaging task. For a simple application in machine vision, where homogeneously illuminated objects must be detected and measured, only two quantization levels, i. e., a *binary image*, may be sufficient. Other applications such as imaging spectroscopy or medical diagnosis with x-ray images require the resolution of faint changes in intensity. Then the standard 8-bit resolution would be too coarse.

2.2.5 *Signed Representation of Images*

Normally we think of “brightness” (irradiance or radiance) as a positive quantity. Consequently, it appears natural to represent it by unsigned numbers ranging in an 8-bit representation, for example, from 0 to 255. This representation

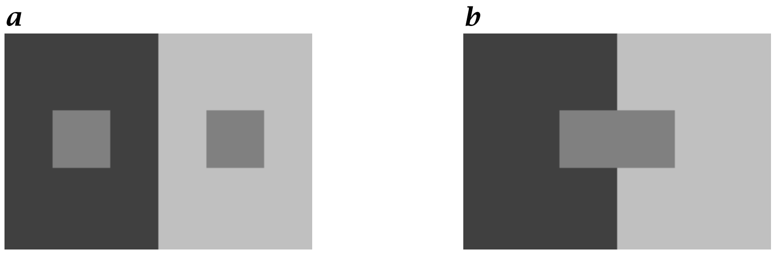

Figure 2.8: *The context determines how "bright" we perceive an object to be. Both squares have the same brightness, but the square on the dark background appears brighter than the square on the light background. The two squares only appear equally bright if they touch each other.*

causes problems, however, as soon as we perform arithmetic operations with images. Subtracting two images is a simple example that can produce negative numbers. Since negative gray values cannot be represented, they wrap around and appear as large positive values. The number -1, for example, results in the positive value 255 given that -1 modulo 256 = 255. Thus we are confronted with the problem of two different representations of gray values, as unsigned and signed 8-bit numbers. Correspondingly, we must have several versions of each algorithm, one for unsigned gray values, one for signed values, and others for mixed cases.

One solution to this problem is to handle gray values *always* as signed numbers. In an 8-bit representation, we can convert unsigned numbers into signed numbers by subtracting 128:

$$q' = (q - 128) \mod 256, \quad 0 \le q < 256. \tag{2.7}$$

Then the mean gray value intensity of 128 becomes the gray value zero and gray values lower than this mean value become negative. Essentially, we regard gray values in this representation as a deviation from a mean value.

This operation converts unsigned gray values to signed gray values which can be stored and processed as such. Only for display must we convert the gray values again to unsigned values by the inverse point operation

$$q = (q' + 128) \mod 256, \quad -128 \le q' < 128, \tag{2.8}$$

which is the same operation as in Eq. (2.7) since all calculations are performed modulo 256.

2.2.6 Luminance Perception of the Human Visual System

With respect to quantization, it is important to know how the human visual system perceives the levels and what luminance differences can be distinguished. Figure 2.8 demonstrates that the small rectangle with a medium luminance appears brighter against the dark background than against the light one, though its absolute luminance is the same. This deception only disappears when the two areas become adjacent.

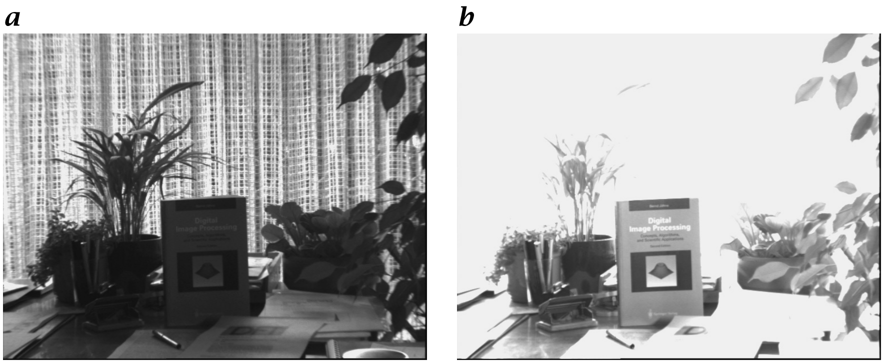

Figure 2.9: *A high-contrast scene captured by a CCD camera with a linear contrast and* ***a*** *a small and* ***b*** *a large aperture.*

The human visual system shows rather a logarithmic than a linear response. This means that we perceive relative and not absolute luminance differences equally well. In a wide range of luminance values, we can resolve relative differences of about 2%. This threshold value depends on a number of factors, especially the spatial frequency (wavelength) of the pattern used for the experiment. At a certain wavelength the luminance resolution is optimal.

The characteristics of the human visual system discussed above are quite different from those of a machine vision system. Typically only 256 gray values are resolved. Thus a digitized image has much lower dynamics than the human visual system. This is the reason why the quality of a digitized image, especially of a scene with high luminance contrast, appears inferior to us compared to what we see directly. In a digital image taken from such a scene with a linear image sensor, either the bright parts are overexposed or the dark parts are underexposed. This is demonstrated by the high-contrast scene in Fig. 2.9.

Although the *relative* resolution is far better than 2% in the bright parts of the image, it is poor in the dark parts. At a gray value of 10, the luminance resolution is only 10%.

One solution for coping with large dynamics in scenes is used in video cameras, which generally convert the irradiance E not linearly, but with an exponential law into the gray value g:

$$g = E^{\gamma}. \tag{2.9}$$

The exponent γ is denoted the *gamma value*. Typically, γ has a value of 0.4. With this exponential conversion, the logarithmic characteristic of the human visual system may be approximated. The contrast range is significantly enhanced. If we presume a minimum relative luminance resolution of 10% and an 8 bit gray scale range, we get contrast ranges of 25 and 316 with $\gamma = 1$ and $\gamma = 0.4$, respectively.

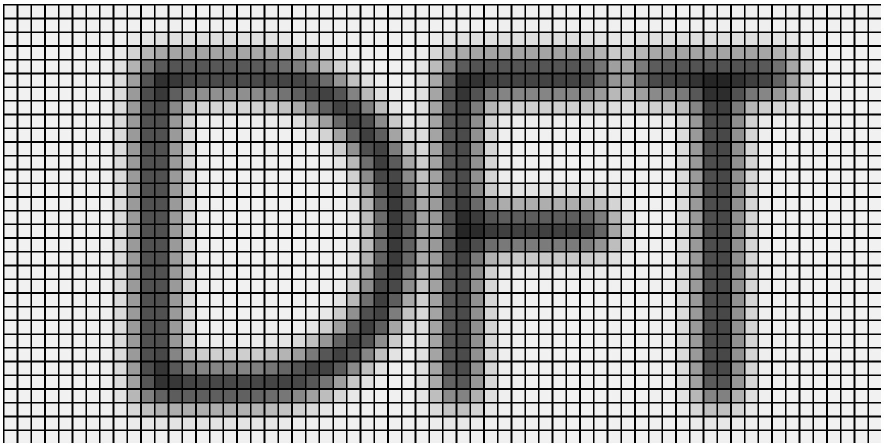

Figure 2.10: *An image can be thought to be composed of basis images in which only one pixel is unequal to zero.*

For many scientific applications, however, it is essential that a linear relation is maintained between the radiance of the observed object and the gray value in the digital image. Thus the gamma value must be set to one for these applications.

2.3 Wave Number Space and Fourier Transform

2.3.1 Vector Spaces

In Section 2.2, the discussion centered around the spatial representation of digital images. Without mentioning it explicitly, we thought of an image as composed of individual pixels (Fig. 2.10). Thus we can compose each image with basis images where just one pixel has a value of one while all other pixels are zero. We denote such a *basis image* with a one at row m, column n by

$$^{m,n}\boldsymbol{P}: \quad {}^{m,n}p_{m',n'} = \begin{cases} 1 & m = m' \wedge n = n' \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

Any arbitrary scalar image can then be composed from the basis images in Eq. (2.10) by

$$\boldsymbol{G} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} g_{m,n}\,{}^{m,n}\boldsymbol{P}, \tag{2.11}$$

where $g_{m,n}$ denotes the gray value at the position (m, n).

It is easy to convince ourselves that the basis images $^{m,n}\boldsymbol{P}$ form an *orthonormal base*. To that end we require an *inner product* (also known as *scalar product*) which can be defined similarly to the scalar product for vectors. The inner product of two images $\boldsymbol{G}$ and $\boldsymbol{H}$ is defined as

$$\langle \boldsymbol{G} | \boldsymbol{H} \rangle = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{m,n} h_{m,n}, \tag{2.12}$$

where the notation for the inner product from quantum mechanics is used in order to distinguish it from matrix multiplication, which is denoted by $\boldsymbol{GH}$.

From Eq. (2.12), we can immediately derive the *orthonormality relation* for the basis images ${}^{m,n}\boldsymbol{P}$:

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} {}^{m',n'}p_{m,n} \, {}^{m'',n''}p_{m,n} = \delta_{m'-m''} \delta_{n'-n''}. \tag{2.13}$$

This says that the inner product between two base images is zero if two different basis images are taken. The scalar product of a basis image with itself is one. The MN basis images thus span an $M \times N$-dimensional vector space over the set of real numbers.

The analogy to the well-known two- and three-dimensional vector spaces $\mathbb{R}^2$ and $\mathbb{R}^3$ helps us to understand how other representations for images can be gained. An $M \times N$ image represents a point in the $M \times N$ vector space. If we change the coordinate system, the image remains the same but its coordinates change. This means that we just observe the same piece of information from a different point of view. We can draw two important conclusions from this elementary fact. First, all representations are equivalent to each other. Each gives a complete representation of the image. Secondly, suitable coordinate transformations lead us from one representation to the other and back again.

From the manifold of other possible representations beside the spatial representation, only one has gained prime importance for image processing. Its base images are periodic patterns and the "coordinate transform" that leads to it is known as the *Fourier transform*. Figure 2.11 shows how the same image that has been composed by individual pixels in Fig. 2.10 is composed of periodic patterns.

A periodic pattern is first characterized by the distance between two maxima or the repetition length, the *wavelength* λ (Fig. 2.12). The direction of the pattern is best described by a vector normal to the lines of constant gray values. If we give this vector $\boldsymbol{k}$ the length $1/\lambda$

$$|\boldsymbol{k}| = 1/\lambda, \tag{2.14}$$

the wavelength and direction can be expressed by one vector, the *wave number* $\boldsymbol{k}$. The components of $\boldsymbol{k} = [k_1, k_2]^T$ directly give the number of wavelengths per unit length in the corresponding direction. The wave

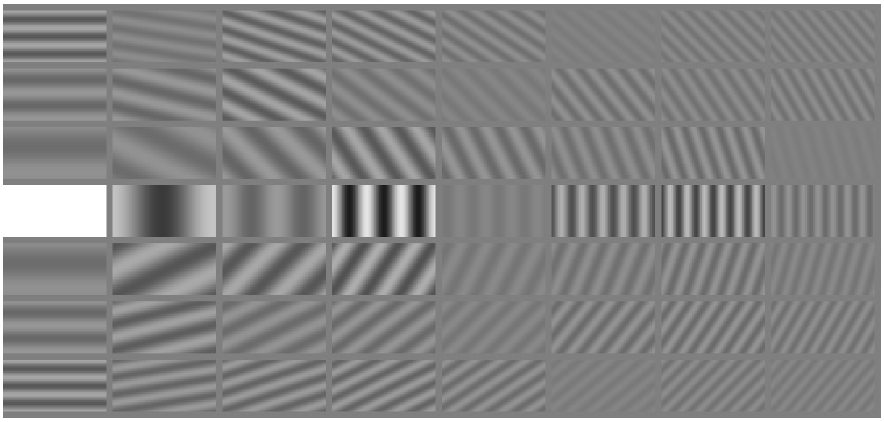

Figure 2.11: *The first 56 periodic patterns, the basis images of the Fourier transform, from which the image in Fig. 2.10 is composed.*

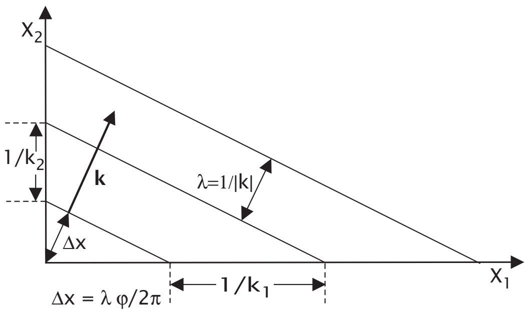

Figure 2.12: *Description of a 2-D periodic pattern by the wavelength λ, wave number **k**, and phase φ.*

number $\boldsymbol{k}$ can be used for the description of periodic patterns in any dimension.

In order to complete the description of a periodic pattern, two more quantities are required: the amplitude r and the relative position of the pattern at the origin (Fig. 2.12). The position is given as the distance Δx of the first maximum from the origin. Because this distance is at most a wavelength, it is best given as a *phase angle* $\varphi = 2\pi\Delta x/\lambda = 2\pi k \cdot \Delta x$ (Fig. 2.12) and the complete description of a periodic pattern is given by

$$r\cos(2\pi\boldsymbol{k}^T\boldsymbol{x} - \varphi). \tag{2.15}$$

This description is, however, mathematically quite awkward. We rather want a simple factor by which the base patterns have to be multiplied, in order to achieve a simple decomposition in periodic patterns. This is only possible by using *complex numbers* $\hat{g} = r\exp(-\mathrm{i}\varphi)$ and the complex exponential function $\exp(\mathrm{i}\varphi) = \cos\varphi + \mathrm{i}\sin\varphi$. The real part of $\hat{g}\exp(2\pi\mathrm{i}\boldsymbol{k}^T\boldsymbol{x})$ gives the periodic pattern in Eq. (2.15):

$$\Re(\hat{g}\exp(2\pi\mathrm{i}\boldsymbol{k}^T\boldsymbol{x})) = r\cos(2\pi\boldsymbol{k}^T\boldsymbol{x} - \varphi). \tag{2.16}$$

In this way the decomposition into periodic patterns requires the extension of real numbers to complex numbers. A real-valued image is thus considered as a complex-valued image with a zero imaginary part.

The subject of the remainder of this chapter is rather mathematical, but it forms the base for image representation and low-level image processing. After introducing both the continuous and discrete Fourier transform in Sections 2.3.2 and 2.3.3, we will discuss all properties of the Fourier transform that are of relevance to image processing in Section 2.3.4. We will take advantage of the fact that we are dealing with images, which makes it easy to illustrate some complex mathematical relations.

2.3.2 One-Dimensional Fourier Transform

First, we will consider the *one-dimensional Fourier transform.*

Definition 2.1 (1-D FT) *If $g(x) : \mathbb{R} \mapsto \mathbb{C}$ is a square integrable function, that is,*

$$\int_{-\infty}^{\infty} |g(x)|^2\,\mathrm{d}x < \infty, \tag{2.17}$$

then the Fourier transform *of $g(x)$, $\hat{g}(k)$ is given by*

$$\boxed{\hat{g}(k) = \int_{-\infty}^{\infty} g(x)\exp(-2\pi\mathrm{i}kx)\,\mathrm{d}x.} \tag{2.18}$$

The Fourier transform maps the vector space of square integrable functions onto itself. The inverse Fourier transform *of $\hat{g}(k)$ results in the original function $g(x)$:*

$$\boxed{g(x) = \int_{-\infty}^{\infty} \hat{g}(k)\exp(2\pi\mathrm{i}kx)\,\mathrm{d}k.} \tag{2.19}$$

The Fourier transform can be written in a more compact form if the abbreviation

$$\mathrm{w} = \mathrm{e}^{2\pi\mathrm{i}} \tag{2.20}$$

is used and the integral is written as an *inner product*:

$$\langle g(x)\,|h(x)\rangle = \int_{-\infty}^{\infty} g^*(x)h(x)\mathrm{d}x, \tag{2.21}$$

where * denotes the conjugate complex. Then

$$\hat{g}(k) = \left\langle \mathrm{w}^{kx} \,|\, g(x) \right\rangle . \tag{2.22}$$

The function w^t can be visualized as a vector that rotates anticlockwise on the *unit circle* in the *complex plane*. The variable t gives the number of revolutions.

