9.6.4 Polynomial interpolation 318
9.6.5 Spline-based interpolation 321
9.6.6 Optimized interpolation 324
9.7 Edge detection 325
9.7.1 Edge detection by first-order derivatives 325
9.7.2 Edge detection by zero crossings 331
9.7.3 Edges in multichannel images 333
9.8 Tensor representation of simple neighborhoods 335
9.8.1 Simple neighborhoods 335
9.8.2 Direction versus orientation 337
9.8.3 First-order tensor representation; structure tensor . . 338
9.8.4 Classification of local neighborhoods 340
9.8.5 Computation of the structure tensor 341
9.8.6 Orientation vector 342
9.8.7 Coherency 343
9.8.8 Color coding of the two-dimensional structure tensor 343
9.9 References 344

9.1 Basics

The extraction of features from multidimensional signals requires the analysis of at least a local neighborhood. By analysis of the local neighborhood a rich set of features can already be extracted. We can distinguish areas of constant gray values from those that contain an edge, a texture, or just noise. Thus this chapter gives an important theoretical basis for low-level signal processing.

9.1.1 Definition of neighborhood operators

A neighborhood operator takes the gray values of the neighborhood around a point, performs some operations with them, and writes the result back on the pixel. This operation is repeated for all points of the signal. Therefore, we can write a neighborhood operation with a multidimensional continuous signal $g(\boldsymbol{x})$ as

$$g'(\boldsymbol{x}) = N(\{g(\boldsymbol{x}')\}, \forall(\boldsymbol{x} - \boldsymbol{x}') \in M) \tag{9.1}$$

where M is an area, called *mask*, *region of support*, or *structure element*. The size and shape of M determines the neighborhood operation by specifying the input values of g in the area M shifted with its origin to the point $\boldsymbol{x}$. The neighborhood operation N itself is not specified here. It can be of any type; its result determines the value of the output g' at $\boldsymbol{x}$. For symmetry reasons the mask is often symmetric and has its center of gravity in the origin.

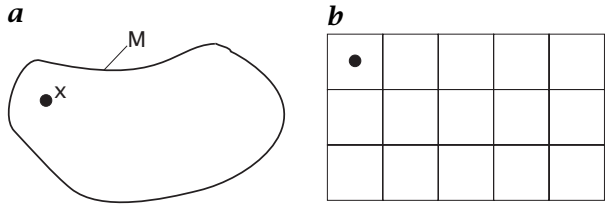

Figure 9.1: *Mask or structure element with* ***a*** *continuous; and* ***b*** *digital 2-D signals on a square lattice. The point that receives the result is marked.*

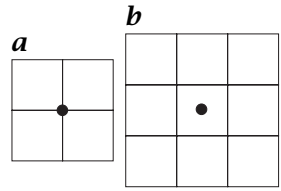

Figure 9.2: *Various types of symmetric masks on 2-D lattices:* ***a*** 2×2 *mask; and* ***b*** 3×3 *mask on a square lattice.*

For digital signals a general neighborhood operation can be expressed as

$$G'_{m,n} = N(G_{m'-m,n'-n}\}, \forall\, [m,n]^T \in M) \tag{9.2}$$

or by equivalent expressions for dimensions other than two.

Although these equations do not specify in any way the type of neighborhood operation that is performed, they still reveal the common structure of all neighborhood operations. Thus very general strategies can be developed to compute them efficiently [CVA2, Section 5.6].

9.1.2 Shape and symmetry of neighborhoods

As we have seen, any type of neighborhood operator is first determined by the size of the mask. With continuous signals, the mask may take any shape. With digital data on orthogonal lattices, the mask is normally of rectangular shape. In any case, we must also specify the point relative to the mask that receives the result of the operation (Fig. 9.1).

With regard to symmetry, the most natural choice is to place the result of the operation at the pixel in the center of the mask. While this is straightforward for continuous masks, it requires more thought for digital signals. Natural choices for masks on an orthogonal lattice are rectangles. Basically, there are two types of symmetric masks: masks with an even or odd size of pixels in each direction. For odd-sized masks, the symmetry center coincides with the central pixel and, thus, seems to be a good choice (Fig. 9.2b). The smallest size of odd-sized

masks includes only the directly neighboring pixels. In one, two, and three dimensions, the mask includes 3, 9, and 27 pixels, respectively.

