17

Edge-Preserving Smoothing Filters

Noise reduction in images is a common objective in image processing, not only for producing pleasing results for human viewing but also to facilitate easier extraction of meaningful information in subsequent steps, for example, in segmentation or feature detection. Simple smoothing filters, such as the Gaussian filter[1] and the filters discussed in Chapter 15 effectively perform low-pass filtering and thus remove high-frequency noise. However, they also tend to suppress high-rate intensity variations that are part of the original signal, thereby destroying image structures that are visually important. The filters described in this chapter are "edge preserving" in the sense that they change their smoothing behavior adaptively depending upon the local image structure. In general, maximum smoothing is performed over "flat" (uniform) image regions, while smoothing is reduced near or across edge-like structures, typically characterized by high intensity gradients.

In the following, three classical types of edge preserving filters are presented, which are largely based on different strategies. The *Kuwahara-type* filters described in Sec. 17.1 partition the filter kernel into smaller sub-kernels and select the most "homogeneous" of the underlying image regions for calculating the filter's result. In contrast, the *bilateral* filter in Sec. 17.2 uses the differences between pixel *values* to control how much each individual pixel in the filter region contributes to the local average. Pixels which are similar to the current center pixel contribute strongly, while highly different pixels add little to the result. Thus, in a sense, the bilateral filter is a non-homogeneous linear filter with a convolution kernel that is adaptively controlled by the local image content. Finally, the *anisotropic diffusion* filters in Sec. 17.3 iteratively smooth the image similar to the process of thermal diffusion, using the image gradient to block the local diffusion at edges and similar structures. It should be noted that all filters described in this chapter are nonlinear and can be applied to either grayscale or color images.

[1] See Chapter 5, Sec. 5.2.

© Springer-Verlag London 2016
W. Burger, M.J. Burge, *Digital Image Processing*, Texts in Computer Science,
DOI 10.1007/978-1-4471-6684-9_17

17.1 Kuwahara-Type Filters

The filters described in this section are all based on a similar concept that has its early roots in the work of Kuwahara et al. [144]. Although many variations have been proposed by other authors, we summarize them here under the term "Kuwahara-type" to indicate their origin and algorithmic similarities.

In principle, these filters work by calculating the mean and variance within neighboring image regions and selecting the mean value of the most "homogeneous" region, that is, the one with the smallest variance, to replace the original (center) pixel. For this purpose, the filter region R is divided into K partially overlapping subregions $R_1, R_2, \ldots, R_K$. At every image position (u, v), the *mean* μ_k and the *variance* σ_k^2 of each subregion R_k are calculated from the corresponding pixel values in I as

$$\mu_k(I,u,v) = \frac{1}{|R_k|} \cdot \sum_{(i,j)\in R_k} I(u+i, v+j) = \frac{1}{n_k} \cdot S_{1,k}(I,u,v), \tag{17.1}$$

$$\sigma_k^2(I,u,v) = \frac{1}{|R_k|} \cdot \sum_{(i,j)\in R_k} \big(I(u+i,v+j) - \mu_k(I,u,v)\big)^2 \tag{17.2}$$

$$= \frac{1}{|R_k|} \cdot \Big(S_{2,k}(I,u,v) - \frac{S_{1,k}^2(I,u,v)}{|R_k|}\Big), \tag{17.3}$$

for $k = 1, \ldots, K$, with[2]

$$S_{1,k}(I,u,v) = \sum_{(i,j)\in R_k} I(u+i, v+j), \tag{17.4}$$

$$S_{2,k}(I,u,v) = \sum_{(i,j)\in R_k} I^2(u+i, v+j). \tag{17.5}$$

The mean (μ) of the subregion with the smallest variance (σ^2) is selected as the update value, that is,

$$I'(u,v) \leftarrow \mu_{k'}(u,v), \quad \text{with } k' = \operatorname*{argmin}_{k=1,\ldots,K} \sigma_k^2(I,u,v). \tag{17.6}$$

The subregion structure originally proposed by Kuwahara et al. [144] is shown in Fig. 17.1(a) for a 3×3 filter ($r = 1$). It uses four square subregions of size $(r+1) \times (r+1)$ that overlap at the center. In general, the size of the whole filter is $(2r+1) \times (2r+1)$. This particular filter process is summarized in Alg. 17.1.

Note that this filter does not have a centered subregion, which means that the center pixel is always replaced by the mean of one of the neighboring regions, even if it had perfectly fit the surrounding values. Thus the filter always performs a spatial shift, which introduces jitter and banding artifacts in regions of smooth intensity change. This effect is reduced with the filter proposed by Tomita and Tsuji [230], which is similar but includes a fifth subregion at its center (Fig. 17.1(b)). Filters of arbitrary size can be built by simply scaling the corresponding structure. In case of the *Tomita-Tsuji* filter, the side length of the subregions should be odd.

[2] $|R_k|$ denotes the size (number of pixels) of the subregion R_k.

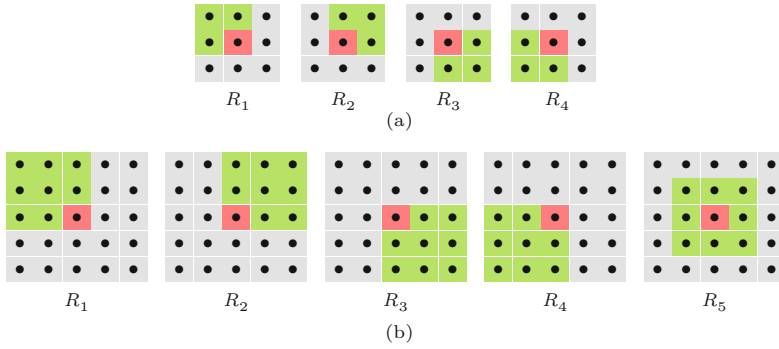

Fig. 17.1
Subregion structures for Kuwahara-type filters. The orginal *Kuwahara-Hachimura* filter (a) considers *four* square, overlapping subregions [144]. *Tomita-Tsuji* filter (b) with *five* subregions ($r = 2$). The current center pixel (red) is contained in all subregions. Das aktuelle Zentralpixel (rot) ist in allen Subregionen enthalten.

Note that replacing a pixel value by the mean of a square neighborhood is equivalent to linear filtering with a simple box kernel, which is not an optimal smoothing operator. To reduce the artifacts caused by the square subregions, alternative filter structures have been proposed, such as the 5×5 *Nagao-Matsuyama* filter [170] shown in Fig. 17.2.

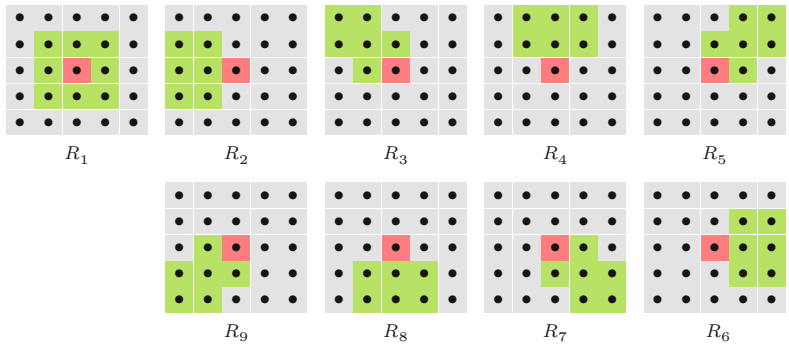

Fig. 17.2
Subregions for the 5×5 ($r = 2$) *Nagao-Matsuyama* filter [170]. Note that the centered subregion (R_1) has a different size than the remaining subregions ($R_2, \ldots, R_9$).

If all subregions are of identical *size* $|R_k| = n$, the quantities

$$\sigma_k^2(I,u,v) \cdot n \;\; = S_{2,k}(I,u,v) - S_{1,k}^2(I,u,v)/n \quad \text{or} \tag{17.7}$$

$$\sigma_k^2(I,u,v) \cdot n^2 = S_{2,k}(I,u,v) \cdot n - S_{1,k}^2(I,u,v) \tag{17.8}$$

can be used to measure the amount of variation within the corresponding subregion. Both expressions require calculating one multiplication less for each pixel than the "real" variance σ_k^2 in Eqn. (17.3). Moreover, if all subregions have the same *shape* (such as the filters in Fig. 17.1), additional optimizations are possible that substantially improve the performance. In this case, the local mean and variance need to be calculated only once over a fixed neighborhood for each image position. This type of filter can be efficiently implemented by using a set of pre-calculated maps for the local variance and mean values, as described in Alg. 17.2. As before, the parameter r specifies the radius of the composite filter, with subregions of size $(r+1) \times (r+1)$ and overall size $(2r+1) \times (2r+1)$. The individual subregions are of size $(r+1) \times (r+1)$; for example, $r = 2$ for the 5×5 filter shown in Fig. 17.1(b).

All these filters tend to generate banding artifacts in smooth image regions due to erratic spatial displacements, which become worse

Alg. 17.1 Simple *Kuwahara-Hachimura* filter.

```
1:  KuwaharaFilter(I)
        Input: I, a grayscale image of size M×N.
        Returns a new (filtered) image of size M×N.
2:      R_1 ← {(−1,−1),(0,−1),(−1,0),(0,0)}
3:      R_2 ← {(0,−1),(1,−1),(0,0),(1,0)}
4:      R_3 ← {(0,0),(1,0),(1,0),(1,1)}
5:      R_4 ← {(−1,0),(0,0),(−1,1),(1,0)}
6:      I' ← Duplicate(I)
7:      (M,N) ← Size(I)
8:      for all image coordinates (u,v) ∈ M×N do
9:          σ²_min ← ∞
10:         for R ← R_1,...,R_4 do
11:             (σ²,μ) ← EvalSubregion(I,R,u,v)
12:             if σ² < σ²_min then
13:                 σ²_min ← σ²
14:                 μ_min ← μ
15:         I'(u,v) ← μ_min
16:     return I'
```

```
17: EvalSubregion(I,R,u,v)
        Returns the variance and mean of the grayscale image I for the
        subregion R positioned at (u,v).
18:     n ← Size(R)
19:     S_1 ← 0,   S_2 ← 0
20:     for all (i,j) ∈ R do
21:         a ← I(u+i,v+j)
22:         S_1 ← S_1 + a                          ▷ Eq. 17.4
23:         S_2 ← S_2 + a²                         ▷ Eq. 17.5
24:     σ² ← (S_2 − S_1²/n)/n      ▷ variance of subregion R, see Eq. 17.1
25:     μ ← S_1/n                  ▷ mean of subregion R, see Eq. 17.3
26:     return (σ²,μ)
```

with increasing filter size. If a centered subregion is used (such as R_5 in Fig. 17.1 or R_1 in Fig. 17.2), one could reduce this effect by applying a threshold (t_σ) to select any off-center subregion R_k *only* if its variance is significantly smaller than the variance of the center region R_1 (see Alg. 17.2, line 13).