Sometimes, it is also convenient to use an operator notation for the Fourier transform:

$$\hat{g} = \mathcal{F} g \quad \text{and} \quad g = \mathcal{F}^{-1} \hat{g}. \tag{2.23}$$

A function and its transform, a *Fourier transform pair*, is simply denoted by $g(x) \circ\!\!-\!\!\bullet\, \hat{g}(k)$.

For the *discrete Fourier transform* (*DFT*), the wave number is now an integer number that indicates how many wavelengths fit into the interval with N elements.

Definition 2.2 (1-D DFT) *The DFT maps an ordered N-tuple of complex numbers g_n, the* complex-valued vector

$$\boldsymbol{g} = \left[g_0, g_1, \ldots, g_{N-1}\right]^T, \tag{2.24}$$

onto another vector $\hat{\boldsymbol{g}}$ of a vector space with the same dimension N.

$$\boxed{\hat{g}_v = \frac{1}{N} \sum_{n=0}^{N-1} g_n \exp\left(-\frac{2\pi \mathrm{i} n v}{N}\right), \quad 0 \le v < N.} \tag{2.25}$$

The back transformation is given by

$$\boxed{g_n = \sum_{v=0}^{N-1} \hat{g}_v \exp\left(\frac{2\pi \mathrm{i} n v}{N}\right), \quad 0 \le n < N.} \tag{2.26}$$

Why we use an asymmetric definition for the DFT here is explained in Section 2.3.6.

Again it is useful to use a convenient abbreviation for the kernel of the DFT; compare Eq. (2.20):

$$\mathrm{w}_N = \mathrm{w}^{1/N} = \exp\left(\frac{2\pi \mathrm{i}}{N}\right). \tag{2.27}$$

As the continuous Fourier transform, the DFT can be considered as the inner product of the vector $\boldsymbol{g}$ with a set of N orthonormal basis vectors

$$\boldsymbol{b}_v = \frac{1}{\sqrt{N}} \left[\mathrm{w}_N^0, \mathrm{w}_N^v, \mathrm{w}_N^{2v}, \ldots, \mathrm{w}_N^{(N-1)v}\right]^T. \tag{2.28}$$

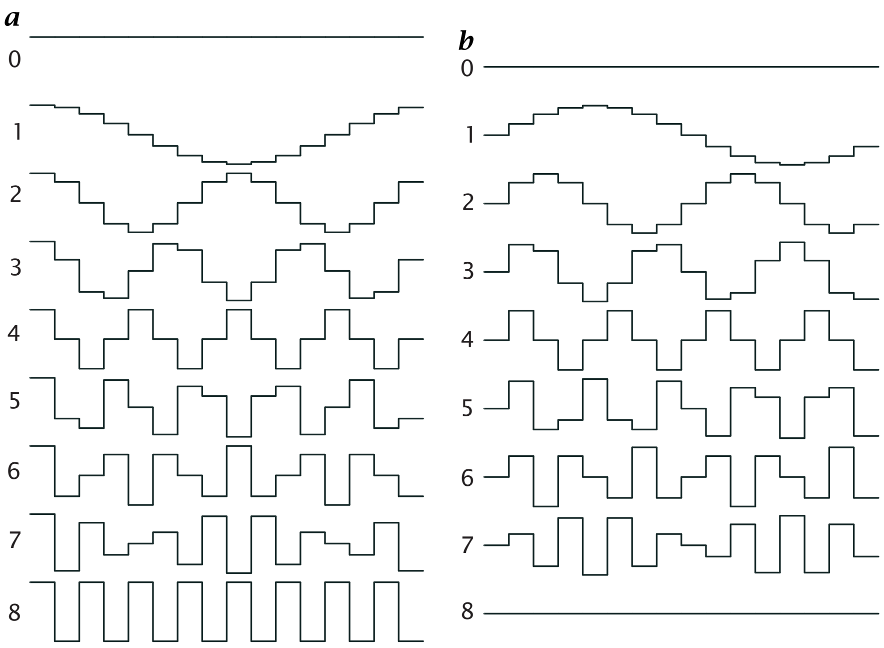

Figure 2.13: *The first 9 basis functions of the DFT for* $N = 16$*;* ***a*** *real part (cosine function),* ***b*** *imaginary part (sine function).*

Then

$$\hat{g}_v = \frac{1}{N}\sum_{n=0}^{N-1} \mathrm{w}_N^{-nv}\, g_n = \frac{1}{\sqrt{N}}\,\langle \boldsymbol{b}_v\,|\boldsymbol{g}\,\rangle = \frac{1}{\sqrt{N}}\boldsymbol{b}_v{}^T\boldsymbol{g}. \tag{2.29}$$

Note the second compact notation of the *scalar product* on the right-hand side of the equation using the superscript T that includes to take the complex conjugate of the first vector.

Equation (2.29) means that the coefficient $\hat{g}_v$ in the Fourier space is obtained by projecting the vector $\boldsymbol{g}$ onto the basis vector $\boldsymbol{b}_v$. The N basis vectors $\boldsymbol{b}_v$ are orthogonal to each other:

$$\boldsymbol{b}_v{}^T\boldsymbol{b}_{v'} = \delta_{v-v'} = \begin{cases} 1 & v = v' \\ 0 & \text{otherwise.} \end{cases} \tag{2.30}$$

Consequently, the set $\boldsymbol{b}_v$ forms an orthonormal basis for the *vector space*, which means that each vector of the vector space can be expressed as a linear combination of the basis vectors of the Fourier space. The DFT calculates the projections of the vector $\boldsymbol{g}$ onto all basis vectors directly, i. e., the components of $\boldsymbol{g}$ in the direction of the basis vectors. In this sense, the DFT is just a special type of coordinate transformation in an M-dimensional vector space. Mathematically, the DFT differs from

Table 2.1: *Comparison of the continuous Fourier transform (FT), the Fourier series (FS), the infinite discrete Fourier transform (IDFT), and the discrete Fourier transform (DFT) in one dimension* ($\mathrm{w} = \mathrm{e}^{2\pi \mathrm{i}}$).

Type	Forward transform	Backward transform
FT: $x, k \in \mathbb{R}$	$\hat{g}(k) = \int\limits_{-\infty}^{\infty} g(x)\mathrm{w}^{-kx}\mathrm{d}x$	$g(x) = \int\limits_{-\infty}^{\infty} \hat{g}(k)\mathrm{w}^{kx}\mathrm{d}k$
FS: $x \in [0, \Delta x]$, $v \in \mathbb{Z}$	$\hat{g}_v = \dfrac{1}{\Delta x}\int\limits_{0}^{\Delta x} g(x)\mathrm{w}^{-vx/\Delta x}\mathrm{d}x$	$g(x) = \sum\limits_{v=-\infty}^{\infty} \hat{g}_v \mathrm{w}^{vx/\Delta x}$
IDFT: $n \in \mathbb{Z}$, $k \in [0, 1/\Delta x]$	$\hat{g}(k) = \sum\limits_{n=-\infty}^{\infty} g_n \mathrm{w}^{-nk\Delta x}$	$g_n = \Delta x \int\limits_{0}^{1/\Delta x} \hat{g}(k)\mathrm{w}^{nk\Delta x}\mathrm{d}k$
DFT: $n, v \in \mathbb{Z}_N$	$\hat{g}_v = \dfrac{1}{N}\sum\limits_{n=0}^{N-1} g_n \mathrm{w}_N^{-vn}$	$g_n = \sum\limits_{v=0}^{N-1} \hat{g}_v \mathrm{w}_N^{vn}$

more familiar coordinate transformations such as rotation in a three-dimensional vector space (Section 7.2.2) only because the vector space is over the field of the complex instead of real numbers and has many more dimensions.

The real and imaginary parts of the basis vectors are sampled sine and cosine functions of different wavelengths (Fig. 2.13). The index v denotes how often the wavelength of the function fits into the interval $[0, M]$. The basis vector $\boldsymbol{b}_0$ is a constant real vector. The projection onto this vector results in the mean of the elements of the vector $\boldsymbol{g}$ multiplied by $\sqrt{N}$.

Besides the continuous and discrete Fourier transforms there are two other forms you may be familiar with: the *Fourier series* (*FS*) that maps a function in a finite interval $[0, \Delta x]$ to an infinite series of coefficients and the *infinite discrete Fourier transform* (*IDFT*) that maps an infinite series of complex numbers to a finite interval $[0, 1/\Delta x]$ in the Fourier space. Therefore it is illustrative to compare the DFT with these transforms (Table 2.1).

2.3.3 Multidimensional Fourier transform

The Fourier transform can easily be extended to *multidimensional* signals.

Definition 2.3 (Multidimensional FT) *If* $g(\boldsymbol{x}) : \mathbb{R}^W \mapsto \mathbb{C}$ *is a square integrable function, that is,*

$$\int\limits_{-\infty}^{\infty} |g(\boldsymbol{x})|^2 \,\mathrm{d}^W x = \langle g(\boldsymbol{x}) \,|g(\boldsymbol{x}) \rangle = \|g(\boldsymbol{x})\|_2^2 < \infty \tag{2.31}$$

then the Fourier transform *of $g(\boldsymbol{x})$, $\hat{g}(\boldsymbol{k})$ is given by*

$$\boxed{\hat{g}(\boldsymbol{k}) = \int_{-\infty}^{\infty} g(\boldsymbol{x}) \exp\left(-2\pi \mathrm{i} \boldsymbol{k}^T \boldsymbol{x}\right) \mathrm{d}^W x = \left\langle \mathrm{w}^{\boldsymbol{x}^T \boldsymbol{k}} \,|\, g(\boldsymbol{x}) \right\rangle} \tag{2.32}$$

and the inverse Fourier transform *by*

$$\boxed{g(\boldsymbol{x}) = \int_{-\infty}^{\infty} \hat{g}(\boldsymbol{k}) \exp\left(2\pi \mathrm{i} \boldsymbol{k}^T \boldsymbol{x}\right) \mathrm{d}^W k = \left\langle \mathrm{w}^{-\boldsymbol{x}^T \boldsymbol{k}} \,|\, \hat{g}(\boldsymbol{k}) \right\rangle.} \tag{2.33}$$

The *scalar product* in the exponent of the kernel $\boldsymbol{x}^T\boldsymbol{k}$ makes the kernel of the Fourier transform separable, that is, it can be written as

$$\mathrm{w}^{\boldsymbol{x}^T \boldsymbol{k}} = \prod_{p=1}^{W} \mathrm{w}^{k_p x_p}. \tag{2.34}$$

The discrete Fourier transform is discussed here for two dimensions. The extension to higher dimensions is straightforward.

Definition 2.4 (2-D DFT) *The 2-D DFT maps complex-valued $M \times N$ matrices on complex-valued $M \times N$ matrices:*

$$\boxed{\hat{g}_{u,v} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{m,n} \exp\left(-\frac{2\pi \mathrm{i} m u}{M}\right) \exp\left(-\frac{2\pi \mathrm{i} n v}{N}\right)} \tag{2.35}$$

or

$$\hat{g}_{u,v} = \frac{1}{MN} \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} g_{m,n} \mathrm{w}_N^{-nv} \right) \mathrm{w}_M^{-mu}. \tag{2.36}$$

In the second line, the abbreviation defined in Eq. (2.27) is used. As in the one-dimensional case, the DFT expands a matrix into a set of NM basis matrices which span the $N \times M$-dimensional vector space over the field of complex numbers. The basis matrices are of the form

$$\underbrace{\boldsymbol{B}_{u,v}}_{M \times N} = \frac{1}{\sqrt{MN}} \begin{bmatrix} \mathrm{w}^0 \\ \mathrm{w}_M^u \\ \mathrm{w}_M^{2u} \\ \vdots \\ \mathrm{w}_M^{(M-1)u} \end{bmatrix} \left[\mathrm{w}^0, \mathrm{w}_N^v, \mathrm{w}_N^{2v}, \ldots, \mathrm{w}_N^{(N-1)v} \right]. \tag{2.37}$$

In this equation, the basis matrices are expressed as an *outer product* of the column and the row vector that form the basis vectors of the one-dimensional DFT (Eq. (2.28)). This reflects the separability of the kernel of the 2-D DFT.

Then the 2-D DFT can again be written as an inner product

$$\hat{g}_{u,v} = \frac{1}{\sqrt{MN}} \langle \boldsymbol{B}_{u,v} | \boldsymbol{G} \rangle , \tag{2.38}$$

where the inner product of two complex-valued matrices is given by

$$\langle \boldsymbol{G} | \boldsymbol{H} \rangle = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g^*_{m,n} h_{m,n}. \tag{2.39}$$

The inverse 2-D DFT is given by

$$\boxed{g_{mn} = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \hat{g}_{u,v} \mathrm{w}_M^{mu} \mathrm{w}_N^{nv} = \sqrt{MN} \left\langle \boldsymbol{B}_{-m,-n} \middle| \hat{\boldsymbol{G}} \right\rangle .} \tag{2.40}$$

2.3.4 Properties of the Fourier transform

In this section we discuss the basic properties of the continuous and discrete Fourier transform. We point out those properties of the FT that are most significant for image processing. Together with some basic Fourier transform pairs (≻ R5), these general properties (≻ R4, ≻ R7) form a powerful framework with which further properties of the Fourier transform and the transforms of many functions can be derived without much effort.

Periodicity of DFT. The kernel of the DFT in Eq. (2.25) shows a characteristic *periodicity*

$$\exp\left(-\frac{2\pi \mathrm{i}(n + lN)}{N}\right) = \exp\left(-\frac{2\pi \mathrm{i} n}{N}\right), \mathrm{w}_N^{(n+lN)} = \mathrm{w}_N^n, \quad \forall\, l \in \mathbb{Z}. \tag{2.41}$$

The definitions of the DFT restrict the spatial domain and the Fourier domain to a finite number of values. If we do not care about this restriction and calculate the forward and back transformation for all integer numbers, we find from Eqs. (2.38) and (2.40) the same periodicities for functions in the space and Fourier domain:

$$\begin{array}{lll} \text{wave number domain} & \hat{g}_{u+kM,v+lN} = \hat{g}_{u,v}, & \forall\, k,l \in \mathbb{Z}, \\ \text{space domain} & g_{m+kM,n+lN} = g_{m,n}, & \forall\, k,l \in \mathbb{Z}. \end{array} \tag{2.42}$$

These equations state a periodic replication in all directions in both domains beyond the original definition range. The periodicity of the DFT gives rise to an interesting geometric interpretation. In the one-dimensional case, the border points g_{N-1} and $g_N = g_0$ are neighboring points. We can interpret this property geometrically by drawing the points of the vector not on a finite line but on a circle, the so-called

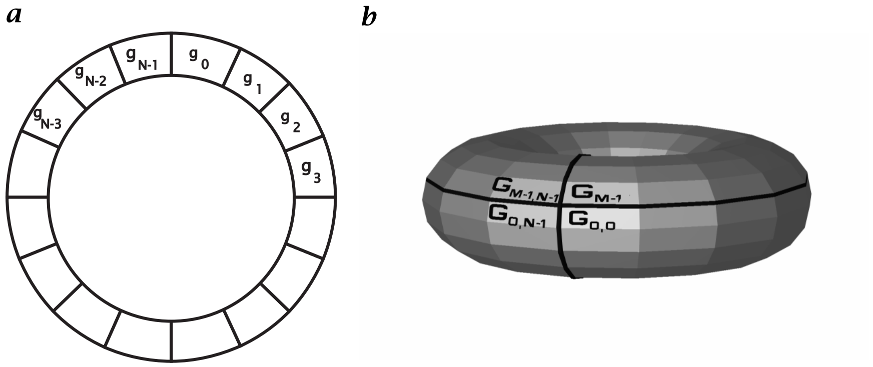

Figure 2.14: *Geometric interpretation of the* periodicity *of the one- and two-dimensional DFT with* ***a*** *the Fourier ring and* ***b*** *the Fourier torus.*

Fourier ring (Fig. 2.14a). This representation has a deeper meaning when we consider the Fourier transform as a special case of the *z-transform* [148]. With two dimensions, a matrix is mapped onto a torus (Fig. 2.14b), the *Fourier torus*.