In contrast, even-sized masks seem not to be suitable for neighborhood operations because there is no pixel that lies in the center of the mask. With adroitness, we can apply them nevertheless, and they turn out to be useful for certain types of neighborhood operations. The result of the neighborhood operation is simply written back to pixels that lay between the original pixels (Fig. 9.2a). Thus, the resulting image is shifted by half the pixel distance into every direction and the receiving central pixel lays directly in the center of the neighborhoods. In effect, the resulting image has one pixel less in every direction. It is very important to be aware of this shift by half the pixel distance. Therefore, image features computed by even-sized masks should never be combined with original gray values because this would lead to considerable errors. Also, a mask must either be even-sided or odd-sized in *all* directions for multidimensional digital signals. Otherwise, the output lattices do not coincide.

The number of pixels contained in the masks increases considerably with their size. If R is the linear size of a mask in D dimensions, the mask has R^D elements. The higher the dimension, the faster the number of elements with the size of the mask increases. Even small neighborhoods include hundreds or thousands of elements. Therefore, it will be a challenging task for higher-dimensional signal processing to develop efficient schemes to compute a neighborhood operation with as few computations as possible. Otherwise, it would not be possible to use them at all.

The challenge for efficient computation schemes is to decrease the number of computations from $O(R^D)$ to a lower order. This means that the number of computations is no longer proportional to R^D but rather to a lower order of the size R of the mask. The ultimate goal is to achieve computation schemes that increase only linearly with the size of the mask ($O(R^1)$) or, even better, do not depend at all on the size of the mask ($O(R^0)$).

9.1.3 Operator notation

In this section, we introduce an *operator notation* for signal-processing operations. It helps us to make complex composite neighbor operations easily comprehensible. All operators will be written in calligraphic letters, such as $\mathcal{B}, \mathcal{D}, \mathcal{H}, \mathcal{S}$. We write

$$\boldsymbol{G}' = \mathcal{H}\boldsymbol{G} \tag{9.3}$$

for an operator $\mathcal{H}$, which transforms the image $\boldsymbol{G}$ into the image $\boldsymbol{G}'$. Note that this notation can be used for any type of signal. It can be

used for continuous as well as digital signals and for signals of any dimension.

Consecutive application is denoted by writing the operators one after the other. The rightmost operator is applied first. Consecutive application of the same operator is expressed by an exponent

$$\underbrace{\mathcal{H}\mathcal{H}\ldots\mathcal{H}}_{\text{p times}} = \mathcal{H}^p \tag{9.4}$$

If the operator acts on a single image, the operand, which is to the right in the equations, will be omitted. In this way we can write operator equations without targets. Furthermore, we will use braces in the usual way to control the order of execution.

The operator notation leads to a *representation-independent notation* of signal-processing operations. A linear shift-invariant operator (see Section 8.6.3) performs a convolution operation in the spatial domain and a complex multiplication in the Fourier domain. With the operator notation, we can write the basic properties of the linear shift-invariant operator (Eq. (8.65)) in an easily comprehensible way and without specifying a target as

$$\begin{array}{ll} \text{commutativity} & \mathcal{H}_1\mathcal{H}_2 = \mathcal{H}_2\mathcal{H}_1 \\ \text{associativity} & \mathcal{H}_1(\mathcal{H}_2\mathcal{H}_3) = (\mathcal{H}_1\mathcal{H}_2)\mathcal{H}_3 \\ \text{distributivity over addition} & (\mathcal{H}_1+\mathcal{H}_2)\mathcal{H}_3 = \mathcal{H}_1\mathcal{H}_2 + \mathcal{H}_2\mathcal{H}_3 \end{array} \tag{9.5}$$

As can be seen from these equations, other operations such as addition can also be used in the operator notation. Care must be taken with any *nonlinear* operator. As soon as nonlinear operators are involved, the order in which the operators are executed must strictly be given. We retain the notation that operators are executed from the left to the right, provided that braces are not used to change the order of execution.

The point operation of pixelwise multiplication in the spatial domain is a simple example for a nonlinear operator. As this operator occurs frequently, it is denoted by a special symbol, a centered dot ($\cdot$). A special symbol is required in order to distinguish it from successive application of operators. The operator expression $\mathcal{B}(\mathcal{D}\cdot\mathcal{D})$, for instance, means: apply the operator $\mathcal{D}$ to the signal, square the result pixelwise, and then apply the operator $\mathcal{B}$. Without parentheses the expression $\mathcal{B}\mathcal{D}\cdot\mathcal{D}$ would mean: apply the operator $\mathcal{D}$ to the image and apply the operator $\mathcal{B}\mathcal{D}$ to the image and then multiply the results point by point. This notation thus gives precedence to the pointwise multiplication over consecutive operator execution. As a placeholder for an object onto which an operator is acting, we will use the symbol ":." In this notation, the forementioned operator combination is written as $\mathcal{B}(\mathcal{D}:\cdot\mathcal{D}:)$.