17.1.1 Application to Color Images

While all of the aforementioned filters were originally designed for grayscale images, they are easily modified to work with color images. We only need to specify how to calculate the variance and mean for any subregion; the decision and replacement mechanisms then remain the same.

Given an RGB color image $\boldsymbol{I} = (I_\mathrm{R}, I_\mathrm{G}, I_\mathrm{B})$ with a subregion R_k, we can calculate the local mean and variance for each color channel as

$$\boldsymbol{\mu}_k(\boldsymbol{I},u,v) = \begin{pmatrix} \mu_k(I_\mathrm{R},u,v) \\ \mu_k(I_\mathrm{G},u,v) \\ \mu_k(I_\mathrm{B},u,v) \end{pmatrix}, \qquad \boldsymbol{\sigma}_k^2(\boldsymbol{I},u,v) = \begin{pmatrix} \sigma_k^2(I_\mathrm{R},u,v) \\ \sigma_k^2(I_\mathrm{G},u,v) \\ \sigma_k^2(I_\mathrm{B},u,v) \end{pmatrix}, \quad (17.9)$$

Alg. 17.2
Fast Kuwahara-type (Tomita-Tsuji) filter with variable size and fixed subregion structure. The filter uses five square subregions of size $(r+1)\times(r+1)$, with a composite filter of $(2r+1)\times(2r+1)$, as shown in Fig. 17.1(b). The purpose of the variance threshold t_σ is to reduce banding effects in smooth image regions (typically $\mathsf{t}_\sigma = 5, \ldots, 50$ for 8-bit images).

FastKuwaharaFilter$(I, r, \mathsf{t}_\sigma)$
Input: I, a grayscale image of size $M\times N$; r, filter radius ($r \geq 1$); t_σ, variance threshold.
Returns a new (filtered) image of size $M \times N$.

$(M, N) \leftarrow \mathsf{Size}(I)$
Create maps:
$\mathsf{S} : M\times N \to \mathbb{R}$ ▷ local variance $\mathsf{S}(u,v) \equiv n \cdot \sigma^2(I,u,v)$
$\mathsf{A} : M\times N \to \mathbb{R}$ ▷ local mean $\mathsf{A}(u,v) \equiv \mu(I,u,v)$
$d_{\min} \leftarrow (r \div 2) - r$ ▷ subregions' left/top position
$d_{\max} \leftarrow d_{\min} + r$ ▷ subregions' right/bottom position
for all image coordinates $(u,v) \in M\times N$ **do**
  $(s,\mu) \leftarrow \mathsf{EvalSquareSubregion}(I,u,v,d_{\min},d_{\max})$
  $\mathsf{S}(u,v) \leftarrow s$
  $\mathsf{A}(u,v) \leftarrow \mu$
$n \leftarrow (r+1)^2$ ▷ fixed subregion size
$I' \leftarrow \mathsf{Duplicate}(I)$
for all image coordinates $(u,v) \in M\times N$ **do**
  $s_{\min} \leftarrow \mathsf{S}(u,v) - \mathsf{t}_\sigma \cdot n$ ▷ variance of center region
  $\mu_{\min} \leftarrow \mathsf{A}(u,v)$ ▷ mean of center region
  for $p \leftarrow d_{\min}, \ldots, d_{\max}$ **do**
    for $q \leftarrow d_{\min}, \ldots, d_{\max}$ **do**
      if $\mathsf{S}(u+p, v+q) < s_{\min}$ **then**
        $s_{\min} \leftarrow \mathsf{S}(u+p, v+q)$
        $\mu_{\min} \leftarrow \mathsf{A}(u+p, v+q)$
  $I'(u,v) \leftarrow \mu_{\min}$
return I'

EvalSquareSubregion$(I, u, v, d_{\min}, d_{\max})$
Returns the variance and mean of the grayscale image I for a square subregion positioned at (u,v).

$S_1 \leftarrow 0, \quad S_2 \leftarrow 0$
for $i \leftarrow d_{\min}, \ldots, d_{\max}$ **do**
  for $j \leftarrow d_{\min}, \ldots, d_{\max}$ **do**
    $a \leftarrow I(u+i, v+j)$
    $S_1 \leftarrow S_1 + a$ ▷ Eq. 17.4
    $S_2 \leftarrow S_2 + a^2$ ▷ Eq. 17.5
$s \leftarrow S_2 - S_1^2/n$ ▷ subregion variance ($s \equiv n \cdot \sigma^2$)
$\mu \leftarrow S_1/n$ ▷ subregion mean (μ)
return (s, μ)

with $\mu_k()$, $\sigma_k^2()$ as defined in Eqns. (17.1) and (17.3), respectively. Analogous to the grayscale case, each pixel is then replaced by the average color in the subregion with the smallest variance, that is,

$$\boldsymbol{I}'(u,v) \leftarrow \boldsymbol{\mu}_{k'}(\boldsymbol{I},u,v), \quad \text{with } k' = \operatorname*{argmin}_{k=1,\ldots,K} \sigma^2_{k,\mathrm{RGB}}(\boldsymbol{I},u,v). \quad (17.10)$$

The overall variance $\sigma^2_{k,\mathrm{RGB}}$, used to determine k' in Eqn. (17.10), can be defined in different ways, for example, as the sum of the variances in the individual color channels, that is,

$$\sigma^2_{k,\mathrm{RGB}}(\boldsymbol{I},u,v) = \sigma^2_k(I_\mathrm{R},u,v) + \sigma^2_k(I_\mathrm{G},u,v) + \sigma^2_k(I_\mathrm{B},u,v). \quad (17.11)$$

Alg. 17.3 Color version of the *Kuwahara-type* filter (adapted from Alg. 17.1). The algorithm uses the definition in Eqn. (17.11) for the total variance σ^2 in the subregion R (see line 25). The vector $\boldsymbol{\mu}$ (calculated in line 26) is the average color of the subregion.

1: **KuwaharaFilterColor**($\boldsymbol{I}$)
Input: $\boldsymbol{I}$, an RGB image of size $M \times N$.
Returns a new (filtered) color image of size $M \times N$.
2: $R_1 \leftarrow \{(-1,-1),(0,-1),(-1,0),(0,0)\}$
3: $R_2 \leftarrow \{(0,-1),(1,-1),(0,0),(1,0)\}$
4: $R_3 \leftarrow \{(0,0),(1,0),(1,0),(1,1)\}$
5: $R_4 \leftarrow \{(-1,0),(0,0),(-1,1),(1,0)\}$
6: $\boldsymbol{I}' \leftarrow \mathsf{Duplicate}(\boldsymbol{I})$
7: $(M,N) \leftarrow \mathsf{Size}(\boldsymbol{I})$
8: **for all** image coordinates $(u,v) \in M \times N$ **do**
9: $\sigma^2_{\min} \leftarrow \infty$
10: **for** $R \leftarrow R_1, \ldots, R_4$ **do**
11: $(\sigma^2, \boldsymbol{\mu}) \leftarrow \mathsf{EvalSubregion}(\boldsymbol{I}, R_k, u, v)$
12: **if** $\sigma^2 < \sigma^2_{\min}$ **then**
13: $\sigma^2_{\min} \leftarrow \sigma^2$
14: $\boldsymbol{\mu}_{\min} \leftarrow \boldsymbol{\mu}$
15: $\boldsymbol{I}'(u,v) \leftarrow \boldsymbol{\mu}_{\min}$
16: **return** $\boldsymbol{I}'$

17: **EvalSubregion**($\boldsymbol{I}, R, u, v$)
Returns the total variance and the mean vector of the color image $\boldsymbol{I}$ for the subregion R positioned at (u,v).
18: $n \leftarrow \mathsf{Size}(R)$
19: $\boldsymbol{S}_1 \leftarrow \mathbf{0}, \quad \boldsymbol{S}_2 \leftarrow \mathbf{0}$ ▷ $\boldsymbol{S}_1, \boldsymbol{S}_2 \in \mathbb{R}^3$
20: **for all** $(i,j) \in R$ **do**
21: $\boldsymbol{a} \leftarrow \boldsymbol{I}(u+i, v+j)$ ▷ $\boldsymbol{a} \in \mathbb{R}^3$
22: $\boldsymbol{S}_1 \leftarrow \boldsymbol{S}_1 + \boldsymbol{a}$
23: $\boldsymbol{S}_2 \leftarrow \boldsymbol{S}_2 + \boldsymbol{a}^2$ ▷ $\boldsymbol{a}^2 = \boldsymbol{a} \cdot \boldsymbol{a}$ (dot product)
24: $\boldsymbol{S} \leftarrow \left(\boldsymbol{S}_2 - \boldsymbol{S}_1^2 \cdot \frac{1}{n}\right) \cdot \frac{1}{n}$ ▷ $\boldsymbol{S} = (\sigma^2_{\mathrm{R}}, \sigma^2_{\mathrm{G}}, \sigma^2_{\mathrm{B}})$
25: $\sigma^2_{\mathrm{RGB}} \leftarrow \Sigma \boldsymbol{S}$ ▷ $\sigma^2_{\mathrm{RGB}} = \sigma^2_{\mathrm{R}} + \sigma^2_{\mathrm{G}} + \sigma^2_{\mathrm{B}}$, total variance in R
26: $\boldsymbol{\mu} \leftarrow \frac{1}{n} \cdot \boldsymbol{S}_1$ ▷ $\boldsymbol{\mu} \in \mathbb{R}^3$, avg. color vector for subregion R
27: **return** $(\sigma^2_{\mathrm{RGB}}, \boldsymbol{\mu})$

This is sometimes called the "total variance". The resulting filter process is summarized in Alg. 17.3 and color examples produced with this algorithm are shown in Figs. 17.3 and 17.4.