Symmetries. Four types of *symmetries* are important for the Fourier transform:

$$\begin{array}{lrcr} \text{even} & g(-\boldsymbol{x}) & = & g(\boldsymbol{x}), \\ \text{odd} & g(-\boldsymbol{x}) & = & -g(\boldsymbol{x}), \\ \text{Hermitian} & g(-\boldsymbol{x}) & = & g^*(\boldsymbol{x}), \\ \text{anti-Hermitian} & g(-\boldsymbol{x}) & = & -g^*(\boldsymbol{x}) \end{array} \tag{2.43}$$

The symbol $*$ denotes the complex conjugate. The *Hermitian symmetry* is of importance because the kernels of the FT Eq. (2.18) and DFT Eq. (2.25) are Hermitian.

Any function $g(\boldsymbol{x})$ can be split into its even and odd parts by

$$\begin{aligned} {}^e g(\boldsymbol{x}) &= \frac{g(\boldsymbol{x}) + g(-\boldsymbol{x})}{2}, \\ {}^o g(\boldsymbol{x}) &= \frac{g(\boldsymbol{x}) - g(-\boldsymbol{x})}{2}. \end{aligned} \tag{2.44}$$

With this partition, the Fourier transform can be split into a cosine and a sine transform:

$$\hat{g}(\boldsymbol{k}) = 2\int_0^\infty {}^e g(\boldsymbol{x}) \cos(2\pi \boldsymbol{k}^T \boldsymbol{x}) \mathrm{d}^W x + 2\mathrm{i}\int_0^\infty {}^o g(\boldsymbol{x}) \sin(2\pi \boldsymbol{k}^T \boldsymbol{x}) \mathrm{d}^W x. \tag{2.45}$$

It follows that if a function is even or odd, its transform is also even or odd. The full symmetry relations are:

$$\begin{array}{lcl}
\text{real} & \circ\!\!-\!\!\!-\!\!\bullet & \text{Hermitian} \\
\text{imaginary} & \circ\!\!-\!\!\!-\!\!\bullet & \text{anti-Hermitian} \\
\text{Hermitian} & \circ\!\!-\!\!\!-\!\!\bullet & \text{real} \\
\text{anti-Hermitian} & \circ\!\!-\!\!\!-\!\!\bullet & \text{imaginary} \\
\text{even} & \circ\!\!-\!\!\!-\!\!\bullet & \text{even} \\
\text{odd} & \circ\!\!-\!\!\!-\!\!\bullet & \text{odd} \\
\text{real and even} & \circ\!\!-\!\!\!-\!\!\bullet & \text{real and even} \\
\text{real and odd} & \circ\!\!-\!\!\!-\!\!\bullet & \text{imaginary and odd} \\
\text{imaginary and even} & \circ\!\!-\!\!\!-\!\!\bullet & \text{imaginary and even} \\
\text{imaginary and odd} & \circ\!\!-\!\!\!-\!\!\bullet & \text{real and odd}
\end{array} \tag{2.46}$$

The DFT shows the same symmetries as the FT (Eqs. (2.43) and (2.46)). In the definition for even and odd functions $g(-\boldsymbol{x}) = \pm g(\boldsymbol{x})$ only $\boldsymbol{x}$ must be replaced by the corresponding indices: $g_{-n} = \pm g_n$ or $g_{-m,-n} = \pm g_{m,n}$. Note that because of the periodicity of the DFT, these symmetry relations can also be written as

$$g_{-m,-n} = \pm g_{m,n} \equiv g_{M-m,N-n} = \pm g_{m,n} \tag{2.47}$$

for even (+ sign) and odd (− sign) functions. This is equivalent to shifting the symmetry center from the origin to the point $[M/2, N/2]^T$.

The study of symmetries is important for practical purposes. Careful consideration of symmetry allows storage space to be saved and algorithms to speed up. Such a case is real-valued images. Real-valued images can be stored in half of the space as complex-valued images. From the symmetry relation Eq. (2.46) we can conclude that real-valued functions exhibit a Hermitian DFT:

$$\begin{array}{rcl}
g_n = g_n^* & \circ\!\!-\!\!\!-\!\!\bullet & \hat{g}_{N-v} = \hat{g}_v^*, \\
g_{mn} = g_{mn}^* & \circ\!\!-\!\!\!-\!\!\bullet & \hat{g}_{M-u,N-v} = \hat{g}_{uv}^*.
\end{array} \tag{2.48}$$

The complex-valued DFT of real-valued vectors is, therefore, completely determined by the values in one half-space. The other half-space is obtained by mirroring at the symmetry center $[N/2]$. Consequently, we need the same amount of storage space for the DFT of a real vector as for the vector itself, as only half of the complex spectrum needs to be stored.

In two and higher dimensions, matters are slightly more complex. Again, the Fourier transform of a real-valued image is determined completely by the values in one half-space, but there are many ways to select the half-space. This means that only *one* component of the wave number is limited to positive values.

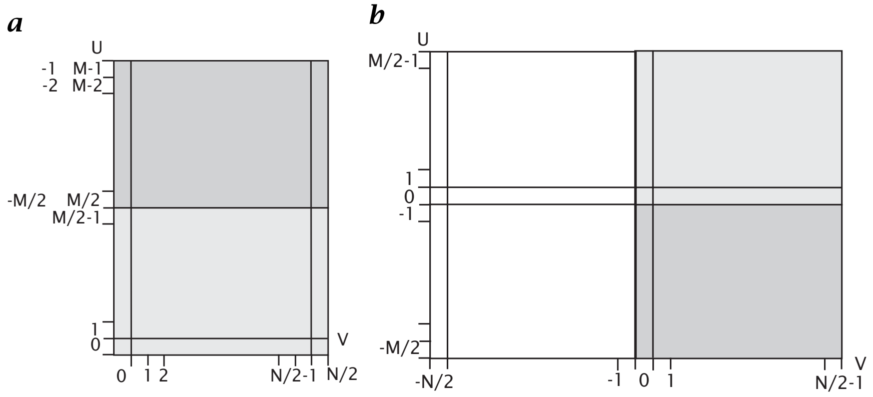

Figure 2.15: ***a*** *Half-space as computed by an in-place Fourier transform algorithm; the wave number zero is in the lower left corner;* ***b*** *FT with the missing half appended and remapped so that the wave number zero is in the center.*

The Fourier transform of a real $M \times N$ image can be represented by M rows and $N/2+1$ columns (Fig. 2.15) assuming that N is even. Unfortunately, $N/2+1$ columns are required, because the first ($m = 0$) and last column ($m = M/2$) are symmetric to themselves according to Eq. (2.48). Thus it appears impossible to overwrite a real image by its complex Fourier transform because we need one more column. A closer examination shows that it works nevertheless. The first and last columns are real-valued because of symmetry reasons ($\hat{g}_{0,N-v} = \hat{g}^*_{0,v}$ and $\hat{g}_{M/2,N-v} = \hat{g}^*_{M/2,v}$). Therefore, the real part of column $M/2$ can be stored in the imaginary part of column 0.

For real-valued image sequences, again we need only a half-space to represent the spectrum. Physically, it makes the most sense to choose the half-space that contains positive frequencies. In contrast to a single image, we obtain the full wave-number space. Now we can identify the spatially identical wave numbers $\boldsymbol{k}$ and $-\boldsymbol{k}$ as structures propagating in opposite directions.

Separability. The kernel of the Fourier transform is separable (Eq. (2.34)). Therefore, the transform of a separable function is also separable:

$$\prod_{p=1}^{W} g(x_p) \circ\!\!-\!\!\bullet \prod_{p=1}^{W} \hat{g}(k_p). \tag{2.49}$$

This property is essential to compute transforms of multidimensional functions efficiently from 1-D transforms because many of them are separable.

Similarity. The similarity theorem states how scaling of the coordinate system influences the Fourier transform. In one dimension, a function can only be scaled ($x' = ax$). In multiple dimensions, the coordinate system can be transformed in a more general way by an affine transform ($\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x}$), i. e., the new basis vectors are linear combinations of the old basis vectors. A special case is the rotation of the coordinate system.

Theorem 2.1 (Similarity) *Let a be a non-zero real number, $\boldsymbol{A}$ a real, invertible matrix, and $\boldsymbol{R}$ an orthogonal matrix representing a rotation of the coordinate system ($\boldsymbol{R}^{-1} = \boldsymbol{R}^T$, $\det \boldsymbol{R} = 1$). Then the following similarity relations hold:*

$$\boxed{\begin{array}{llcc} \textit{Scalar} & g(a\boldsymbol{x}) & \circ\!\!-\!\!\bullet & \dfrac{1}{|a|^W}\hat{g}(\boldsymbol{k}/a), \\ \textit{Affine transform} & g(\boldsymbol{A}\boldsymbol{x}) & \circ\!\!-\!\!\bullet & \dfrac{1}{\det \boldsymbol{A}}\hat{g}((\boldsymbol{A}^T)^{-1}\boldsymbol{k}), \\ \textit{Rotation} & g(\boldsymbol{R}\boldsymbol{x}) & \circ\!\!-\!\!\bullet & \hat{g}(\boldsymbol{R}\boldsymbol{k}). \end{array}} \tag{2.50}$$

If a function is squeezed in the spatial domain, it is stretched in the Fourier domain, and vice versa. A rotation of the coordinate system in the spatial domain causes the identical rotation in the Fourier domain.

The above similarity theorems do not apply to the discrete Fourier transform because an arbitrary scaling and rotation is not possible. A stretching of a discrete function is only possible by an integer factor K (*upsampling*) and the newly generated discrete points are filled with zeros:

$$(\boldsymbol{g}_{\uparrow K})_n = \begin{cases} g_{n/K} & n = 0, K, 2K, \ldots (N-1)K) \\ 0 & \text{otherwise.} \end{cases} \tag{2.51}$$

Theorem 2.2 (Similarity, discrete) *Let $\boldsymbol{g}$ be a complex-valued vector with N elements and $K \in \mathbb{N}$. Then the discrete Fourier transform of the upsampled vector $\boldsymbol{g}_{\uparrow K}$ with KN elements is given by*

$$\boxed{\boldsymbol{g}_{\uparrow K} \quad \circ\!\!-\!\!\bullet \quad \frac{1}{K}\hat{\boldsymbol{g}} \quad \text{with} \quad \hat{g}_{kN+v} = \hat{g}_v.} \tag{2.52}$$

Upsampling by a factor K thus simply results in a K-fold replication of the Fourier transform. Note that because of the periodicity of the discrete Fourier transform discussed at the beginning of this section, $\hat{g}_{kN+v} = \hat{g}_v$.

Shift. In Section 2.3.1 we discussed some properties of the basis images of the Fourier space, the complex exponential functions $\exp\left(2\pi \mathrm{i}\boldsymbol{k}^T\boldsymbol{x}\right)$. A spatial shift of these functions causes a multiplication by a phase factor:

$$\exp\left(2\pi \mathrm{i}(\boldsymbol{x} - \boldsymbol{x}_0)^T\boldsymbol{k}\right) = \exp\left(-2\pi \mathrm{i}\boldsymbol{x}_0^T\boldsymbol{k}\right)\exp\left(2\pi \mathrm{i}\boldsymbol{k}^T\boldsymbol{x}\right). \tag{2.53}$$

As a direct consequence of the linearity of the Fourier transform, we can formulate the following *shift theorem*:

Theorem 2.3 (Shift) *If the function $g(\boldsymbol{x})$ has the Fourier transform $\hat{g}(\boldsymbol{k})$, then $g(\boldsymbol{x} - \boldsymbol{x}_0)$ has the Fourier transforms* $\exp(-2\pi \mathrm{i} \boldsymbol{x}_0^T \boldsymbol{k})\hat{g}(\boldsymbol{k})$.

Thus, a shift in the spatial domain does not change the Fourier transform except for a wave number-dependent phase change $-2\pi \boldsymbol{x}_0^T \boldsymbol{k}$.

The shift theorem can also be applied in the Fourier domain. A shift in the Fourier space, $\hat{g}(\boldsymbol{k} - \boldsymbol{k}_0)$, results in a signal in the spatial domain that is modulated by a complex exponential with the wave number vector $\boldsymbol{k}_0$: $\exp(2\pi \mathrm{i} \boldsymbol{k}_0^T \boldsymbol{x}) g(\boldsymbol{x})$.

Convolution. *Convolution* is one of the most important operations for signal processing. For a continuous signal it is defined by

$$(g * h)(\boldsymbol{x}) = \int_{-\infty}^{\infty} h(\boldsymbol{x}') g(\boldsymbol{x} - \boldsymbol{x}') \mathrm{d}^W x'. \tag{2.54}$$

In signal processing, the function $h(\boldsymbol{x})$ is normally zero except for a small area around zero and is often denoted as the *convolution mask*. Thus, the convolution with $h(\boldsymbol{x})$ results in a new function $g'(\boldsymbol{x})$ whose values are a kind of weighted average of $g(\boldsymbol{x})$ in a small neighborhood around $\boldsymbol{x}$. It changes the signal in a defined way, for example, makes it smoother. Therefore it is also called a *filter*.