Alternatively [109], one could define the combined color variance as the norm of the *color covariance matrix*[3] for the subregion R_k,

$$\Sigma_k(\boldsymbol{I}, u, v) = \begin{pmatrix} \sigma_{k,\mathrm{RR}} & \sigma_{k,\mathrm{RG}} & \sigma_{k,\mathrm{RB}} \\ \sigma_{k,\mathrm{GR}} & \sigma_{k,\mathrm{GG}} & \sigma_{k,\mathrm{GB}} \\ \sigma_{k,\mathrm{BR}} & \sigma_{k,\mathrm{BG}} & \sigma_{k,\mathrm{BB}} \end{pmatrix}, \tag{17.12}$$

$$\text{with} \qquad \sigma_{k,pq} = \frac{1}{|R_k|} \cdot \sum_{(i,j) \in R_k} \big[I_p(u+i, v+j) - \mu_k(I_p, u, v)\big] \cdot \big[I_q(u+i, v+j) - \mu_k(I_q, u, v)\big], \tag{17.13}$$

for all possible color pairs $(p,q) \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}^2$. Note that $\sigma_{k,pp} = \sigma^2_{k,p}$ and $\sigma_{k,pq} = \sigma_{k,qp}$, and thus the matrix Σ_k is symmetric and only 6 of its 9 entries need to be calculated. The (Frobenius) *norm* of the 3×3 color covariance matrix is defined as

[3] See Sec. D.2 in the Appendix for details.

(a) RGB test image with selected details

(b) $r = 1$ (3×3 filter)

(c) $r = 2$ (5×5 filter)

(d) $r = 3$ (7×7 filter)

(e) $r = 4$ (9×9 filter)

Fig. 17.3
Kuwahara-type (*Tomita-Tsuji*) filter—color example using the variance definition in Eqn. (17.11). The filter radius is varied from $r = 1$ (b) to $r = 4$ (e).

$$\sigma^2_{k,\text{RGB}} = \|\Sigma_k(\boldsymbol{I}, u, v)\|_2^2 = \sum_{\substack{p,q \in \\ \{\text{R,G,B}\}}} (\sigma_{k,pq})^2. \tag{17.14}$$

Note that the total variance in Eqn. (17.11)—which is simpler to calculate than this norm—is equivalent to the *trace* of the covariance matrix Σ_k.

(a) 5 × 5 *Tomita-Tsuji* filter ($r = 2$)

(b) 5 × 5 *Nagao-Matsuyama* filter

Fig. 17.4 Color versions of the *Tomita-Tsuji* (Fig. 17.1(b)) and *Nagao-Matsuyama* filter (Fig. 17.2). Both filters are of size 5 × 5 and use the variance definition in Eqn. (17.11). Results are visually similar, but in general the *Nagao-Matsuyama* filter is slightly less destructive on diagonal structures. Original image in Fig. 17.3(a).

Since each pixel of the filtered image is calculated as the *mean* (i.e., a linear combination) of a set of original color pixels, the results depend on the color space used, as discussed in Chapter 15, Sec. 15.1.2.

17.2 Bilateral Filter

Traditional linear smoothing filters operate by convolving the image with a kernel, whose coefficients act as weights for the corresponding image pixels and only depend on the spatial distance from the center coordinate. Pixels close to the filter center are typically given larger weights while pixels at a greater distance carry smaller weights. Thus the convolution kernel effectively encodes the closeness of the underlying pixels in space. In the following, a filter whose weights depend only on the distance in the spatial domain is called a *domain filter*.

To make smoothing filters less destructive on edges, a typical strategy is to exclude individual pixels from the filter operation or to reduce the weight of their contribution if they are very dissimilar *in value* to the pixel found at the center position. This operation too can be formulated as a filter, but this time the kernel coefficients depend only upon the differences in pixel *values* or *range*. Therefore this is called a *range filter*, as explained in more detail Sec. 17.2.2. The idea of the *bilateral* filter, proposed by Tomasi and Manduchi in [229], is to *combine* both domain and range filtering into a common, edge-preserving smoothing filter.

17.2.1 Domain Filter

In an ordinary 2D linear filter (or "convolution") operation,[4]

[4] See also Chapter 5, Eqn. (5.5) on page 92.

$$I'(u,v) \leftarrow \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(u+m, v+n) \cdot H(m,n) \tag{17.15}$$

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) \cdot H(i-u, j-v), \tag{17.16}$$

every new pixel value $I'(u,v)$ is the weighted average of the original image pixels I in a certain neighborhood, with the weights specified by the elements of the filter kernel H.[5] The weight assigned to each pixel only depends on its spatial position relative to the current center coordinate (u,v). In particular, $H(0,0)$ specifies the weight of the center pixel $I(u,v)$, and $H(m,n)$ is the weight assigned to a pixel displaced by (m,n) from the center. Since only the spatial image coordinates are relevant, such a filter is called a *domain filter*. Obviously, ordinary filters as we know them are *all* domain filters.

17.2.2 Range Filter

Although the idea may appear strange at first, one could also apply a linear filter to the pixel *values* or *range* of an image in the form

$$I'_r(u,v) \leftarrow \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) \cdot H_{\mathrm{r}}\big(I(i,j) - I(u,v)\big). \tag{17.17}$$

The contribution of each pixel is specified by the function H_{r} and depends on the difference between its own *value* $I(i,j)$ and the value at the current center pixel $I(u,v)$. The operation in Eqn. (17.17) is called a *range filter*, where the spatial position of a contributing pixel is irrelevant and only the difference in values is considered. For a given position (u,v), all surrounding image pixels $I(i,j)$ with the same value contribute equally to the result $I'_r(u,v)$. Consequently, the application of a *range* filter has no *spatial* effect upon the image—in contrast to a *domain* filter, no blurring or sharpening will occur. Instead, a range filter effectively performs a global *point operation* by remapping the intensity or color values. However, a global *range filter* by itself is of little use, since it combines pixels from the entire image and only changes the intensity or color map of the image, equivalent to a nonlinear, image-dependent point operation.

17.2.3 Bilateral Filter—General Idea

The key idea behind the bilateral filter is to *combine* domain filtering (Eqn. (17.16)) *and* range filtering (Eqn. (17.17)) in the form

$$I'(u,v) = \frac{1}{W_{u,v}} \cdot \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) \cdot \underbrace{H_{\mathrm{d}}(i-u, j-v) \cdot H_{\mathrm{r}}\big(I(i,j) - I(u,v)\big)}_{w_{i,j}}, \tag{17.18}$$

[5] In Eqn. (17.16), functions I and H are assumed to be zero outside their domains of definition.

where H_d, H_r are the *domain* and *range* kernels, respectively, $w_{i,j}$ are the composite weights, and

$$W_{u,v} = \sum_{\substack{i=\\-\infty}}^{\infty} \sum_{\substack{j=\\-\infty}}^{\infty} w_{i,j} = \sum_{\substack{i=\\-\infty}}^{\infty} \sum_{\substack{j=\\-\infty}}^{\infty} H_\mathrm{d}(i-u, j-v) \cdot H_\mathrm{r}\big(I(i,j)-I(u,v)\big) \tag{17.19}$$

is the (position-dependent) sum of the weights $w_{i,j}$ used to normalize the combined filter kernel.

In this form, the scope of range filtering is constrained to the spatial neighborhood defined by the domain kernel H_d. At a given filter position (u, v), the weight $w_{i,j}$ assigned to each contributing pixel depends upon (1) its spatial position relative to (u, v), and (2) the similarity of its pixel value to the value at the center position (u, v). In other words, the resulting pixel is the weighted average of pixels that are nearby *and* similar to the original pixel. In a flat image region, where most surrounding pixels have values similar to the center pixel, the bilateral filter acts as a conventional smoothing filter, controlled only by the domain kernel H_d. However, when placed near a step edge or on an intensity ridge, only those pixels are included in the smoothing process that are similar in value to the center pixel, thus avoiding blurring the edges.

If the domain kernel H_d has a limited radius D, or size $(2D{+}1) \times (2D{+}1)$, the bilateral filter defined in Eqn. (17.18) can be written as

$$I'(u,v) = \frac{\displaystyle\sum_{\substack{i=\\u-D}}^{u+D} \sum_{\substack{j=\\v-D}}^{v+D} I(i,j) \cdot H_\mathrm{d}(i-u, j-v) \cdot H_\mathrm{r}\left(I(i,j)-I(u,v)\right)}{\displaystyle\sum_{\substack{i=\\u-D}}^{u+D} \sum_{\substack{j=\\v-D}}^{v+D} H_\mathrm{d}(i-u, j-v) \cdot H_\mathrm{r}\left(I(i,j)-I(u,v)\right)} \tag{17.20}$$

$$= \frac{\displaystyle\sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} I(u+m, v+n) \cdot H_\mathrm{d}(m,n) \cdot H_\mathrm{r}\left(I(u+m,v+n)-I(u,v)\right)}{\displaystyle\sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} H_\mathrm{d}(m,n) \cdot H_\mathrm{r}\left(I(u+m,v+n)-I(u,v)\right)} \tag{17.21}$$

(by substituting $(i-u) \to m$ and $(j-v) \to n$). The effective, space variant filter kernel for the image I at position (u, v) then is

$$\bar{H}_{I,u,v}(i,j) = \frac{H_\mathrm{d}(i,j) \cdot H_\mathrm{r}\big(I(u+i,v+j)-I(u,v)\big)}{\displaystyle\sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} H_\mathrm{d}(m,n) \cdot H_\mathrm{r}\left(I(u+m,v+n)-I(u,v)\right)}, \tag{17.22}$$

for $-D \le i, j \le D$, whereas $\bar{H}_{I,u,v}(i,j) = 0$ otherwise. This quantity specifies the contribution of the original image pixels $I(u{+}i, v{+}j)$ to the resulting new pixel value $I'(u, v)$.

17.2.4 Bilateral Filter with Gaussian Kernels

A special (but common) case is the use of Gaussian kernels for both the domain and the range parts of the bilateral filter. The discrete 2D Gaussian *domain kernel* of width σ_d is defined as

$$H_\mathrm{d}^{\mathrm{G},\sigma_\mathrm{d}}(m,n) = \frac{1}{2\pi\sigma_\mathrm{d}^2} \cdot e^{-\frac{\rho^2}{2\sigma_\mathrm{d}^2}} = \frac{1}{2\pi\sigma_\mathrm{d}^2} \cdot e^{-\frac{m^2+n^2}{2\sigma_\mathrm{d}^2}} \tag{17.23}$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma_\mathrm{d}} \cdot \exp\Big(-\tfrac{m^2}{2\sigma_\mathrm{d}^2}\Big) \cdot \frac{1}{\sqrt{2\pi}\,\sigma_\mathrm{d}} \cdot \exp\Big(-\tfrac{n^2}{2\sigma_\mathrm{d}^2}\Big), \tag{17.24}$$

for $m, n \in \mathbb{Z}$. It has its maximum at the center ($m = n = 0$) and declines smoothly and isotropically with increasing radius $\rho = \sqrt{m^2+n^2}$; for $\rho > 3.5\sigma_\mathrm{d}$, $H_\mathrm{d}^{\mathrm{G},\sigma_\mathrm{d}}(m,n)$ is practically zero. The factorization in Eqn. (17.24) indicates that the Gaussian 2D kernel can be separated into 1D Gaussians, allowing for a more efficient implementation.[6] The constant factors $1/(\sqrt{2\pi}\,\sigma_\mathrm{d})$ can be omitted in practice, since the bilateral filter requires individual normalization at each image position (Eqn. (17.19)).

Similarly, the corresponding *range filter kernel* is defined as a (continuous) *1*D Gaussian of width σ_r,

$$H_\mathrm{r}^{\mathrm{G},\sigma_\mathrm{r}}(x) = \frac{1}{\sqrt{2\pi}\,\sigma_\mathrm{r}} \cdot e^{-\frac{x^2}{2\sigma_\mathrm{r}^2}} = \frac{1}{\sqrt{2\pi}\,\sigma_\mathrm{r}} \cdot \exp\Big(-\tfrac{x^2}{2\sigma_\mathrm{r}^2}\Big), \tag{17.25}$$

for $x \in \mathbb{R}$. The constant factor $1/(\sqrt{2\pi}\,\sigma_\mathrm{r})$ may again be omitted and the resulting composite filter (Eqn. (17.18)) can thus be written as

$$I'(u,v) = \frac{1}{W_{u,v}} \cdot \sum_{\substack{i=\\u-D}}^{u+D} \sum_{\substack{j=\\v-D}}^{v+D} \Big[I(i,j) \cdot H_\mathrm{d}^{\mathrm{G},\sigma_\mathrm{d}}(i-u, j-v) \cdot H_\mathrm{r}^{\mathrm{G},\sigma_\mathrm{r}}(I(i,j) - I(u,v)) \Big] \tag{17.26}$$

$$= \frac{1}{W_{u,v}} \cdot \sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} \Big[I(u+m, v+n) \cdot H_\mathrm{d}^{\mathrm{G},\sigma_\mathrm{d}}(m,n) \cdot H_\mathrm{r}^{\mathrm{G},\sigma_\mathrm{r}}(I(u+m,v+n) - I(u,v)) \Big] \tag{17.27}$$

$$= \frac{1}{W_{u,v}} \cdot \sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} \Big[I(u+m, v+n) \cdot \exp\big(-\tfrac{m^2+n^2}{2\sigma_\mathrm{d}^2}\big) \cdot \exp\big(-\tfrac{(I(u+m,v+n)-I(u,v))^2}{2\sigma_\mathrm{r}^2}\big) \Big], \tag{17.28}$$

with $D = \lceil 3.5 \cdot \sigma_\mathrm{d} \rceil$ and

$$W_{u,v} = \sum_{\substack{m=\\-D}}^{D} \sum_{\substack{n=\\-D}}^{D} \exp\big(-\tfrac{m^2+n^2}{2\sigma_\mathrm{d}^2}\big) \cdot \exp\big(-\tfrac{(I(u+m,v+n)-I(u,v))^2}{2\sigma_\mathrm{r}^2}\big). \tag{17.29}$$

For 8-bit grayscale images, with pixel values in the range $[0, 255]$, the width of the range kernel is typically set to $\sigma_\mathrm{r} = 10, \ldots, 50$. The width of the domain kernel (σ_d) depends on the desired amount of spatial smoothing. Algorithm 17.4 gives a summary of the steps involved in bilateral filtering for grayscale images.

[6] See also Chapter 5, Sec. 5.3.3.

Alg. 17.4 Bilateral filter with Gaussian kernels (grayscale version).

BilateralFilterGray$(I, \sigma_\mathrm{d}, \sigma_\mathrm{r})$

Input: I, a grayscale image of size $M \times N$; σ_d, width of the 2D Gaussian *domain* kernel; σ_r, width of the 1D Gaussian *range* kernel. Returns a new filtered image of size $M \times N$.

$(M, N) \leftarrow \mathsf{Size}(I)$
$D \leftarrow \lceil 3.5 \cdot \sigma_\mathrm{d} \rceil$ ▷ width of domain filter kernel
$I' \leftarrow \mathsf{Duplicate}(I)$
for all image coordinates $(u, v) \in M \times N$ **do**
  $S \leftarrow 0$ ▷ sum of weighted pixel values
  $W \leftarrow 0$ ▷ sum of weights
  $a \leftarrow I(u, v)$ ▷ center pixel value
  for $m \leftarrow -D, \ldots, D$ **do**
    for $n \leftarrow -D, \ldots, D$ **do**
      $b \leftarrow I(u+m, v+n)$ ▷ off-center pixel value
      $w_\mathrm{d} \leftarrow \exp\left(-\frac{m^2+n^2}{2\sigma_\mathrm{d}^2}\right)$ ▷ domain coefficient
      $w_\mathrm{r} \leftarrow \exp\left(-\frac{(a-b)^2}{2\sigma_\mathrm{r}^2}\right)$ ▷ range coefficient
      $w \leftarrow w_\mathrm{d} \cdot w_\mathrm{r}$ ▷ composite coefficient
      $S \leftarrow S + w \cdot b$
      $W \leftarrow W + w$
  $I'(u, v) \leftarrow S/W$
return I'

Figures 17.5–17.9 show the effective, space-variant filter kernels (see Eqn. (17.22)) and the results of applying a bilateral filter with Gaussian domain and range kernels in different situations. Uniform noise was applied to the original images to demonstrate the filtering effect. One can see clearly how the range part makes the combined filter kernel adapt to the local image structure. Only those surrounding parts that have brightness values similar to the center pixel are included in the filter operation. The filter parameters were set to $\sigma_\mathrm{d} = 2.0$ and $\sigma_\mathrm{r} = 50$; the domain kernel is of size 15×15.

17.2.5 Application to Color Images

Linear smoothing filters are typically used on color images by separately applying the same filter to the individual color channels. As discussed in Chapter 15, Sec. 15.1, this is legitimate if a suitable working color space is used to avoid the introduction of unnatural intensity and chromaticity values. Thus, for the domain-part of the bilateral filter, the same considerations apply as for any linear smoothing filter. However, as will be described, the bilateral filter as a whole cannot be implemented by filtering the color channels separately.

In the *range* part of the filter, the weight assigned to each contributing pixel depends on its difference to the value of the center pixel. Given a suitable distance measure $\mathrm{dist}(\boldsymbol{a}, \boldsymbol{b})$ between two color vectors $\boldsymbol{a}, \boldsymbol{b}$, the bilateral filter in Eqn. (17.18) can be easily modified for a color image $\boldsymbol{I}$ to

$$\boldsymbol{I}'(u,v) = \frac{1}{W_{u,v}} \cdot \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \boldsymbol{I}(i,j) \cdot H_\mathrm{d}(i-u, j-v) \cdot H_\mathrm{r}\big(\mathrm{dist}(\boldsymbol{I}(i,j), \boldsymbol{I}(u,v))\big), \tag{17.30}$$

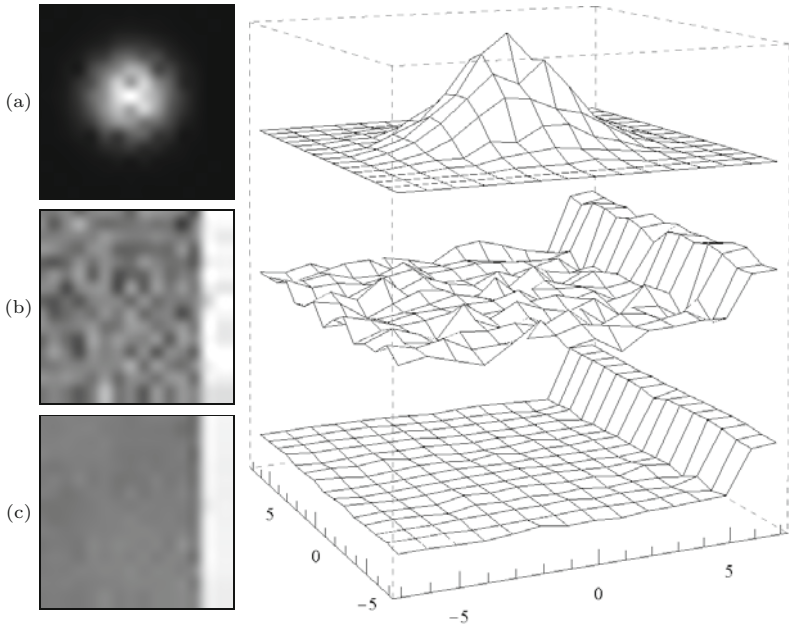

Fig. 17.5
Bilateral filter response when positioned in a flat, noisy image region. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