One- and two-dimensional discrete convolution are defined analogous to Eq. (2.54) by

$$g'_n = \sum_{n'=0}^{N-1} h_{n'} g_{n-n'}, \quad g'_{m,n} = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} h_{m'n'} g_{m-m',n-n'} \tag{2.55}$$

The *convolution theorem* for the FT and DFT states:

Theorem 2.4 (Convolution) *If $g(\boldsymbol{x})$ $(\boldsymbol{g}, \boldsymbol{G})$ has the Fourier transforms $\hat{g}(\boldsymbol{k})$ $(\hat{\boldsymbol{g}}, \hat{\boldsymbol{G}})$ and $h(\boldsymbol{x})$, $(\boldsymbol{h}, \boldsymbol{H})$ has the Fourier transforms $\hat{h}(\boldsymbol{k})$ $(\hat{\boldsymbol{h}}, \hat{\boldsymbol{H}})$, then $h * g$ $(\boldsymbol{h} * \boldsymbol{g}, \boldsymbol{H} * \boldsymbol{G})$ has the Fourier transforms $\hat{h}(\boldsymbol{k})\hat{g}(\boldsymbol{k})$, $(N\hat{\boldsymbol{h}}\hat{\boldsymbol{g}}, MN\hat{\boldsymbol{H}}\hat{\boldsymbol{G}})$:*

$$\boxed{\begin{array}{llcl} \textit{FT:} & h(\boldsymbol{x}) * g(\boldsymbol{x}) & \circ\!\!-\!\!\bullet & \hat{h}(\boldsymbol{k})\hat{g}(\boldsymbol{k}), \\ \textit{1-D DFT:} & \boldsymbol{h} * \boldsymbol{g} & \circ\!\!-\!\!\bullet & N\hat{\boldsymbol{h}}\hat{\boldsymbol{g}}, \\ \textit{2-D DFT:} & \boldsymbol{H} * \boldsymbol{G} & \circ\!\!-\!\!\bullet & MN\hat{\boldsymbol{H}}\hat{\boldsymbol{G}}. \end{array}} \tag{2.56}$$

Thus, convolution of two functions means multiplication of their transforms. Likewise, convolution of two functions in the Fourier domain means multiplication in the space domain. The simplicity of convolution in the Fourier space stems from the fact that the base functions of the Fourier domain, the complex exponentials $\exp\left(2\pi \mathrm{i} \boldsymbol{k}^T \boldsymbol{x}\right)$, are joint eigenfunctions of all convolution operators. This means that a convolution operator does not change these functions except for the multiplication by a factor.

From the convolution theorem, the following properties are immediately evident. Convolution is

$$\begin{array}{ll} \text{commutative} & h * g = g * h, \\ \text{associative} & h_1 * (h_2 * g) = (h_1 * h_2) * g, \\ \text{distributive over addition} & (h_1 + h_2) * g = h_1 * g + h_2 * g. \end{array} \tag{2.57}$$

In order to grasp the importance of these properties of convolution, we note that two operations that do not look so at first glance, are also convolution operations: the shift operation and all derivative operators.

In both cases the Fourier transform is only multiplied by a complex factor. For a shift operation this can be seen directly from the shift theorem (Theorem 2.3). The convolution mask of a shift operator S is a shifted δ distribution:

$$S(\boldsymbol{s})g(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{s}) * g(\boldsymbol{x}). \tag{2.58}$$

For a partial derivative of a function in the spatial domain the *differentiation theorem* states:

Theorem 2.5 (Differentiation) *If $g(\boldsymbol{x})$ is differentiable for all $\boldsymbol{x}$ and has the Fourier transform $\hat{g}(\boldsymbol{k})$, then the Fourier transform of the partial derivative $\partial g(\boldsymbol{x})/\partial x_p$ is $2\pi \mathrm{i} k_p \hat{g}(\boldsymbol{k})$:*

$$\boxed{\frac{\partial g(\boldsymbol{x})}{\partial x_p} \quad \circ\!\!-\!\!\bullet \quad 2\pi \mathrm{i} k_p \hat{g}(\boldsymbol{k}).} \tag{2.59}$$

The differentiation theorem results directly from the definition of the inverse Fourier transform in Eq. (2.33) by interchanging the partial derivative with the Fourier integral.

The inverse Fourier transform of $2\pi i k_1$, that is, the corresponding convolution mask, is no longer an ordinary function ($2\pi i k_1$ is not absolutely integrable) but the derivative of the δ distribution:

$$2\pi i k \quad \circ\!\!-\!\!\bullet \quad \delta'(x) = \frac{\mathrm{d}\delta(x)}{\mathrm{d}x} = \lim_{a \to 0} \frac{\mathrm{d}}{\mathrm{d}x} \left(\frac{\exp(-\pi x^2/a^2)}{a} \right). \tag{2.60}$$

Of course, the derivation of the δ distribution exists—as all properties of distributions—only in the sense as a limit of a sequence of functions as shown in the preceding equation.

With the knowledge of derivative and shift operators being convolution operators, we can use the properties summarized in Eq. (2.57) to draw some important conclusions. As any convolution operator commutes with the shift operator, convolution is a shift-invariant operation.

Furthermore, we can first differentiate a signal and then perform a convolution operation or vice versa and obtain the same result. The properties in Eq. (2.57) are essential for an effective computation of convolution operations.

Central-limit theorem. The central-limit theorem is mostly known for its importance in the theory of probability [149]. However, it also plays an important role for signal processing as it is a rigorous statement of the tendency that cascaded convolution tends to approach Gaussian form ($\propto \exp(-ax^2)$). Because the Fourier transform of the Gaussian is also a Gaussian (≻ R6), this means that *both* the Fourier transform (the transfer function) and the mask of a convolution approach Gaussian shape.

Thus the central-limit theorem is central to the unique role of the Gaussian function for signal processing. The sufficient conditions under which the central-limit theorem is valid can be formulated in different ways. We use here the conditions from [149] and express the theorem with respect to convolution.

Theorem 2.6 (Central-limit theorem) *Given N functions $h_n(x)$ with a zero mean $\int_{-\infty}^{\infty} x h_n(x)\mathrm{d}x$ and the variance $\sigma_n^2 = \int_{-\infty}^{\infty} x^2 h_n(x)\mathrm{d}x$ with $z = x/\sigma$, $\sigma^2 = \sum_{n=1}^{N} \sigma_n^2$ then*

$$h = \lim_{N\to\infty} h_1 * h_2 * \ldots * h_N \propto \exp(-z^2/2) \qquad (2.61)$$

provided that

$$\lim_{N\to\infty} \sum_{n=1}^{N} \sigma_n^2 \to \infty \qquad (2.62)$$

and there exists a number $\alpha > 2$ and a finite constant c such that

$$\int_{-\infty}^{\infty} x^{\alpha} h_n(x)\mathrm{d}x < c < \infty \quad \forall n. \qquad (2.63)$$

The theorem is of great practical importance because — especially if h_n is smooth — the Gaussian shape is approximated sufficiently accurately for values of N as low as 5.

Smoothness and compactness. The smoother a function is, the more compact is its Fourier transform. This general rule can be formulated more quantitatively if we express the smoothness by the number of derivatives that are continuous and the compactness by the asymptotic behavior for large values of k. Then we can state: If a function $g(x)$ and its first $n-1$ derivatives are continuous, its Fourier transform decreases at least as rapidly as $|k|^{-(n+1)}$ for large k, that is, $\lim_{|k|\to\infty} |k|^n g(k) = 0$.

As simple examples we can take the box and triangle functions (see next section). The box function is discontinuous ($n = 0$), its Fourier transform, the sinc function, decays with $|k|^{-1}$. In contrast, the triangle function is continuous, but its first derivative is discontinuous. Therefore, its Fourier transform, the sinc^2 function, decays steeper with $|k|^{-2}$. In order to include also impulsive functions (δ distributions) in this relation, we note that the derivative of a discontinuous function becomes impulsive. Therefore, we can state: If the nth derivative of a function becomes impulsive, the function's Fourier transform decays with $|k|^{-n}$.

The relation between smoothness and compactness is an extension of reciprocity between the spatial and Fourier domain. What is strongly localized in one domain is widely extended in the other and vice versa.

Uncertainty relation. This general law of reciprocity finds another quantitative expression in the classical *uncertainty relation* or the *bandwidth-duration product*. This theorem relates the mean square width of a function and its Fourier transform. The mean square width $(\Delta x)^2$ is defined as

$$(\Delta x)^2 = \frac{\int\limits_{-\infty}^{\infty} x^2 \left|g(x)\right|^2 \mathrm{d}x}{\int\limits_{-\infty}^{\infty} \left|g(x)\right|^2 \mathrm{d}x} - \left(\frac{\int\limits_{-\infty}^{\infty} x \left|g(x)\right|^2 \mathrm{d}x}{\int\limits_{-\infty}^{\infty} \left|g(x)\right|^2 \mathrm{d}x} \right)^2 . \qquad (2.64)$$

It is essentially the variance of $\left|g(x)\right|^2$, a measure of the width of the distribution of the "energy" of the signal. The uncertainty relation states:

Theorem 2.7 (Uncertainty relation) *The product of the variance of* $\left|g(x)\right|^2$, $(\Delta x)^2$, *and of the variance of* $\left|\hat{g}(k)\right|^2$, $(\Delta k)^2$, *cannot be smaller than* $1/4\pi$*:*

$$\Delta x \Delta k \geq 1/(4\pi). \qquad (2.65)$$

The relations between compactness and smoothness and the uncertainty relation give some basic guidance for the design of linear filter (convolution) operators.

2.3.5 Phase and Amplitude

As outlined above, the DFT can be regarded as a coordinate transformation in a finite-dimensional vector space. Therefore, the image information is completely conserved. The inverse transform results in the original image again.

In Fourier space, we observe the image from another "point of view". Each point in the Fourier domain contains two pieces of information: the *amplitude* and the *phase*, i. e., relative position, of a periodic structure.

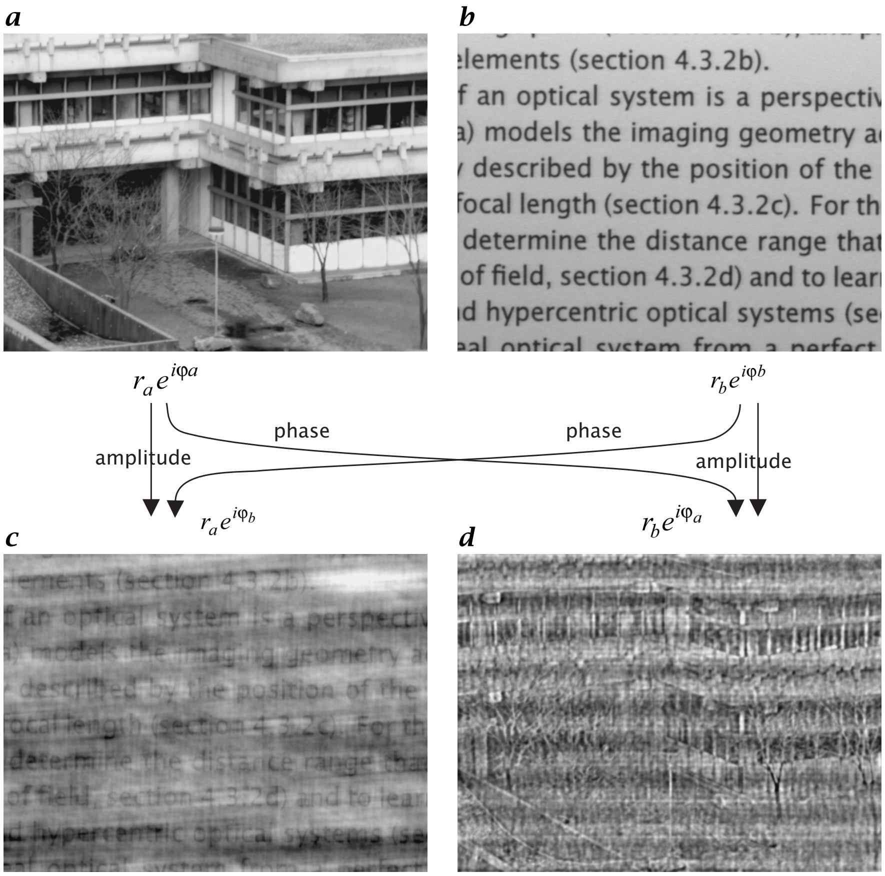

Figure 2.16: *Illustration of the importance of phase and amplitude in Fourier space for the image content:* ***a, b*** *two original images;* ***c*** *composite image using the phase from image* ***b*** *and the amplitude from image* ***a;*** ***d*** *composite image using the phase from image* ***a*** *and the amplitude from image* ***b.***

Given this composition, we ask whether the phase or the amplitude contains the more significant information on the structure in the image, or whether both are of equal importance.

In order to answer this question, we perform a simple experiment. Figure 2.16a, b shows two images. One shows Heidelberg University buildings, the other several lines of printed text. Both images are Fourier transformed and then the phase and amplitude are interchanged as illustrated in Fig. 2.16c, d. The result of this interchange is surprising. It is the phase that determines the content of an image for both images. Both images look patchy but the significant information is preserved.

From this experiment, we can conclude that the phase of the Fourier transform carries essential information about the image structure. The amplitude alone implies only *that* such a periodic structure is contained in the image but not *where*. We can also illustrate this important fact with the *shift theorem* (Theorem 2.3, p. 54 and $\succ$ R7). A shift of an object in the space domain leads to a shift of the phase in the wave number domain only. The amplitude is not changed. If we do not know the phase of its Fourier components, we know neither what the object looks like nor where it is located.

It becomes obvious also that the *power spectrum*, i. e., the squared amplitude of the Fourier components (see also Section 3.5.3), contains only very little information, since all the phase information is lost. If a gray value can be associated with the amplitude of a physical process, say a harmonic oscillation, then the power spectrum gives the distribution of the energy in the wave number domain.

2.3.6 *Alternative Definitions*

In the literature several variations of the Fourier transform exist, which can lead to a lot of confusions and errors. This has to do with the definition of the wave number. The definition of the wave number as a reciprocal wavelength $k = 1/\lambda$ is the most useful for signal processing, because k directly gives the number of wavelengths per unit length. In physics and electrical engineering, however, a definition including the factor 2π is more common: $\breve{k} = 2\pi/\lambda$. With this notation, two forms of the Fourier transform can be defined: the asymmetric form

$$\hat{g}(\breve{k}) = \left\langle \exp(\mathrm{i}\breve{k}x) \,\middle|\, g(x) \right\rangle, \; g(x) = \frac{1}{2\pi} \left\langle \exp(-\mathrm{i}\breve{k}x) \,\middle|\, \hat{g}(\breve{k}) \right\rangle \tag{2.66}$$

and the symmetric form

$$\hat{g}(\breve{k}) = \frac{1}{\sqrt{2\pi}} \left\langle \exp(\mathrm{i}\breve{k}x) \,\middle|\, g(x) \right\rangle, \; g(x) = \frac{1}{\sqrt{2\pi}} \left\langle \exp(-\mathrm{i}\breve{k}x) \,\middle|\, \hat{g}(\breve{k}) \right\rangle. \tag{2.67}$$

Because all three versions of the Fourier transform are in common use, it is likely that wrong factors in Fourier transform pairs will be obtained. The rules for conversion of Fourier transform pairs between the three versions can directly be inferred from the definitions and are summarized here:

$$\begin{array}{lll} k = 1/\lambda, \text{ Eq. (2.22)} & g(x) \;\circ\!\!-\!\!\bullet & \hat{g}(k) \\ \breve{k} = 2\pi/\lambda, \text{ Eq. (2.66)} & g(x) \;\circ\!\!-\!\!\bullet & \hat{g}(\breve{k}/2\pi) \\ \breve{k} = 2\pi/\lambda, \text{ Eq. (2.67)} & g(x) \;\circ\!\!-\!\!\bullet & \hat{g}(\breve{k}/\sqrt{2\pi})/\sqrt{2\pi}. \end{array} \tag{2.68}$$

2.3.7 *Practical Application of the DFT*

Units. For a practical application of the DFT, it is important to consider the various factors that can be used in the definition of the DFT and to give them a clear meaning. Besides the definition in Eq. (2.29) two others are commonly used:

$$
\begin{array}{llcl}
(a) & \hat{g}_v = \dfrac{1}{\sqrt{N}} \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{-nv} g_n & \bullet\!\!-\!\!\circ & g_n = \dfrac{1}{\sqrt{N}} \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{nv} \hat{g}_v, \\
(b) & \hat{g}_v = \dfrac{1}{N} \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{-nv} g_n & \bullet\!\!-\!\!\circ & g_n = \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{nv} \hat{g}_v, \\
(c) & \hat{g}_v = \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{-nv} g_n & \bullet\!\!-\!\!\circ & g_n = \dfrac{1}{N} \sum\limits_{n=0}^{N-1} \mathrm{w}_N^{nv} \hat{g}_v.
\end{array}
\tag{2.69}
$$

Mathematically spoken, the symmetric definition (a) is the most elegant because it uses in both directions the scalar product with the orthonormal base vectors in Eqs. (2.28) and (2.29). In practice, definition (b) is used most often, because $\hat{g}_0$ gives the mean value of the vector in the spatial domain, independent of its length:

$$
\hat{g}_0 = \frac{1}{N} \sum_{n=0}^{N-1} \mathrm{w}_N^{-n0} g_n = \frac{1}{N} \sum_{n=0}^{N-1} g_n. \tag{2.70}
$$

Therefore we will use definition (b) almost everywhere in this book.

In practice it is important to know which spatial or temporal intervals have been used to sample the discrete signals. Only then is it possible to compare DFTs correctly that have been sampled with different intervals. The relation can be seen most easily if we approximate the Fourier integral in Eq. (2.18) by a sum and sample the values in the spatial and temporal domain using $x = n\Delta x$, $k = v\Delta k$ und $\Delta x \Delta k = 1/N$:

$$
\begin{aligned}
\hat{g}(v\Delta k) &= \int_{-\infty}^{\infty} g(x) \exp\left(-2\pi \mathrm{i} v \Delta k x\right) \mathrm{d}x \\
&\approx \sum_{n=0}^{N-1} g_n \exp\left(-2\pi \mathrm{i} n v \Delta x \Delta k\right) \Delta x \\
&= N\Delta x \frac{1}{N} \sum_{n=0}^{N-1} g_n \exp\left(-\frac{2\pi \mathrm{i} n v}{N}\right) = N\Delta x \hat{g}_v.
\end{aligned}
\tag{2.71}
$$

These equations state that the Fourier transform g_v computed with the DFT must be multiplied by the factor $N\Delta x = 1/\Delta k$ in order to relate it to a unit interval of the wave number. Without this scaling, the Fourier transform is related to the interval $\Delta k = 1/(N\Delta x)$ and thus differs for signals sampled with different rates.

For 2-D and higher-dimensional signals corresponding relations are valid:

$$
\hat{g}(v\Delta k_1, u\Delta k_2) \approx N\Delta x M \Delta y \hat{g}_{uv} = \frac{1}{\Delta k_1 \Delta k_2} \hat{g}_{uv}. \tag{2.72}
$$

The same scaling must be applied to the squared signals (*energy*) and not the squared factors from Eq. (2.71). This result follows from the *Rayleigh theorem*

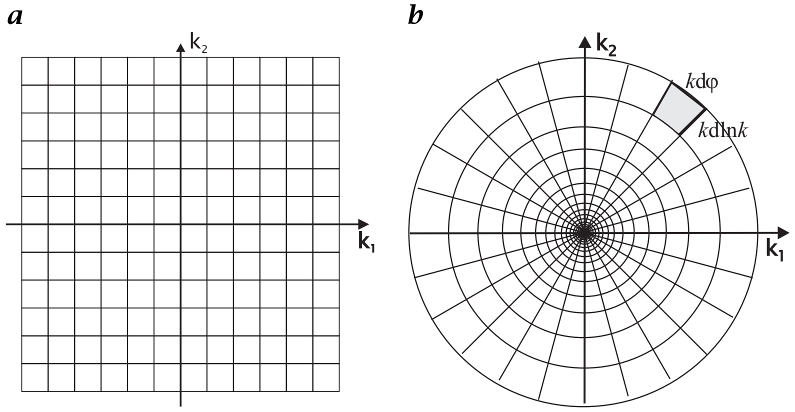

Figure 2.17: *Partition of the Fourier domain into **a** Cartesian and **b** logarithmic-polar intervals.*

for continuous and discrete signals (≻ R4, ≻ R7):

$$\begin{aligned}&\text{Continuous:} && \int_{-\infty}^{\infty} |g(x)|^2 \, dx = \int_{-\infty}^{\infty} |\hat{g}(k)|^2 \, dk, \approx \sum_{v=0}^{N-1} |\hat{g}(v\Delta k)|^2 \, \Delta k \\ &\text{Discrete:} && \frac{1}{N}\sum_{n=0}^{N-1} |g_n|^2 = \sum_{v=0}^{N-1} |\hat{g}_v|^2 .\end{aligned} \tag{2.73}$$

The Rayleigh theorem says that the signal energy can either be integrated in the spatial or the Fourier domain. For discrete Signals this means that the average energy is either given by averaging the squared signal in the spatial domain or by summing up the squared magnitude of the signal in the Fourier domain (if we use definition (b) of the DFT in Eq. (2.69)). From the approximation of the integral over the squared magnitude in the Fourier domain by a sum in Eq. (2.73), we can conclude that $|\hat{g}(v\Delta k)|^2 \approx |\hat{g}_v|^2 / \Delta k$. The units of the thus scaled squared magnitudes in the Fourier space are $\cdot / m^{-1}$ or $\cdot / Hz$ for *time series*, where $\cdot$ stands for the units of the squared signal.

Dynamic Range. While in most cases it is sufficient to represent an image with 256 quantization levels, i. e., one byte per pixel, the Fourier transform of an image needs a much larger dynamical range. Typically, we observe a strong decrease of the Fourier components with the magnitude of the wave number (Fig. 2.15). Consequently, at least 16-bit integers or 32-bit floating-point numbers are necessary to represent an image in the Fourier domain without significant rounding errors.

The reason for this behavior is not the insignificance of high wave numbers in images. If we simply omit them, we blur the image. The decrease is caused by the fact that the *relative* resolution is increasing. It is natural to think of relative resolutions, because we are better able to distinguish relative distance differences than absolute ones. We can, for example, easily see the difference of 10 cm in 1 m, but not in 1 km. If we apply this concept to the Fourier domain, it

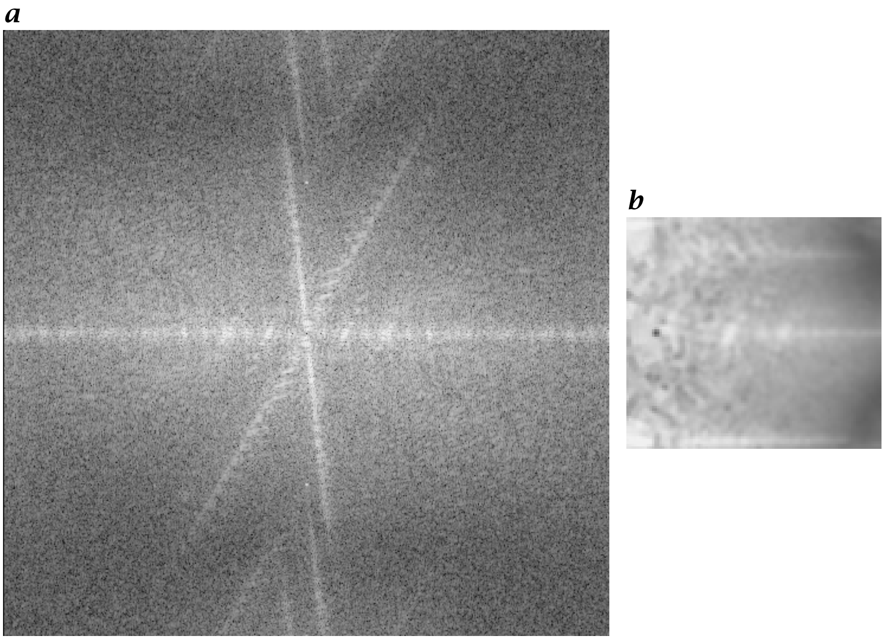

Figure 2.18: *Representation of the Fourier transformed image in Fig. 2.7 in* ***a*** *Cartesian and* ***b*** *logarithmic polar coordinates. Shown is the power spectrum* $|\hat{G}_{uv}|^2$*) multiplied by* k^2*. The gray scale is logarithmic and covers 6 decades (see also Fig. 2.15).*

seems to be more natural to represent the images in a so-called *log-polar coordinate system* as illustrated in Fig. 2.17. A discrete grid in this coordinate system separates the space into angular and $\ln k$ intervals. Thus the cell area is proportional to k^2. To account for this increase of the area, the Fourier components need to be multiplied by k^2 in this representation:

$$\int_{-\infty}^{\infty} |\hat{g}(\boldsymbol{k})|^2 \mathrm{d}k_1 \mathrm{d}k_2 = \int_{-\infty}^{\infty} k^2 |\hat{g}(\boldsymbol{k})|^2 \mathrm{d}\ln k \mathrm{d}\varphi. \tag{2.74}$$

If we assume that the power spectrum $|\hat{g}(\boldsymbol{k})|^2$ is flat in the natural log-polar coordinate system, it will decrease with k^{-2} in Cartesian coordinates.

For a display of power spectra, it is common to take the logarithm of the gray values in order to compress the high dynamic range. The discussion of log-polar coordinate systems suggests that multiplication by k^2 is a valuable alternative. Likewise, representation in a log-polar coordinate system allows a much better evaluation of the directions of the spatial structures and the smaller scales (Fig. 2.18).

2.4 *Discrete Unitary Transforms*

2.4.1 *General Properties*

In Sections 2.3.1 and 2.3.2, we learnt that the discrete Fourier transform can be regarded as a linear transformation in a vector space. Thus it is only an example of a large class of transformations, called *unitary transforms*. In this section, we discuss some of their general features that will be of help for a deeper insight into image processing. Furthermore, we give examples of other unitary transforms, which have gained importance in digital image processing.

Unitary transforms are defined for vector spaces over the field of complex numbers, for which an *inner product* is defined. Both the FT in Eq. (2.22) and DFT in Eq. (2.29) basically compute scalar products.

The basic theorem about unitary transform states:

Theorem 2.8 (Unitary transform) *Let V be a finite-dimensional inner product vector space. Let $\boldsymbol{U}$ be a one-to-one linear transformation of V onto itself. Then the following are equivalent:*

1. *$\boldsymbol{U}$ is unitary.*
2. *$\boldsymbol{U}$ preserves the inner product, i. e., $\langle \boldsymbol{g} | \boldsymbol{h} \rangle = \langle \boldsymbol{U}\boldsymbol{g} | \boldsymbol{U}\boldsymbol{h} \rangle$, $\forall \boldsymbol{g}, \boldsymbol{h} \in V$.*
3. *The inverse of $\boldsymbol{U}$, $\boldsymbol{U}^{-1}$, is the adjoint of $\boldsymbol{U}$: $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$.*
4. *The row vectors (and column vectors) of $\boldsymbol{U}$ form an orthonormal basis of the vector space V.*

In this theorem, the most important properties of a unitary transform are already related to each other: a unitary transform preserves the inner product. This implies that another important property, the *norm*, is also preserved:

$$\|\boldsymbol{g}\|_2 = \langle \boldsymbol{g} | \boldsymbol{g} \rangle^{1/2} = \langle \boldsymbol{U}\boldsymbol{g} | \boldsymbol{U}\boldsymbol{g} \rangle^{1/2}. \tag{2.75}$$

It is appropriate to think of the norm as the *length* or *magnitude* of the vector. Rotation in $\mathbb{R}^2$ or $\mathbb{R}^3$ is an example of a transform where the preservation of the length of the vectors is obvious (compare also the discussion of homogeneous coordinates in Section 7.7).

The product of two unitary transforms, $\boldsymbol{U}_1\boldsymbol{U}_2$, is unitary. As the identity operator $\boldsymbol{I}$ is unitary, as is the inverse of a unitary operator, the set of all unitary transforms on an inner product space is a *group* under the operation of composition. In practice, this means that we can compose/decompose complex unitary transforms from/into simpler or elementary transforms.

We will illustrate some of the properties of unitary transforms discussed with the discrete Fourier transform. First we consider the one-dimensional DFT in symmetric definition Eq. (2.69):

$$\hat{g}_v = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} g_n \mathrm{w}_M^{-nv}.$$

This equation can be regarded as a multiplication of the $N \times N$ matrix $\boldsymbol{W}_N$

$$(\boldsymbol{W}_N)_{nv} = \mathrm{w}_N^{-nv}$$

with the vector $\boldsymbol{g}$:

$$\hat{\boldsymbol{g}} = \frac{1}{\sqrt{N}} \boldsymbol{W}_N \boldsymbol{g}. \tag{2.76}$$

Explicitly, the DFT for an 8-dimensional vector is given by

$$\begin{bmatrix} \hat{g}_0 \\ \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \hat{g}_4 \\ \hat{g}_5 \\ \hat{g}_6 \\ \hat{g}_7 \end{bmatrix} = \frac{1}{\sqrt{8}} \begin{bmatrix} w_8^0 & w_8^0 & w_8^0 & w_8^0 & w_8^0 & w_8^0 & w_8^0 & w_8^0 \\ w_8^0 & w_8^7 & w_8^6 & w_8^5 & w_8^4 & w_8^3 & w_8^2 & w_8^1 \\ w_8^0 & w_8^6 & w_8^4 & w_8^2 & w_8^0 & w_8^6 & w_8^4 & w_8^2 \\ w_8^0 & w_8^5 & w_8^2 & w_8^7 & w_8^4 & w_8^1 & w_8^6 & w_8^3 \\ w_8^0 & w_8^4 & w_8^0 & w_8^4 & w_8^0 & w_8^4 & w_8^0 & w_8^4 \\ w_8^0 & w_8^3 & w_8^6 & w_8^1 & w_8^4 & w_8^7 & w_8^2 & w_8^5 \\ w_8^0 & w_8^2 & w_8^4 & w_8^6 & w_8^0 & w_8^2 & w_8^4 & w_8^6 \\ w_8^0 & w_8^1 & w_8^2 & w_8^3 & w_8^4 & w_8^5 & w_8^6 & w_8^7 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \end{bmatrix}$$

We made use of the periodicity of the kernel of the DFT Eq. (2.41) to limit the exponents of W between 0 and 7. The transformation matrix for the DFT is symmetric ($\boldsymbol{W} = \boldsymbol{W}^T$); $\boldsymbol{W}^{T*}$ is the back transformation.

For the two-dimensional DFT, we can write similar equations if we map the $M \times N$ matrix onto an MN-dimensional vector. There is a simpler way, however, if we make use of the separability of the kernel of the DFT as expressed in Eq. (2.38). Using the $M \times M$ matrix $\boldsymbol{W}_M$ and the $N \times N$ matrix $\boldsymbol{W}_N$ analogously to the one-dimensional case, we can write Eq. (2.76) as

$$\hat{g}_{uv} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{mn} (\boldsymbol{W}_M)_{mu} (\boldsymbol{W}_N)_{nv}, \tag{2.77}$$

or, in matrix notation,

$$\underbrace{\hat{\boldsymbol{G}}}_{M\times N} = \frac{1}{\sqrt{MN}} \underbrace{\boldsymbol{W}_M{}^T}_{M\times M} \underbrace{\boldsymbol{G}}_{M\times N} \underbrace{\boldsymbol{W}_N}_{N\times N} = \frac{1}{\sqrt{MN}} \boldsymbol{W}_M \boldsymbol{G} \boldsymbol{W}_N. \tag{2.78}$$

Physicists will be reminded of the theoretical foundations of *quantum mechanics* which are formulated in an inner product vector space of infinite dimension, the *Hilbert space*. In digital image processing, the difficulties associated with infinite-dimensional vector spaces can be avoided.