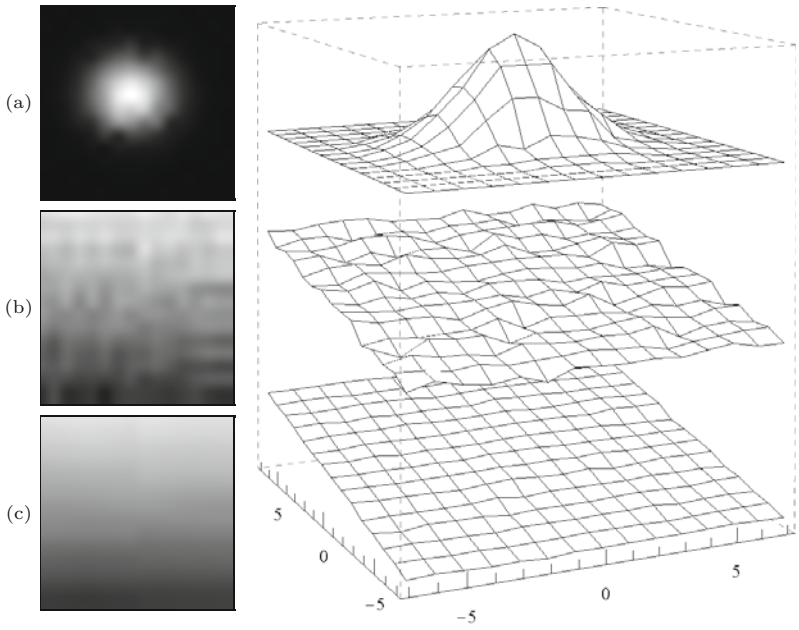

Fig. 17.6
Bilateral filter response when positioned on a linear ramp. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

with

$$W_{u,v} = \sum_{i,j} H_{\mathrm{d}}(i-u, j-v) \cdot H_{\mathrm{r}}\big(\mathrm{dist}(\boldsymbol{I}(i,j), \boldsymbol{I}(u,v))\big). \qquad (17.31)$$

It is common to use one of the popular norms for measuring color distances, such as the L_1, L_2 (Euclidean), or the L_∞ (maximum) norms, for example,

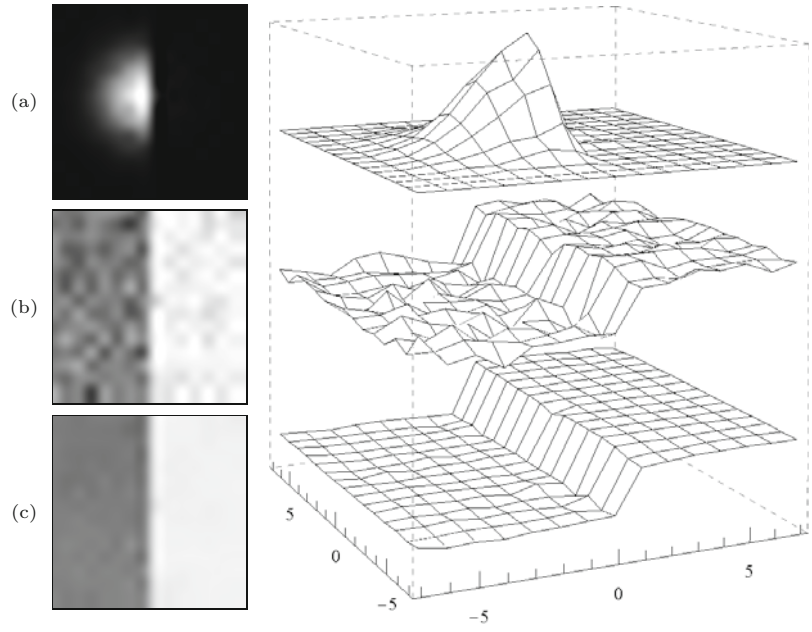

Fig. 17.7
Bilateral filter response when positioned left to a vertical step edge. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

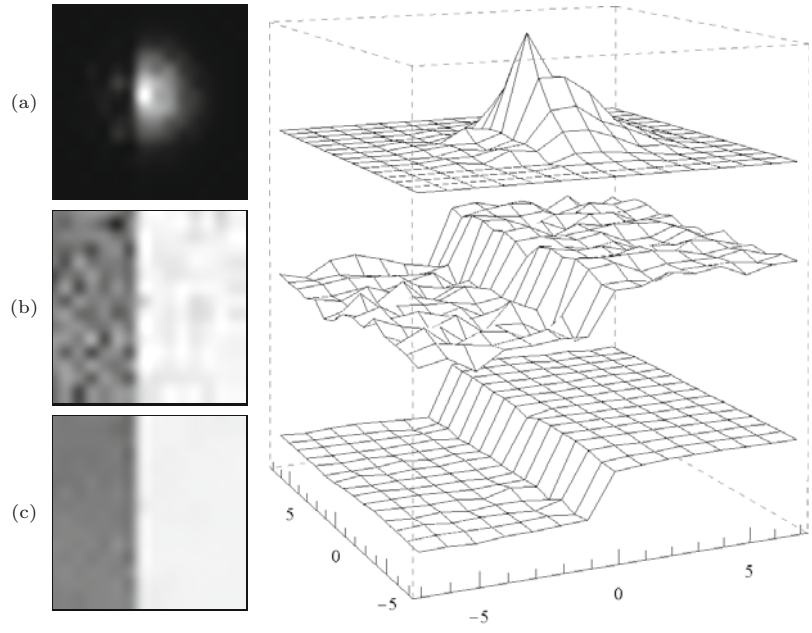

Fig. 17.8
Bilateral filter response when positioned right to a vertical step edge. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

$$\mathrm{dist}_1(\boldsymbol{a},\boldsymbol{b}) \;:=\; \tfrac{1}{3}\cdot\|\boldsymbol{a}-\boldsymbol{b}\|_1 = \tfrac{1}{3}\cdot\textstyle\sum_{k=1}^{K}|a_k-b_k|\,, \tag{17.32}$$

$$\mathrm{dist}_2(\boldsymbol{a},\boldsymbol{b}) \;:=\; \tfrac{1}{\sqrt{3}}\cdot\|\boldsymbol{a}-\boldsymbol{b}\|_2 = \tfrac{1}{\sqrt{3}}\cdot\Big(\textstyle\sum_{k=1}^{K}(a_k-b_k)^2\Big)^{1/2}, \tag{17.33}$$

$$\mathrm{dist}_\infty(\boldsymbol{a},\boldsymbol{b}) := \|\boldsymbol{a}-\boldsymbol{b}\|_\infty = \max_k |a_k-b_k|\,. \tag{17.34}$$

The normalizing factors $1/3$ and $1/\sqrt{3}$ in Eqns. (17.32)–(17.33) are necessary to obtain results comparable in magnitude to those of

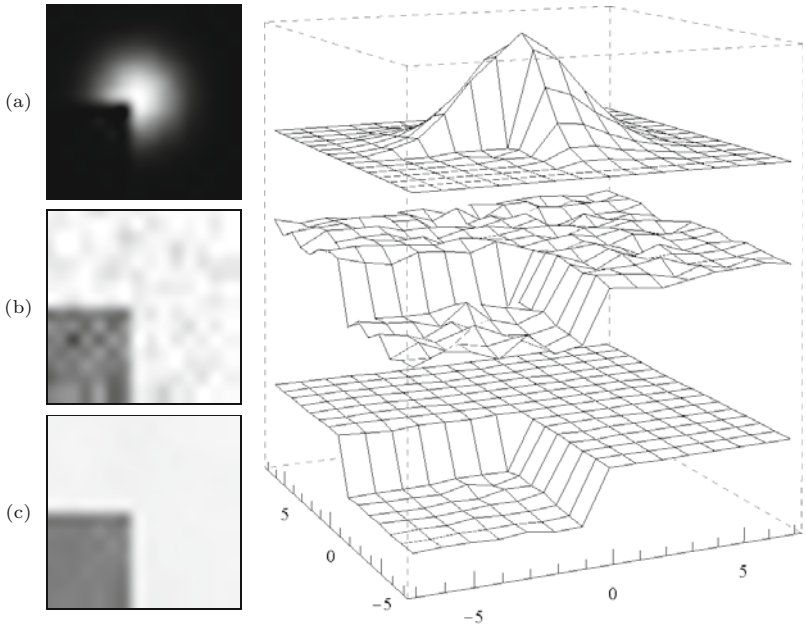

Fig. 17.9
Bilateral filter response when positioned at a corner. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

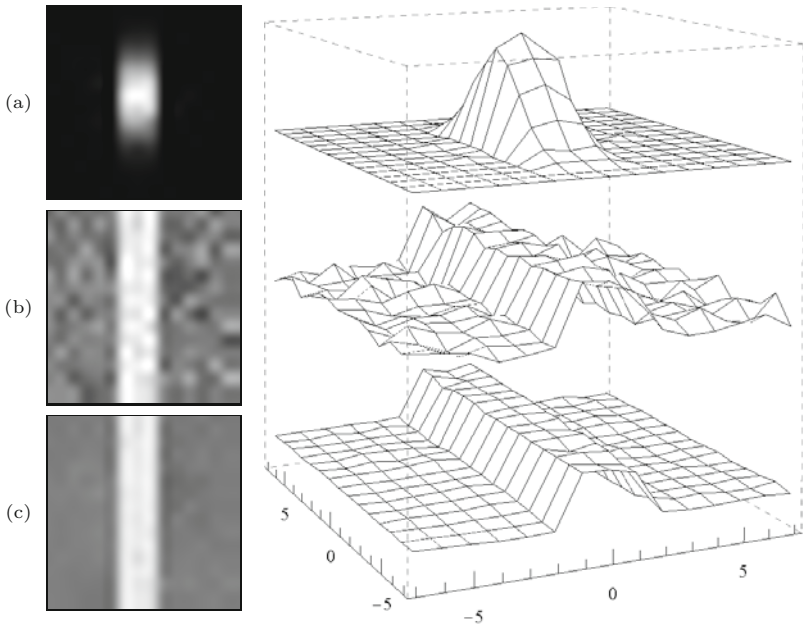

Fig. 17.10
Bilateral filter response when positioned on a vertical ridge. Original image function (b), filtered image (c), combined impulse response (a) of the filter at the given position.

grayscale images when using the same range kernel H_{r}.[7] Of course in most color spaces, none of these norms measures perceived color difference.[8] However, the distance function itself is not really critical since it only affects the relative *weights* assigned to the contributing

[7] For example, with 8-bit RGB color images, $\mathrm{dist}(\boldsymbol{a}, \boldsymbol{b})$ is always in the range $[0, 255]$.

[8] The CIELAB and CIELUV color spaces are designed to use the Euclidean distance (L_2 norm) as a valid metric for color difference.

Alg. 17.5 Bilateral filter with Gaussian kernels (color version). The function $\text{dist}(\boldsymbol{a}, \boldsymbol{b})$ measures the distance between two colors $\boldsymbol{a}$ and $\boldsymbol{b}$, for example, using the L_2 norm (line 5, see Eqns. (17.32)–(17.34) for other options).

BilateralFilterColor$(\boldsymbol{I}, \sigma_\text{d}, \sigma_\text{r})$
Input: $\boldsymbol{I}$, a color image of size $M \times N$; σ_d, width of the 2D Gaussian *domain* kernel; σ_r, width of the 1D Gaussian *range* kernel. Returns a new filtered color image of size $M \times N$.

2: $(M, N) \leftarrow \mathsf{Size}(I)$
3: $D \leftarrow \lceil 3.5 \cdot \sigma_\text{d} \rceil$ ▷ width of domain filter kernel
4: $\boldsymbol{I}' \leftarrow \mathsf{Duplicate}(\boldsymbol{I})$
5: $\text{dist}(\boldsymbol{a}, \boldsymbol{b}) := \frac{1}{\sqrt{3}} \cdot \|\boldsymbol{a} - \boldsymbol{b}\|_2$ ▷ color distance (e.g., Euclidean)
6: **for all** image coordinates $(u, v) \in (M \times N)$ **do**
7: $\boldsymbol{S} \leftarrow \boldsymbol{0}$ ▷ $\boldsymbol{S} \in \mathbb{R}^3$, sum of weighted pixel vectors
8: $W \leftarrow 0$ ▷ sum of pixel weights (scalar)
9: $\boldsymbol{a} \leftarrow \boldsymbol{I}(u, v)$ ▷ $\boldsymbol{a} \in \mathbb{R}^3$, center pixel vector
10: **for** $m \leftarrow -D, \ldots, D$ **do**
11: **for** $n \leftarrow -D, \ldots, D$ **do**
12: $\boldsymbol{b} \leftarrow \boldsymbol{I}(u + m, v + n)$ ▷ $\boldsymbol{b} \in \mathbb{R}^3$, off-center pixel vector
13: $w_\text{d} \leftarrow \exp\left(-\frac{m^2+n^2}{2\sigma_\text{d}^2}\right)$ ▷ domain coefficient
14: $w_\text{r} \leftarrow \exp\left(-\frac{(\text{dist}(\boldsymbol{a}, \boldsymbol{b}))^2}{2\sigma_\text{r}^2}\right)$ ▷ range coefficient
15: $w \leftarrow w_\text{d} \cdot w_\text{r}$ ▷ composite coefficient
16: $\boldsymbol{S} \leftarrow \boldsymbol{S} + w \cdot \boldsymbol{b}$
17: $W \leftarrow W + w$
18: $\boldsymbol{I}'(u, v) \leftarrow \frac{1}{W} \cdot \boldsymbol{S}$
19: **return** $\boldsymbol{I}'$

color pixels. Regardless of the distance function used, the resulting chromaticities are linear, convex combinations of the original colors in the filter region, and thus the choice of the working color space is more important (see Chapter 15, Sec. 15.1).

The process of bilateral filtering for color images (again using Gaussian kernels for the domain and the range filters) is summarized in Alg. 17.5. The Euclidean distance (L_2 norm) is used to measure the difference between color vectors. The examples in Fig. 17.11 were produced using sRGB as the color working space.

17.2.6 Efficient Implementation by x/y Separation

The bilateral filter, if implemented in the way described in Algs. 17.4–17.5, is computationally expensive, with a time complexity of $\mathcal{O}(K^2)$ for each pixel, where K denotes the radius of the filter. Some mild speedup is possible by tabulating the domain and range kernels, but the performance of the brute-force implementation is usually not acceptable for practical applications. In [185] a separable *approximation* of the bilateral filter is proposed that brings about a significant performance increase. In this implementation, a 1D bilateral filter is first applied in the horizontal direction only, which uses 1D domain and range kernels h_d and h_r, respectively, and produces the intermediate image $I^{\triangleright}$, that is,

(a) $\sigma_r = 10$

(b) $\sigma_r = 20$

(c) $\sigma_r = 50$

(d) $\sigma_r = 100$

Fig. 17.11
Bilateral filter—color example. A Gaussian kernel with $\sigma_d = 2.0$ (kernel size 15×15) is used for the domain part of the filter; working color space is sRGB. The width of the range filter is varied from $\sigma_r = 10$ to 100. The filter was applied in sRGB color space.

$$I^{\triangleright}(u,v) = \frac{\sum_{m=-D}^{D} I(u+m,v) \cdot h_d(m) \cdot h_r\big(I(u+m,v) - I(u,v)\big)}{\sum_{m=-D}^{D} h_d(m) \cdot h_r\big(I(u+m,v) - I(u,v)\big)}. \tag{17.35}$$

In the second pass, the *same* filter is applied to the intermediate result $I^{\triangleright}$ in the vertical direction to obtain the final result I' as

$$I'(u,v) = \frac{\sum_{n=-D}^{D} I^{\triangleright}(u,v+n) \cdot h_d(n) \cdot h_r\left(I^{\triangleright}(u,v+n) - I^{\triangleright}(u,v)\right)}{\sum_{n=-D}^{D} h_d(n) \cdot h_r\left(I^{\triangleright}(u,v+n) - I^{\triangleright}(u,v)\right)}, \tag{17.36}$$

for all (u,v), using the same 1D domain and range kernels h_d and h_r, respectively, as in Eqn. (17.35).

For the *horizontal* part of the filter, the effective space-variant kernel at image position (u, v) is

$$\bar{h}^{\triangleright}_{I,u,v}(i) = \frac{h_\mathrm{d}(i) \cdot h_\mathrm{r}\left(I(u+i,v) - I(u,v)\right)}{\sum_{m=-D}^{D} h_\mathrm{d}(i) \cdot h_\mathrm{r}\left(I(u+m,v) - I(u,v)\right)}, \tag{17.37}$$

for $-D \le i \le D$ (zero otherwise). Analogously, the effective kernel for the *vertical* part of the filter is

$$\bar{h}^{\triangledown}_{I,u,v}(j) = \frac{h_\mathrm{d}(i) \cdot h_\mathrm{r}\left(I(u,v+j) - I(u,v)\right)}{\sum_{n=-D}^{D} h_\mathrm{d}(j) \cdot h_\mathrm{r}\left(I(u,v+j) - I(u,v)\right)}, \tag{17.38}$$

again for $-D \le j \le D$. For the *combined* filter, the effective 2D kernel at position (u, v) then is

$$\bar{H}_{I,u,v}(i,j) = \begin{cases} \bar{h}^{\triangleright}_{I,u,v}(i) \cdot \bar{h}^{\triangledown}_{I^{\triangleright},u,v}(j) & \text{for } -D \le i,j \le D, \\ 0 & \text{otherwise,} \end{cases} \tag{17.39}$$

where I is the original image and $I^{\triangleright}$ denotes the intermediate image, as defined in Eqn. (17.35).

Alternatively, the vertical filter could be applied first, followed by the horizontal filter. Algorithm 17.6 shows a direct implementation of the separable bilateral filter for grayscale images, using Gaussian kernels for both the domain and the range parts of the filter. Again, the extension to color images is straightforward (see Eqn. (17.31) and Exercise 17.3).

As intended, the advantage of the separable filter is performance. For a given kernel radius D, the original (non-separable) requires $\mathcal{O}(D^2)$ calculations for each pixel, while the separable version takes only $\mathcal{O}(D)$ steps. This means a substantial saving and speed increase, particularly for large filters.

Figure 17.12 shows the response of the 1D separable bilateral filter in various situations. The results produced by the separable filter are very similar to those obtained with the original filter in Figs. 17.5–17.9, partly because the local structures in these images are parallel to the coordinate axes. In general, the results are different, as demonstrated for a diagonal step edge in Fig. 17.13. The effective filter kernels are shown in Fig. 17.13(g, h) for an anchor point positioned on the bright side of the edge. It can be seen that, while the kernel of the full filter Fig. 17.13(g) is orientation-insensitive, the upper part of the separable kernel is clearly truncated in Fig. 17.13(h). But although the separable bilateral filter is sensitive to local structure orientation, it performs well and is usually a sufficient substitute for the non-separable version [185]. The color examples shown in Fig. 17.14 demonstrate the effects of 1D bilateral filtering in the x- and y-directions. Note that the results are not exactly the same if the filter is first applied in the x- or in y-direction, but usually the differences are negligible.