After discussing the general features, some illustrative examples of unitary transforms will be given. However, none of these transforms is as important as the Fourier transform in signal and image processing.

2.4.2 Cosine, Sine, and Hartley Transforms

It is often inconvenient that the discrete Fourier transform maps real-valued to complex-valued images. We can derive a real transformation if we decompose the complex DFT into its real and imaginary parts:

$$(\boldsymbol{W}_N)_{nv} = \cos\left(-\frac{2\pi nv}{N}\right) + \mathrm{i}\sin\left(-\frac{2\pi nv}{N}\right). \tag{2.79}$$

Neither the cosine nor the sine part is useful as a transformation kernel, since these functions do not form a complete basis of the vector space. The cosine

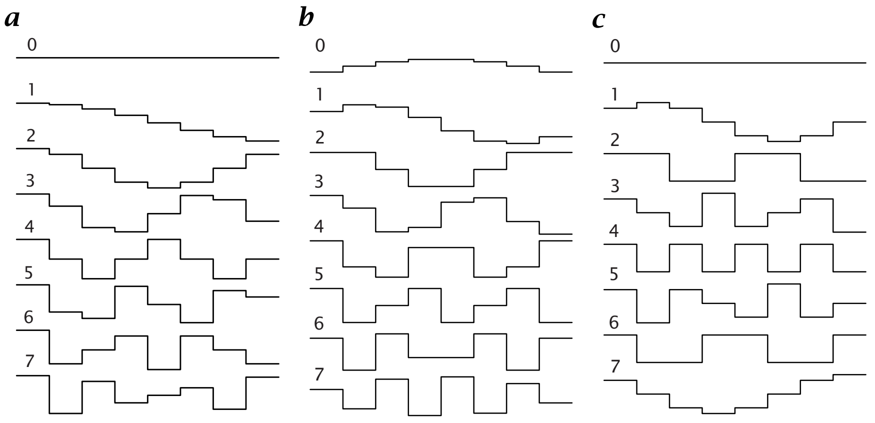

Figure 2.19: *Base functions of one-dimensional unitary transforms for* $N = 8$*:* ***a*** *cosine transform,* ***b*** *sine transform, and* ***c*** *Hartley transform.*

and sine functions only span the subspaces of the even and odd functions, respectively.

This problem can be solved by limiting the *cosine transform* and the *sine transform* to the positive half space in the spatial and Fourier domains. Then symmetry properties play no role and the two transforms are defined as

$$\begin{aligned}
{}^c\hat{g}(k) &= \int_0^\infty g(x)\sqrt{2}\cos(2\pi kx)\mathrm{d}x \quad \bullet\!\!-\!\!\circ \quad g(x) = \int_0^\infty {}^c\hat{g}(k)\sqrt{2}\cos(2\pi kx)\mathrm{d}k \\
{}^s\hat{g}(k) &= \int_0^\infty g(x)\sqrt{2}\sin(2\pi kx)\mathrm{d}x \quad \bullet\!\!-\!\!\circ \quad g(x) = \int_0^\infty {}^s\hat{g}(k)\sqrt{2}\sin(2\pi kx)\mathrm{d}k.
\end{aligned} \tag{2.80}$$

For the corresponding discrete transforms, adding trigonometric functions with half-integer wavelengths can generate base vectors with the missing symmetry. This is equivalent to doubling the base wavelength. Consequently, the kernels for the *cosine* and *sine transforms* in an N-dimensional vector space are

$$c_{nv} = \sqrt{\frac{2}{N}}\cos\left(\frac{\pi n v}{N}\right), \quad s_{nv} = \sqrt{\frac{2}{N+1}}\sin\left(\frac{\pi(n+1)(v+1)}{N+1}\right). \tag{2.81}$$

Figure 2.19a, b shows the base functions of the 1-D cosine and sine functions. From the graphs, it is easy to imagine that all the base functions are orthogonal to each other. Because of the doubling of the periods, both transforms now contain even and odd functions. The base functions with half-integer wavelengths fill in the functions with the originally missing symmetry.

The cosine transform has gained importance for *image data compression* [97]. It is included in the standard compression algorithm proposed by the Joint Photographic Experts Group (*JPEG*).

The *Hartley transform* (*HT*) is a much more elegant solution than the cosine and sine transforms for a transform that avoids complex numbers. By adding the cosine and sine function, we get an asymmetric kernel

$$\operatorname{cas} 2\pi kx = \cos(2\pi kx) + \sin(2\pi kx) = \sqrt{2}\cos(2\pi(kx - 1/8)) \tag{2.82}$$

that is suitable for a transform over the whole space domain:

$${}^h\hat{g}(k) = \int_{-\infty}^{\infty} g(x)\operatorname{cas}(2\pi kx)\mathrm{d}x \quad \bullet\!\!-\!\!\circ \quad g(x) = \int_{-\infty}^{\infty} {}^h\hat{g}(k)\operatorname{cas}(2\pi kx)\mathrm{d}k. \tag{2.83}$$

The corresponding *discrete Hartley transform* (*DHT*) is defined as:

$${}^h\hat{g}_v = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} g_n \operatorname{cas}(2\pi nv/N) \quad \bullet\!\!-\!\!\circ \quad g_n = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} {}^h\hat{g}_v \operatorname{cas}(2\pi nv/N). \tag{2.84}$$

The base vectors for $N = 8$ are shown in Fig. 2.19c. Despite all elegance of the Hartley transform for real-valued signals, it shows a number of disadvantages in comparison to the Fourier transform. Especially the simple shift theorem of the Fourier transform (Theorem 2.3, p. 54) is no longer valid. A shift rather causes base functions with positive and negative wave numbers to be combined with each other:

$$\begin{array}{lcl} g(x - x_0) & \circ\!\!-\!\!\bullet & {}^h\hat{g}(k)\cos(2\pi kx_0) + {}^h\hat{g}(-k)\sin(2\pi kx_0), \\ g_{n-n'} & \circ\!\!-\!\!\bullet & {}^h\hat{g}_v\cos(2\pi n'v/N) + {}^h\hat{g}_{N-v}\sin(2\pi n'v/N). \end{array} \tag{2.85}$$

Similar complications arise with the convolution theorem for the Hartley transform (≻ R8).

2.4.3 *Hadamard Transform*

The base functions of the *Hadamard transform* are orthogonal binary patterns (Fig. 2.20a). Some of these patterns are regular rectangular waves, others are not. The Hadamard transform is computationally efficient, because its kernel contains only the figures 1 and -1. Thus only additions and subtractions are necessary to compute the transform.

2.4.4 *Haar Transform*

The base vectors of all the transforms considered so far are characterized by the fact that the base functions spread out over the whole vector or image. In this sense we denote these transforms as *global*. All locality is lost. If we have, for example, two independent objects in our image, then they will be simultaneously decomposed into these global patterns and will no longer be recognizable as two individual objects in the transform.

The *Haar transform* is an example of a unitary transform which partly preserves local information, since its base functions are pairs of impulses which are nonzero only at the position of the impulse (Fig. 2.20a). With the Haar transform the position resolution is better for smaller structures. As with the Hadamard transform, the Haar transform is computationally efficient. Its kernel only includes the figures -1, 0, and 1.

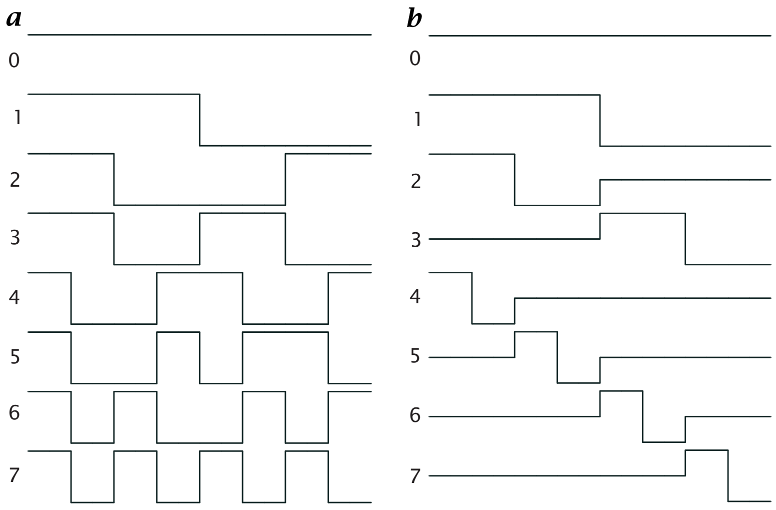

Figure 2.20: *First 8 base functions of one-dimensional unitary transforms for* $N = 16$*:* ***a*** *Hadamard transform and* ***b*** *Haar transform.*

2.5 *Fast Algorithms for Unitary Transforms*

2.5.1 *Importance of Fast Algorithms*

Without an effective algorithm to calculate the discrete Fourier transform, it would not be possible to use the Fourier transform in image processing. Applied directly, Eq. (2.38) is prohibitively expensive. Each point in the transformed image requires N^2 complex multiplications and $N^2 - 1$ complex additions (not counting the calculation of the cosine and sine functions in the kernel). In total, we need N^4 complex multiplications and $N^2(N^2 - 1)$ complex additions. This adds up to about $8N^4$ floating point operations. For a 512×512 image, this results in 5×10^{11} operations. A 2-GHz PentiumIV processor on a PC delivers about 500 MFLOPs (million floating point operations per second) if programmed in a high-level language with an optimizing compiler. A single DFT of a 512×512 image with 5×10^{11} operations would require about 1,000 s or 0.3 h, much too slow to be of any relevance for practical applications. Thus, the urgent need arises to minimize the number of computations by finding a suitable algorithm. This is an important topic in computer science. To find one we must study the inner structure of the given task, its *computational complexity*, and try to find out how it may be solved with the minimum number of operations.

As an intuitive example, consider the following simple *search* problem. A friend lives in a high-rise building with N floors. We want to find out on which floor his apartment is located. Our questions will only be answered with yes or no. How many questions must we pose to find out where he lives? The simplest and most straightforward approach is to ask “Do you live on floor n?”. In the best case, our initial guess is right, but it is more likely to be wrong so that the same question has to be asked with other floor numbers again and again. In the worst case, we must ask exactly $N - 1$ questions, in the mean $N/2$ questions. With

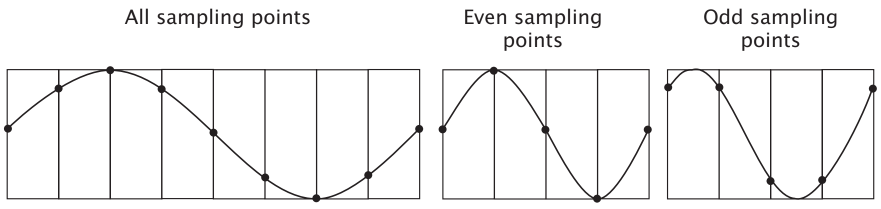

Figure 2.21: *Decomposition of a vector into two vectors containing the even and odd sampling points.*

each question, we can only rule out one out of N possibilities, a quite inefficient approach.

With the question "Do you live in the top half of the building?", however, we can rule out half of the possibilities with just one question. After the answer, we know that he either lives in the top or bottom half, and can continue our questioning in the same manner by splitting up the remaining possibilities into two halves. With this strategy, we need far fewer questions. If the number of floors is a power of two, say 2^l, we need exactly l questions. Thus for N floors, we need $\operatorname{ld} N$ questions, where ld denotes the logarithm to the base of two. The strategy applied recursively here for a more efficient solution to the search problem is called *divide and conquer*.

One measure of the computational complexity of a problem with N components is the largest power of N that occurs in the count of operations necessary to solve it. This approximation is useful, since the largest power in N dominates the number of operations necessary for large N. We speak of a zero-order problem $O(N^0)$ if the number of operations does not depend on its size and a linear-order problem $O(N)$ if the number of computations increases linearly with the size. Likewise for solutions. The straightforward solution of the search problem discussed in the previous example is a solution of order N, $O(N)$, the divide-and-conquer strategy is one of $O(\operatorname{ld} N)$.

2.5.2 *The 1-D Radix-2 FFT Algorithms*

First we consider fast algorithms for the one-dimensional DFT, commonly abbreviated as *FFT* algorithms for *fast Fourier transform*. We assume that the dimension of the vector is a power of two, $N = 2^l$. As the direct solution according to Eq. (2.29) is $O(N^2)$ it seems useful to use the divide-and-conquer strategy. If we can split the transformation into two parts with vectors the size of $N/2$, we reduce the number of operations from N^2 to $2(N/2)^2 = N^2/2$. This procedure can be applied recursively $\operatorname{ld} N$ times, until we obtain a vector of size 1, whose DFT is trivial because nothing at all has to be done. Of course, this procedure only works if the partitioning is possible and the number of additional operations to put the split transforms together is not of a higher order than $O(N)$.

The result of the recursive partitioning is interesting. We do not have to perform a DFT at all. The whole algorithm to compute the DFT has been shifted over to the recursive composition stages. If these compositions are of the order $O(N)$, the computation of the DFT totals to $O(N \operatorname{ld} N)$ since $\operatorname{ld} N$ compositions have to be performed. In comparison to the direct solution of the order $O(N^2)$, this is a tremendous saving in the number of operations. For $N = 2^{10} = 1024$, the number is reduced by a factor of about 100.

We part the vector into two vectors by choosing the even and odd elements separately (Fig. 2.21):

$$\begin{aligned}
\hat{g}_v &= \sum_{n=0}^{N-1} g_n \exp\left(-\frac{2\pi\mathrm{i}nv}{N}\right) \\
&= \sum_{n=0}^{N/2-1} g_{2n} \exp\left(-\frac{2\pi\mathrm{i}2nv}{N}\right) + \sum_{n=0}^{N/2-1} g_{2n+1} \exp\left(-\frac{2\pi\mathrm{i}(2n+1)v}{N}\right) \\
&= \sum_{n=0}^{N/2-1} g_{2n} \exp\left(-\frac{2\pi\mathrm{i}nv}{N/2}\right) + \exp\left(-\frac{2\pi\mathrm{i}v}{N}\right) \sum_{n=0}^{N/2-1} g_{2n+1} \exp\left(-\frac{2\pi\mathrm{i}nv}{N/2}\right).
\end{aligned} \tag{2.86}$$

Both sums constitute a DFT with $N' = N/2$. The second sum is multiplied by a phase factor which depends only on the wave number v. This phase factor results from the shift theorem, since the odd elements are shifted one place to the left.