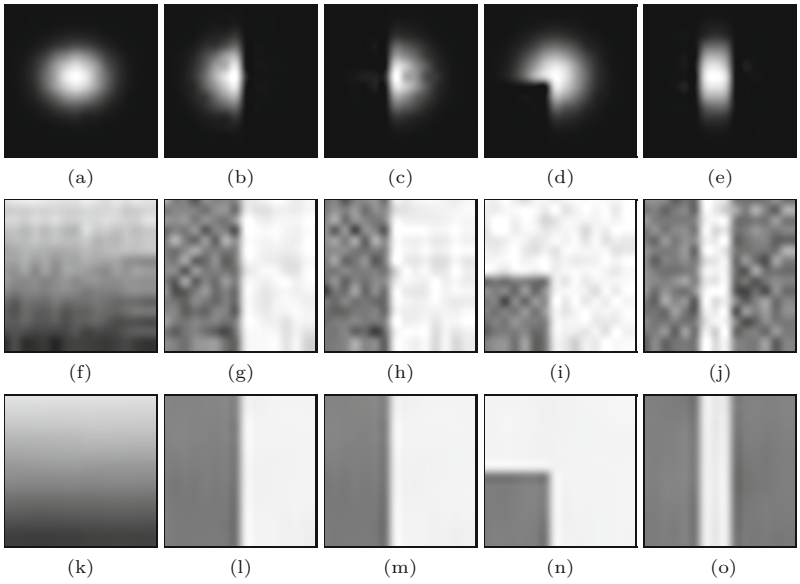

Fig. 17.12
Response of a *separable* bilateral filter in various situations. Effective kernel $\bar{H}_{I,u,v}$ (Eqn. (17.39)) at the center pixel (a–e), original image data (f–j), filtered image data (k–o). Settings are the same as in Figs. 17.5–17.9.

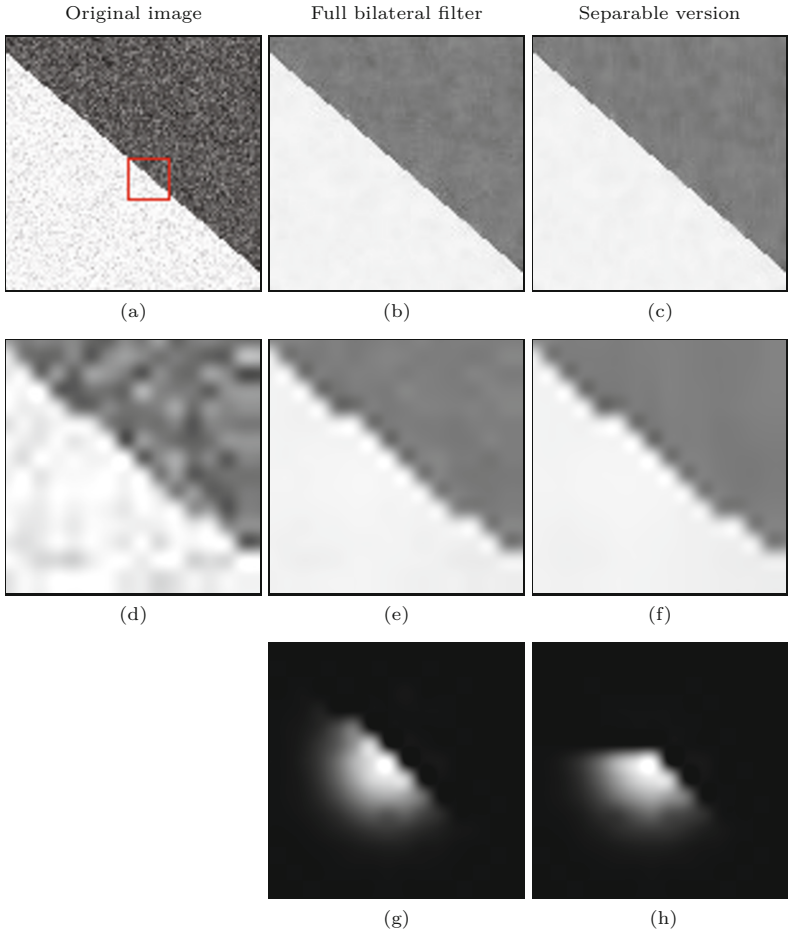

Fig. 17.13
Bilateral filter—full vs. separable version. Original image (a) and enlarged detail (d). Results of the full bilateral filter (b, e) and the separable version (c, f). The corresponding local filter kernels for the center pixel (positioned on the bright side of the step edge) for the full filter (g) and the separable version (h). Note how the upper part of the kernel in (h) is truncated along the horizontal axis, which shows that the separable filter is orientation-sensitive. In both cases, $\sigma_\mathrm{d} = 2.0$, $\sigma_\mathrm{r} = 25$.

Alg. 17.6 Separable bilateral filter with Gaussian kernels (adapted from Alg. 17.4). The input image is processed in two passes. In each pass, a 1D kernel is applied in horizontal or vertical direction, respectively (see Eqns. (17.35)–(17.36)). Note that results of the separable filter are similar (but not identical) to the full (2D) bilateral filter in Alg. 17.4.

BilateralFilterGraySeparable$(I, \sigma_\mathrm{d}, \sigma_\mathrm{r})$

Input: I, a grayscale image of size $M \times N$; σ_d, width of the 2D Gaussian *domain* kernel; σ_r, width of the 1D Gaussian *range* kernel. Returns a new filtered image of size $M \times N$.

$(M, N) \leftarrow \mathsf{Size}(I)$
$D \leftarrow \lceil 3.5 \cdot \sigma_\mathrm{d} \rceil$ ▷ width of domain filter kernel
$I^{\triangleright} \leftarrow \mathsf{Duplicate}(I)$
Pass 1 (horizontal):
for all coordinates $(u, v) \in M \times N$ **do**
  $a \leftarrow I(u, v)$
  $S \leftarrow 0, \quad W \leftarrow 0$
  for $m \leftarrow -D, \ldots, D$ **do**
    $b \leftarrow I(u + m, v)$
    $w_\mathrm{d} \leftarrow \exp\left(-\frac{m^2}{2\sigma_\mathrm{d}^2}\right)$ ▷ domain kernel coeff. $h_\mathrm{d}(m)$
    $w_\mathrm{r} \leftarrow \exp\left(-\frac{(a-b)^2}{2\sigma_\mathrm{r}^2}\right)$ ▷ range kernel coeff. $h_\mathrm{r}(b)$
    $w \leftarrow w_\mathrm{d} \cdot w_\mathrm{r}$ ▷ composite filter coeff.
    $S \leftarrow S + w \cdot b$
    $W \leftarrow W + w$
  $I^{\triangleright}(u, v) \leftarrow S/W$ ▷ see Eq. 17.35
$I' \leftarrow \mathsf{Duplicate}(I)$
Pass 2 (vertical):
for all coordinates $(u, v) \in M \times N$ **do**
  $a \leftarrow I^{\triangleright}(u, v)$
  $S \leftarrow 0, \quad W \leftarrow 0$
  for $n \leftarrow -D, \ldots, D$ **do**
    $b \leftarrow I^{\triangleright}(u, v + n)$
    $w_\mathrm{d} \leftarrow \exp\left(-\frac{n^2}{2\sigma_\mathrm{d}^2}\right)$ ▷ domain kernel coeff. $H_\mathrm{d}(n)$
    $w_\mathrm{r} \leftarrow \exp\left(-\frac{(a-b)^2}{2\sigma_\mathrm{r}^2}\right)$ ▷ range kernel coeff. $H_\mathrm{r}(b)$
    $w \leftarrow w_\mathrm{d} \cdot w_\mathrm{r}$ ▷ composite filter coeff.
    $S \leftarrow S + w \cdot b$
    $W \leftarrow W + w$
  $I'(u, v) \leftarrow S/W$ ▷ see Eq. 17.36
return I'

17.2.7 Further Reading

A thorough analysis of the bilateral filter as well as its relationship to adaptive smoothing and nonlinear diffusion can be found in [16] and [67]. In addition to the simple separable implementation described, several other fast versions of the bilateral filter have been proposed. For example, the method described in [65] approximates the bilateral filter by filtering sub-sampled copies of the image with discrete intensity kernels and recombining the results using linear interpolation. An improved and theoretically well-grounded version of this method was presented in [179]. The fast technique proposed in [253] eliminates the redundant calculations performed in partly overlapping image regions, albeit being restricted to the use of box-shaped domain kernels. As demonstrated in [187, 259], real-time performance using arbitrary-shaped kernels can be obtained by decomposing the filter into a set of smaller spatial filters.

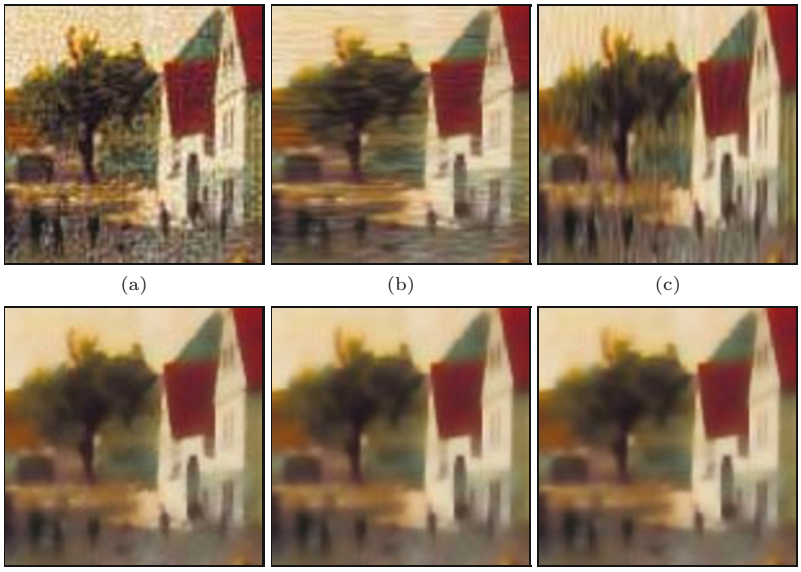

Fig. 17.14
Separable bilateral filter (color example). Original image (a), bilateral filter applied only in the x-direction (b) and only in the y-direction (c). Result of applying the *full* bilateral filter (d) and the *separable* bilateral filter applied in x/y order (e) and y/x order (f). Settings: $\sigma_d = 2.0$, $\sigma_r = 50$, L_2 color distance.