As an example, we take the base vector with $v = 1$ and $N = 8$ (Fig. 2.21). Taking the odd sampling points, the function shows a phase shift of $\pi/4$. This phase shift is exactly compensated by the phase factor in Eq. (2.86):

$$\exp(-2\pi\mathrm{i}v/N) = \exp(-\pi\mathrm{i}/4).$$

The operations necessary to combine the partial Fourier transforms are just one complex multiplication and addition, i. e., $O(N^1)$. Some more detailed considerations are necessary, however, since the DFT over the half-sized vectors only yields $N/2$ values. In order to see how the composition of the N values works, we study the values for v from 0 to $N/2 - 1$ and $N/2$ to $N - 1$ separately. The partial transformations over the even and odd sampling points are abbreviated by ${}^e\hat{g}_v$ and ${}^o\hat{g}_v$, respectively. For the first part, we can just take the partitioning as expressed in Eq. (2.86). For the second part, $v' = v + N/2$, only the phase factor changes. The addition of $N/2$ results in a change of sign:

$$\exp\left(-\frac{2\pi\mathrm{i}(v + N/2)}{N}\right) = -\exp\left(-\frac{2\pi\mathrm{i}v}{N}\right) \quad \text{or} \quad \mathrm{w}_N^{-(v+N/2)} = -\mathrm{w}_N^{-v}.$$

Making use of this symmetry we can write

$$\left.\begin{aligned}
\hat{g}_v &= {}^e\hat{g}_v + \mathrm{w}_N^{-v}\, {}^o\hat{g}_v \\
\hat{g}_{v+N/2} &= {}^e\hat{g}_v - \mathrm{w}_N^{-v}\, {}^o\hat{g}_v.
\end{aligned}\right\} \quad 0 \le v < N/2. \tag{2.87}$$

The Fourier transforms for the indices v and $v + N/2$ only differ by the sign of the second term. Thus for the composition of *two* terms we only need *one* complex multiplication. The partitioning is now applied recursively. The two transformations of the $N/2$-dimensional vectors are parted again into two transformations each. We obtain similar expressions as in Eq. (2.86) with the only difference being that the phase factor has doubled to $\exp[-(2\pi\mathrm{i}v)/(N/2)]$. The

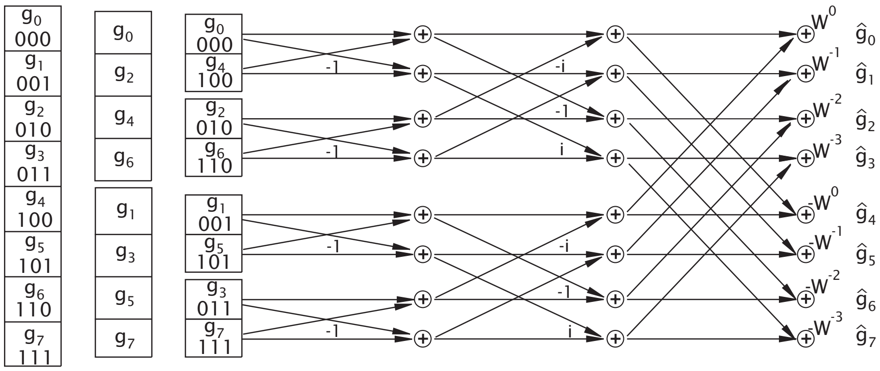

Figure 2.22: *Signal flow diagram of the radix-2 decimation-in-time Fourier transform algorithm for $N = 8$; for further explanation, see text.*

even and odd parts of the even vector contain the points $\{0, 4, 8, \cdots, N/2 - 4\}$ and $\{2, 6, 10, \cdots, N/2 - 2\}$, respectively.

In the last step, we decompose a vector with two elements into two vectors with one element. As the DFT of a single-element vector is an identical operation Eq. (2.29), no further calculations are necessary.

After the decomposition is complete, we can use Eq. (2.87) recursively with appropriate phase factors to compose the original vector step by step in the inverse order. In the first step, we compose vectors with just two elements. Thus we only need the phase factor for $v = 0$ which is equal to one. Consequently, the first composition step has a very simple form:

$$\begin{aligned} \hat{g}_0 &= g_0 + g_1 \\ \hat{g}_{0+N/2} = \hat{g}_1 &= g_0 - g_1. \end{aligned} \qquad (2.88)$$

The algorithm we have discussed is called a *decimation-in-time* FFT algorithm, as the signal is decimated in the space domain. All steps of the FFT algorithm are shown in the signal flow diagram in Fig. 2.22 for $N = 8$. The left half of the diagram shows the decimation steps. The first column contains the original vector, the second the result of the first decomposition step into two vectors. The vectors with the even and odd elements are put in the lower and upper halves, respectively. This decomposition is continued until we obtain vectors with one element.

As a result of the decomposition, the elements of the vectors are arranged in a new order. That is all that is performed in the decomposition steps. No computations are required. We can easily understand the new ordering scheme if we represent the indices of the vector with dual numbers. In the first decomposition step we order the elements according to the least significant bit, first the even elements (least significant bit is zero), then the odd elements (least significant bit is one). With each further decomposition step, the bit that governs the sorting is shifted one place to the left. In the end, we obtain a sorting in which the ordering of the bits is completely reversed. The element with the index $1 = 001_2$, for example, will be at the position $4 = 100_2$, and vice versa.

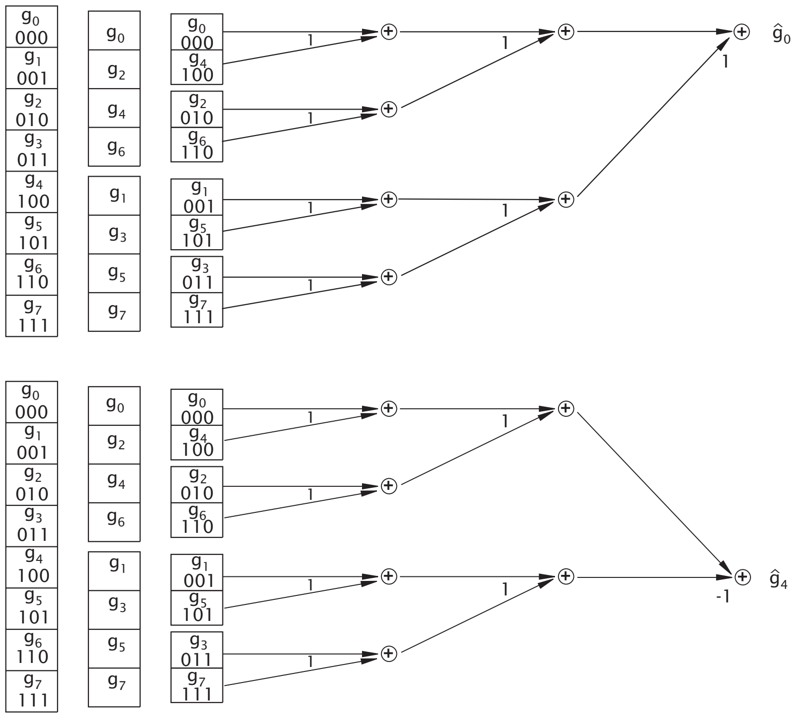

Figure 2.23: *Signal flow path for the calculation of $\hat{g}_0$ and $\hat{g}_4$ with the decimation-in-time FFT algorithm for an 8-dimensional vector.*

Consequently, the chain of decomposition steps can be performed with one operation by interchanging the elements at the normal and bit-reversed positions. This reordering is known as *bit reversal*.

Further steps on the right side of the signal flow diagram show the stepwise composition to vectors of double the size. The composition to the 2-dimensional vectors is given by Eq. (2.88). The operations are pictured with arrows and points having the following meaning: points represent a figure, an element of the vector. These points are called the *nodes* of the signal flow graph. The arrows transfer the figure from one point to another. During the transfer the figure is multiplied by the factor written close to the arrow. If the associated factor is missing, no multiplication takes place. A value of a knot is the sum of the values transferred from the previous level.

The elementary operation of the FFT algorithm involves only two knots. The lower knot is multiplied with a phase factor. The sum and difference of the two values are then transferred to the upper and lower knot, respectively. Because of the crossover of the signal paths, this operation is denoted as a *butterfly operation*.

We gain further insight into the FFT algorithm if we trace back the calculation of a single element. Figure 2.23 shows the signal paths for $\hat{g}_0$ and $\hat{g}_4$. For each level we go back the number of knots contributing to the calculation doubles.

In the last stage all the elements are involved. The signal path for $\hat{g}_0$ and $\hat{g}_4$ are identical but for the last stage, thus nicely demonstrating the efficiency of the FFT algorithm.

All phase factors in the signal path for $\hat{g}_0$ are one. As expected from Eq. (2.29), $\hat{g}_0$ contains the sum of all the elements of the vector $\boldsymbol{g}$,

$$\hat{g}_0 = [(g_0 + g_4) + (g_2 + g_6)] + [(g_1 + g_5) + (g_3 + g_7)],$$

while in the last stage for $\hat{g}_4$ the addition is replaced by a subtraction

$$\hat{g}_4 = [(g_0 + g_4) + (g_2 + g_6)] - [(g_1 + g_5) + (g_3 + g_7)].$$

In Section 2.4, we learnt that the DFT is an example of a unitary transform which is generally performed by multiplying a unitary matrix with the vector. What does the FFT algorithm mean in this context? The signal flow graph in Fig. 2.22 shows that the vector is transformed in several steps. Consequently, the unitary transformation matrix is broken up into several partial transformation matrices that are applied one after the other.

If we take the algorithm for $N = 8$ as shown in Fig. 2.22, the unitary matrix is split up into three simpler transformations with sparse unitary transformations:

$$\begin{bmatrix} \hat{g}_0 \\ \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \hat{g}_4 \\ \hat{g}_5 \\ \hat{g}_6 \\ \hat{g}_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & w^{-1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & w^{-2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & w^{-3} \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -w^{-1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -w^{-2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -w^{-3} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & i & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & i \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -i \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \end{bmatrix}$$

The reader can verify that these transformation matrices reflect all the properties of a single level of the FFT algorithm. The matrix decomposition emphasizes that the FFT algorithm can also be considered as a clever method to decompose the unitary transformation matrix into sparse partial unitary transforms.

2.5.3 *Measures for Fast Algorithms*

According to the number of arithmetic operations required, there are many other fast Fourier transform algorithms which are still more effective. Most of them are based on polynomial algebra and number theory. An in-depth discussion of these algorithms is given by Blahut [11]. However, the mere number of arithmetic operations is not the only measure for an efficient algorithm. We must also consider a number of other factors.

Access to the data requires additional operations. Consider the simple example of the addition of two vectors. There, besides addition, the following operations are performed: the addresses of the appropriate elements must be calculated; the two elements are read into registers, and the result of these additions is written back to the memory. Depending on the architecture of the hardware used, these five operations constitute a significant overhead which may take much more time than the addition itself. Consequently, an algorithm with a complicated scheme to access the elements of a vector might add a considerable overhead to the arithmetic operations. In effect, a simpler algorithm with more arithmetic operations but less overhead to compute addresses may be faster.

Another factor for rating algorithms is the amount of storage space needed. This not only includes the space for the code but also storage space required for intermediate results or tables for constants. For example, a so-called in-place FFT algorithm, which can perform the Fourier transform on an image without using an intermediate storage area for the image, is very advantageous. Often there is a trade-off between storage space and speed. Many integer FFT algorithms, for example, precalculate the complex phase factors w_N^v and store them in statically allocated tables.

To a large extent the efficiency of algorithms depends on the computer architecture used to implement them. If multiplication is performed either in software or by a microcoded instruction, it is much slower than addition or memory access. In this case, the aim of fast algorithms is to reduce the number of multiplications even at the cost of more additions or a more complex memory access. Such a strategy makes no sense on some modern high-speed architectures where pipelined floating-point addition and multiplication take just one clock cycle. The faster the operations on the processor, the more the memory access becomes the bottleneck. Fast algorithms must now consider effective memory access schemes. It is crucial that as many computations as possible can be performed with one and the same set of data. In this way, these data can be kept in a fast intermediate storage area, known as the *memory cache*, and no direct access to the much slower general memory (RAM) is required.

After this detailed discussion of the algorithm, we can now estimate the number of operations necessary. At each stage of the composition, $N/2$ complex multiplications and N complex additions are carried out. In total we need $N/2\,\mathrm{ld}N$ complex multiplications and $N\,\mathrm{ld}N$ complex additions. A deeper analysis shows that we can save even more multiplications. In the first two composition steps only trivial multiplications by 1 or i occur (compare Fig. 2.22). For further steps the number of trivial multiplications decreases by a factor of two. If our algorithm could avoid all the trivial multiplications, the number of multiplications would be reduced to $(N/2)(\mathrm{ld}\,N - 3)$.

The FFT algorithm is a classic example of a *fast algorithm*. The computational savings are enormous. For a 512-element vector, only 1536 instead of 262 144 complex multiplications are needed compared to the direct calculation according to Eq. (2.29). The number of multiplications has been reduced by a factor 170. Using the FFT algorithm, the discrete Fourier transform can no longer be regarded as a computationally expensive operation, since only a few operations are necessary per element of the vector. For a vector with 512 elements, only 3 complex multiplications and 8 complex additions, corresponding to 12 real multiplications and 24 real additions, need to be computed per pixel.

2.5.4 *Radix-4 Decimation-in-Time FFT*

Having worked out one fast algorithm, we still do not know whether the algorithm is optimal or if even more efficient algorithms can be found. Actually, we have applied only one special case of the *divide-and-conquer* strategy. Instead of parting the vector into two pieces, we could have chosen any other partition, say P Q-dimensional vectors, if $N = PQ$. This type of algorithms is called a *Cooley-Tukey algorithm* [11].

Another partition often used is the *radix-4 FFT algorithm*. We can decompose a vector into four components:

$$\begin{aligned}\hat{g}_v &= \sum_{n=0}^{N/4-1} g_{4n} \mathrm{w}_N^{-4nv} + \mathrm{w}_N^{-v} \sum_{n=0}^{N/4-1} g_{4n+1} \mathrm{w}_N^{-4nv} \\ &+ \mathrm{w}_N^{-2v} \sum_{n=0}^{N/4-1} g_{4n+2} \mathrm{w}_N^{-4nv} + \mathrm{w}_N^{-3v} \sum_{n=0}^{N/4-1} g_{4n+3} \mathrm{w}_N^{-4nv}.\end{aligned}$$

For simpler equations, we will use similar abbreviations as for the radix-2 algorithm and denote the partial transformations by ${}^0\hat{g}, \cdots, {}^3\hat{g}$. Making use of the symmetry of w_N^v, the transformations into quarters of each of the vectors are given by

$$\begin{aligned}\hat{g}_v &= {}^0\hat{g}_v + \mathrm{w}_N^{-v}\,{}^1\hat{g}_v + \mathrm{w}_N^{-2v}\,{}^2\hat{g}_v + \mathrm{w}_N^{-3v}\,{}^3\hat{g}_v \\ \hat{g}_{v+N/4} &= {}^0\hat{g}_v - \mathrm{i}\mathrm{w}_N^{-v}\,{}^1\hat{g}_v - \mathrm{w}_N^{-2v}\,{}^2\hat{g}_u + \mathrm{i}\mathrm{w}_N^{-3v}\,{}^3\hat{g}_v \\ \hat{g}_{v+N/2} &= {}^0\hat{g}_v - \mathrm{w}_N^{-v}\,{}^1\hat{g}_v + \mathrm{w}_N^{-2v}\,{}^2\hat{g}_v - \mathrm{w}_N^{-3v}\,{}^3\hat{g}_v \\ \hat{g}_{v+3N/4} &= {}^0\hat{g}_v + \mathrm{i}\mathrm{w}_N^{-v}\,{}^1\hat{g}_v - \mathrm{w}_N^{-2v}\,{}^2\hat{g}_v - \mathrm{i}\mathrm{w}_N^{-3v}\,{}^3\hat{g}_v\end{aligned}$$

or, in matrix notation,

$$\begin{bmatrix} \hat{g}_v \\ \hat{g}_{v+N/4} \\ \hat{g}_{v+N/2} \\ \hat{g}_{v+3N/4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{bmatrix} \begin{bmatrix} {}^0\hat{g}_v \\ \mathrm{w}_N^{-v}\,{}^1\hat{g}_v \\ \mathrm{w}_N^{-2v}\,{}^2\hat{g}_v \\ \mathrm{w}_N^{-3v}\,{}^3\hat{g}_v \end{bmatrix}.$$

To compose 4-tuple elements of the vector, 12 complex additions and 3 complex multiplications are needed. We can reduce the number of additions further by decomposing the matrix into two simpler matrices:

$$\begin{bmatrix} \hat{g}_v \\ \hat{g}_{v+N/4} \\ \hat{g}_{v+N/2} \\ \hat{g}_{v+3N/4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -\mathrm{i} \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & \mathrm{i} \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} {}^0\hat{g}_v \\ \mathrm{w}_N^{-v}\,{}^1\hat{g}_v \\ \mathrm{w}_N^{-2v}\,{}^2\hat{g}_v \\ \mathrm{w}_N^{-3v}\,{}^3\hat{g}_v \end{bmatrix}.$$

The first matrix multiplication yields intermediate results which can be used for several operations in the second stage. In this way, we save four additions. We can apply this decomposition recursively $\log_4 N$ times. As for the radix-2 algorithm, only trivial multiplications in the first composition step are needed. At all other stages, multiplications occur for 3/4 of the points. In total, $3/4N(\log_4 N - 1) = 3/8N(\mathrm{ld}N - 2)$ complex multiplications and $2N \log_4 N = N\mathrm{ld}N$ complex additions are necessary for the radix-4 algorithm. While the number of additions remains equal, 25 % fewer multiplications are required than for the radix-2 algorithm.

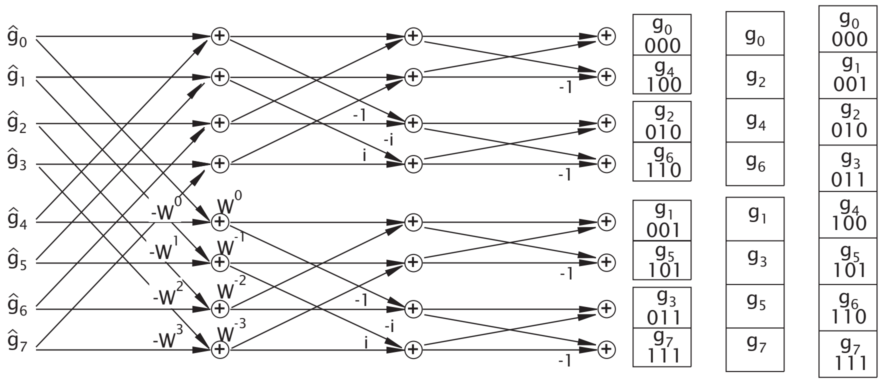

Figure 2.24: *Signal flow diagram of the radix-2 decimation-in-frequency FFT algorithm for $N = 8$.*

2.5.5 *Radix-2 Decimation-in-Frequency FFT*

The *decimation-in-frequency FFT* is another example of a Cooley-Tukey algorithm. This time, we break the N-dimensional input vector into $N/2$ first and $N/2$ second components. This partition breaks the output vector into its even and odd components:

$$\begin{aligned}\hat{g}_{2v} &= \sum_{n=0}^{N/2-1} (g_n + g_{n+N/2}) \mathrm{w}_{N/2}^{-nv} \\ \hat{g}_{2v+1} &= \sum_{n=0}^{N/2-1} W_N^{-n} (g_n - g_{n+N/2}) \mathrm{w}_{N/2}^{-nv}.\end{aligned} \tag{2.89}$$

A recursive application of this partition results in a bit reversal of the elements in the output vector, but not the input vector. As an example, the signal flow graph for $N = 8$ is shown in Fig. 2.24. A comparison with the decimation-in-time flow graph (Fig. 2.22) shows that all steps are performed in reverse order. Even the elementary butterfly operations of the decimation-in-frequency algorithm are the inverse of the butterfly operation in the decimation-in-time algorithm.

2.5.6 *Multidimensional FFT Algorithms*

Generally, there are two possible ways to develop fast algorithms for *multidimensional discrete Fourier transforms*. Firstly, we can decompose the multidimensional DFT into 1-D DFTs and use fast algorithms for them. Secondly, we can generalize the approaches of the 1-D FFT for higher dimensions. In this section, we show examples for both possible ways.

Decomposition into 1-D Transforms. A two-dimensional DFT can be broken up in two one-dimensional DFTs because of the separability of the kernel. In the 2-D case Eq. (2.38), we obtain

$$\hat{g}_{u,v} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \left[\sum_{n=0}^{N-1} g_{m,n} \exp\left(-\frac{2\pi \mathrm{i} nv}{N}\right) \right] \exp\left(-\frac{2\pi \mathrm{i} mu}{M}\right). \tag{2.90}$$

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

Figure 2.25: *Decomposition of an image matrix into four partitions for the 2-D radix-2 FFT algorithm.*

The inner summation forms M 1-D DFTs of the rows, the outer N 1-D DFTs of the columns, i. e., the 2-D FFT is computed as M row transformations followed by N column transformations

$$\text{Row transform} \qquad \tilde{g}_{m,v} = \frac{1}{N}\sum_{n=0}^{N-1} g_{m,n} \exp\left(-\frac{2\pi \mathrm{i} n v}{N}\right)$$

$$\text{Column transform} \qquad \hat{g}_{u,v} = \frac{1}{M}\sum_{m=0}^{M-1} \tilde{g}_{m,v} \exp\left(-\frac{2\pi \mathrm{i} m u}{M}\right).$$

In an analogous way, a W-dimensional DFT can be composed of W one-dimensional DFTs.

Multidimensional Decomposition. A decomposition is also directly possible in multidimensional spaces. We will demonstrate such algorithms with the simple case of a 2-D radix-2 decimation-in-time algorithm.

We decompose an $M \times N$ matrix into four submatrices by taking only every second pixel in every second line (Fig. 2.25). This decomposition yields

$$\begin{bmatrix} \hat{g}_{u,v} \\ \hat{g}_{u,v+N/2} \\ \hat{g}_{u+M/2,v} \\ \hat{g}_{u+M/2,v+N/2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} {}^{0,0}\hat{g}_{u,v} \\ \mathrm{w}_N^{-v}\;{}^{0,1}\hat{g}_{u,v} \\ \mathrm{w}_M^{-u}\;{}^{1,0}\hat{g}_{u,v} \\ \mathrm{w}_M^{-u}\mathrm{w}_N^{-v}\;{}^{1,1}\hat{g}_{u,v} \end{bmatrix}.$$

The superscripts in front of $\hat{g}$ denote the corresponding partial transformation. The 2-D radix-2 algorithm is very similar to the 1-D radix-4 algorithm. In a similar manner as for the 1-D radix-4 algorithm (Section 2.5.4), we can reduce the number of additions from 12 to 8 by factorizing the matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

The 2-D radix-2 algorithm for an $N \times N$ matrix requires $(3/4N^2)\,\mathrm{ld}\,N$ complex multiplications, 25 % fewer than the separation into two 1-D radix-2 FFTs. However, the multidimensional decomposition has the disadvantage that the memory access pattern is more complex than for the 1-D Fourier transform. With the partition into a 1-D transform, the access to memory becomes local, yielding a higher cache hit rate than with the distributed access of the multidimensional decomposition.

2.5.7 *Fourier Transform of Real Images*

So far, we have only discussed the Fourier transform of complex-valued signals. The same algorithms can be used also for real-valued signals. Then they are less efficient, however, because the Fourier transform of a real-valued signal is Hermitian (Section 2.3.4) and thus only half of the Fourier coefficients are independent. This corresponds to the fact that also half of the signal, namely the imaginary part, is zero.

It is obvious that another factor two in computational speed can be gained for the DFT of real data. The easiest way to do so is to compute two real 1-D sequences at once. This concept can easily be applied to the DFT of images, because many 1-D DFTs must be computed. Thus we can put the first row $\boldsymbol{x}$ in the real part and the second row $\boldsymbol{y}$ in the imaginary part and yield the complex vector $\boldsymbol{z} = \boldsymbol{x} + \mathrm{i}\boldsymbol{y}$. From the symmetry properties discussed in Section 2.3.4, we infer that the transforms of the real and imaginary parts map in Fourier space to the Hermitian and anti-Hermitian parts. Thus the Fourier transforms of the two real M-dimensional vectors are given by

$$\hat{x}_v = 1/2(\hat{z}_v + \hat{z}^*_{N-v}), \quad \mathrm{i}\hat{y}_v = 1/2(\hat{z}_v - \hat{z}^*_{N-v}). \tag{2.91}$$

2.6 *Exercises*

2.1: Spatial resolution of images

Representation of images with interactively adjustable number of points (dip6ex02.01).

2.2: Quantization of images

Representation of images with interactively adjustable number of quantization levels (dip6ex02.02).

2.3: Context-dependent brightness perception

Interactive demonstration of the context-dependent brightness perception of the human visual system (dip6ex02.03).

2.4: Contrast resolution of the human visual system

Interactive experiment to determine the contrast resolution of the human visual system (dip6ex02.04).

2.5: Gamma value

Interactive adjustment of the gamma value for image display (dip6ex02.05).

2.6: *Contrast resolution with a logarithmic imaging sensor

Compute the relative brightness resolution $\Delta g'/g'$ caused by digitalization ($\Delta g' = 1$) of an image sensor with a logarithmic response of the form

$$g' = a_0 + a_1 \log g$$

and a contrast range of six decades for 8 and 10 bit resolution. The minimum gray value g is mapped to $g' = 0$ and the 110^6 times higher maximum gray value to either $g' = 255$ or $g' = 1023$.

2.7: Partitioning into periodic patterns

Interactive demonstration of the partitioning of an image into periodic patterns, i. e., the basis functions of the Fourier transform (dip6ex02.06).

2.8: Fourier transform

Interactive tutorial for the Fourier transform (dip6ex02.07).

2.9: Contrast range of Fourier transformed images

Interactive tutorial for the computation of the Fourier transform and the contrast range of Fourier transformed images (dip6ex02.08).

2.10: Phase and amplitude of the Fourier transform

Interactive tutorial for the meaning and importance of the amplitude and phase of the Fourier transform of images (dip6ex02.9).

2.11: *Shift theorem of the Fourier transform

Prove the shift theorem (Theorem 2.3, p. 54) of the Fourier transform.

2.12: **Fourier transform pairs

Compute the Fourier transform of the following functions in the spatial domain using the Fourier transform pairs listed in ≻ R5 and ≻ R6 and the basic theorems of the Fourier transform (Section 2.3.4 und ≻ R4):

a) $$\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

b) $$\frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right)$$

c) $$\cos^2(\boldsymbol{k}_0\boldsymbol{x}), \quad \sin^2(\boldsymbol{k}_0\boldsymbol{x})$$

d) $$\Lambda(x) = \begin{cases} 1 - |x| & |x| \leq 1 \\ 0 & \text{sonst} \end{cases} \quad \text{(triangle function)}$$

e) $$\cos(\boldsymbol{k}_0\boldsymbol{x}) \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}_0)^2}{2\sigma^2}\right) \quad \text{(wave packet)}$$

With some functions, different ways to compute the Fourier transform are possible. Carefully list all steps of your solution and indicate, which theorems you used.

2.13: ***DFT**

With this exercise, it is easy to get acquainted with the 1-D discrete Fourier transform.

1. Compute the basis functions of the DFT for vectors with 4 and 8 elements.
2. Compute the Fourier transform of the vector $[4\ 1\ 2\ 1]^T$
3. Compute the Fourier transform of the vector $[1\ 4\ 1\ 2]^T$ to see how the shift theorem (Theorem 2.3, p. 54) works.
4. Compute the Fourier transform of the vector $[4\ 0\ 1\ 0\ 2\ 0\ 1\ 0]^T$ to see how the discrete similar theorem (Theorem 2.2, p. 53) works.
5. Convolve the vector $[4\ 1\ 2\ 1]^T$ with $[2\ 1\ 0\ 1]^T/4$ and compute the Fourier transform of the second vector and of the convolved vectors to see how the discrete convolution theorem (Theorem 2.4, p. 54) works.

2.14: ****Derivation theorem of the DFT**

While almost all theorems of the continuous FT can easily be transferred to the discrete FT (compare ≻R4 to ≻R7), there are problems with the derivation theorem because the derivation can only be approximated by finite difference in a discrete space. Prove the theorem for the symmetric finite difference for the 1-D DFT

$$(g_{n+1} - g_{n-1})/2 \circ\!\!-\!\!\bullet \ \mathrm{i}\sin(2\pi v/N)\hat{g}_v$$

and show why this theorem is an approximation to the derivation theorem of the continuous FT.

2.15: ****Invariant Fourier transform pairs**

Which functions are invariant to the continuous Fourier transform, i. e., do not change their form except for a scaling factor? (Hint: check ≻R6 in the reference part of the book.) Do these invariant Fourier transform pairs have a special importance for signal processing?

2.16: ****Symmetries of the Fourier transform**

Prove the following symmetry relations for Fourier transform pairs:

Spatial domain	Fourier domain
Hermitian $g(-\boldsymbol{x}) = g^\star(\boldsymbol{x})$	real: $\hat{g}^\star(\boldsymbol{k}) = \hat{g}(\boldsymbol{k})$
real $g^\star(\boldsymbol{x}) = g(\boldsymbol{x})$	Hermitian: $\hat{g}(-\boldsymbol{k}) = \hat{g}^\star(\boldsymbol{k})$
real and even	real and even
real and odd	imaginary and odd
separable: $g(x_1)h(x_2)$	separable: $\hat{g}(k_1)\hat{h}(k_2)$
rotational symmetric $g(\lvert\boldsymbol{x}\rvert)$	rotational symmetric $\hat{g}(\lvert\boldsymbol{k}\rvert)$

2.17: ***Radix-3 FFT Algorithm**

Does a Radix-3 FFT algorithm have the same order $O(N \,\mathrm{ld}\, N)$ as Radix-2 and Radix -4 algorithms? Are more or less numbers of computational steps necessary?

2.18: ***FFT of real signals**

In Section 2.5.7 we discussed a method how the Fourier transform of a real image can be computed efficiently. Another method is possible. It is based on the same decomposition principle as the radix-2 FFT algorithm (Section 2.5.2, Eq. (2.86)). The real vector is partitioned into two. The even-numbered points are thought to be the real part of a complex vector. From this vector, the Fourier transform is computed. Show how the Fourier transform of the real vector can be computed from the Fourier transform of the complex vector. (This method has the significant advantage that it can also be applied for a single real vector in contrast to the method described in Section 2.5.7.)

2.7 *Further Readings*

The classical textbook on the Fourier transform — and still one of the best — is Bracewell [13]. An excellent source for various transforms is the “Handbook on Transforms” by Poularikas [156]. For the basics of linear algebra, especially unitary transforms, the reader is referred to one of the modern textbooks on linear algebra, e. g., Meyer [137], Anton [5], or Lay [118]. It is still worthwhile to read the historical article of Cooley and Tukey [25] about the discovery of the first fast Fourier transform algorithm. The monograph of Blahut [11] covers a variety of fast algorithms for the Fourier transform. Aho et al. [3] give a general coverage of the design and analysis of algorithm in a very clear and understandable way. The extensive textbook of Cormen et al. [26] can also be recommended. Both textbooks include the FFT.