17.3 Anisotropic Diffusion Filters

Diffusion is a concept adopted from physics that models the spatial propagation of particles or state properties within substances. In the real world, certain physical properties (such as temperature) tend to diffuse homogeneously through a physical body, that is, equally in all directions. The idea viewing image smoothing as a diffusion process has a long history in image processing (see, e.g., [11,139]). To smooth an image and, at the same time, preserve edges or other "interesting" image structures, the diffusion process must somehow be made locally *non*-homogeneous; otherwise the entire image would come out equally blurred. Typically, the dominant smoothing direction is chosen to be *parallel* to nearby image contours, while smoothing is inhibited in the perpendicular direction, that is, *across* the contours.

Since the pioneering work by Perona and Malik [182], anisotropic diffusion has seen continued interest in the image processing community and research in this area is still strong today. The main elements of their approach are outlined in Sec. 17.3.2. While various other formulations have been proposed since, a key contribution by Weickert [250, 251] and Tschumperlé [233, 236] unified them into a common framework and demonstrated their extension to color images. They also proposed to separate the actual smoothing process from the smoothing geometry in order to obtain better control of the local smoothing behavior. In Sec. 17.3.4 we give a brief introduction to the approach proposed by Tschumperlé and Deriche, as initially described in [233]. Beyond these selected examples, a vast literature exists on this topic, including excellent reviews [95, 250], textbook material [125, 205], and journal articles (see [3, 45, 52, 173, 206, 226], for example).

17.3.1 Homogeneous Diffusion and the Heat Equation

Assume that in a homogeneous, 3D volume some physical property (e.g., temperature) is specified by a continuous function $f(\boldsymbol{x}, t)$ at position $\boldsymbol{x} = (x, y, z)$ and time t. With the system left to itself, the local differences in the property f will equalize over time until a global equilibrium is reached. This *diffusion process* in 3D space (x, y, z) and time (t) can be expressed using a partial differential equation (PDE),

$$\frac{\partial f}{\partial t} = c \cdot (\nabla^2 f) = c \cdot \Big(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}\Big). \tag{17.40}$$

This is the so-called *heat equation*, where $\nabla^2 f$ denotes the *Laplace operator*[9] applied to the scalar-valued function f, and c is a constant which describes the (thermal) *conductivity* or *conductivity coefficient* of the material. Since the conductivity is independent of position and orientation (c is constant), the resulting process is *isotropic*, that is, the heat spreads evenly in all directions.

For simplicity, we assume $c = 1$. Since f is a multi-dimensional function in space and time, we make this fact a bit more transparent by attaching explicit space and time coordinates $\boldsymbol{x}$ and τ to Eqn. (17.40), that is,

$$\frac{\partial f}{\partial t}(\boldsymbol{x}, \tau) = \frac{\partial^2 f}{\partial x^2}(\boldsymbol{x}, \tau) + \frac{\partial^2 f}{\partial y^2}(\boldsymbol{x}, \tau) + \frac{\partial^2 f}{\partial z^2}(\boldsymbol{x}, \tau), \tag{17.41}$$

or, written more compactly,

$$f_t(\boldsymbol{x}, \tau) = f_{\mathrm{xx}}(\boldsymbol{x}, \tau) \; + \; f_{\mathrm{yy}}(\boldsymbol{x}, \tau) \; + \; f_{zz}(\boldsymbol{x}, \tau). \tag{17.42}$$

Diffusion in images

A continuous, time-varying image I may be treated analogously to the function $f(\boldsymbol{x}, \tau)$, with the local intensities taking on the role of the temperature values in Eqn. (17.42). In this 2D case, the isotropic diffusion equation can be written as[10]

$$\frac{\partial I}{\partial t} = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \qquad \text{or} \tag{17.43}$$

$$I_t(\boldsymbol{x}, \tau) = I_{\mathrm{xx}}(\boldsymbol{x}, \tau) + I_{\mathrm{yy}}(\boldsymbol{x}, \tau), \tag{17.44}$$

with the derivatives $I_t = \partial I / \partial t$, $I_{\mathrm{xx}} = \partial^2 I / \partial x^2$, and $I_{\mathrm{yy}} = \partial^2 I / \partial y^2$. An approximate, numerical solution of such a PDE can be obtained by replacing the derivatives with finite differences.

Starting with the initial (typically noisy) image $I^{(0)} = I$, the solution to the differential equation in Eqn. (17.44) can be calculated iteratively in the form

[9] Remember that ∇f denotes the *gradient* of the function f, which is a vector for any multi-dimensional function. The Laplace operator (or *Laplacian*) $\nabla^2 f$ corresponds to the *divergence* of the *gradient* of f, denoted $\mathrm{div}\, \nabla f$, which is a scalar value (see Secs. C.2.5 and C.2.4 in the Appendix). Other notations for the Laplacian are $\nabla \cdot (\nabla f)$, $(\nabla \cdot \nabla) f$, $\nabla \cdot \nabla f$, $\nabla^2 f$, or Δf.

[10] Function arguments $(\boldsymbol{\xi}, \tau)$ are omitted here for better readability.

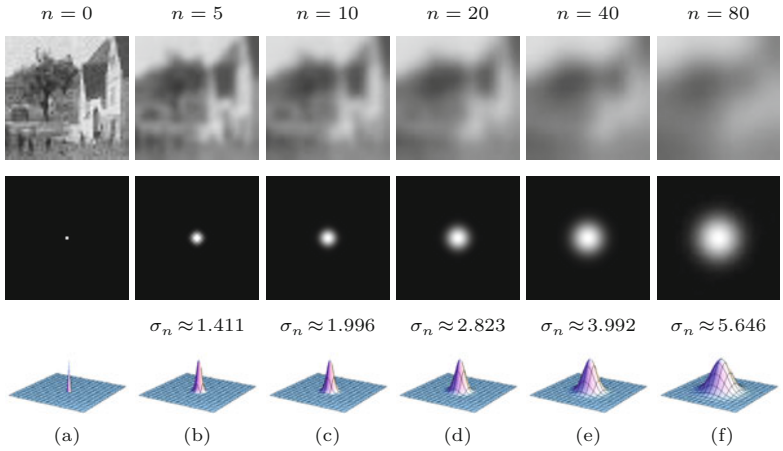

Fig. 17.15
Discrete isotropic diffusion. Blurred images and impulse response obtained after n iterations, with $\alpha = 0.20$ (see Eqn. (17.45)). The size of the images is 50×50. The width of the equivalent Gaussian kernel (σ_n) grows with the square root of n (the number of iterations). Impulse response plots are normalized to identical peak values.

$$I^{(n)}(\boldsymbol{u}) \leftarrow \begin{cases} I(\boldsymbol{u}) & \text{for } n = 0, \\ I^{(n-1)}(\boldsymbol{u}) + \alpha \cdot \left[\nabla^2 I^{(n-1)}(\boldsymbol{u})\right] & \text{for } n > 0, \end{cases} \tag{17.45}$$

for each image position $\boldsymbol{u} = (u, v)$, with n denoting the iteration number. This is called the "direct" solution method (there are other methods but this is the simplest). The constant α in Eqn. (17.45) is the time increment, which controls the speed of the diffusion process. Its value should be in the range $(0, 0.25]$ for the numerical scheme to be stable. At each iteration n, the variations in the image function are reduced and (depending on the boundary conditions) the image function should eventually flatten out to a constant plane as n approaches infinity.

For a discrete image I, the Laplacian $\nabla^2 I$ in Eqn. (17.45) can be approximated by a linear 2D filter,

$$\nabla^2 I \approx I * H^L, \qquad \text{with } H^L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{17.46}$$

as described earlier.[11] An essential property of isotropic diffusion is that it has the same effect as a Gaussian filter whose width grows with the elapsed time. For a discrete 2D image, in particular, the result obtained after n diffusion steps (Eqn. (17.45)), is the same as applying a linear filter to the original image I,

$$I^{(n)} \equiv I * H^{\mathrm{G},\sigma_n}, \tag{17.47}$$

with the normalized Gaussian kernel

$$H^{\mathrm{G},\sigma_n}(x, y) = \frac{1}{2\pi\sigma_n^2} \cdot e^{-\frac{x^2+y^2}{2\sigma_n^2}} \tag{17.48}$$

of width $\sigma_n = \sqrt{2t} = \sqrt{2n/\alpha}$. The example in Fig. 17.15 illustrates this Gaussian smoothing behavior obtained with discrete isotropic diffusion.

[11] See also Chapter 6, Sec. 6.6.1 and Sec. C.3.1 in the Appendix.

17.3.2 Perona-Malik Filter

Isotropic diffusion, as we have described, is a homogeneous operation that is independent of the underlying image content. Like any Gaussian filter, it effectively suppresses image noise but also tends to blur away sharp boundaries and detailed structures, a property that is often undesirable. The idea proposed in [182] is to make the conductivity coefficient *variable* and dependent on the local image structure. This is done by replacing the conductivity constant c in Eqn. (17.40), which can be written as

$$\frac{\partial I}{\partial t}(\boldsymbol{x},\tau) = c \cdot [\nabla^2 I](\boldsymbol{x},\tau), \tag{17.49}$$

by a *function* $c(\boldsymbol{x},t)$ that *varies* over space $\boldsymbol{x}$ and time t, that is,

$$\frac{\partial I}{\partial t}(\boldsymbol{x},\tau) = c(\boldsymbol{x},\tau) \cdot [\nabla^2 I](\boldsymbol{x},\tau). \tag{17.50}$$

If the conductivity function $c()$ is constant, then the equation reduces to the isotropic diffusion model in Eqn. (17.44).

Different behaviors can be implemented by selecting a particular function $c()$. To achieve edge-preserving smoothing, the conductivity $c()$ is chosen as a function of the magnitude of the local gradient vector ∇I, that is,

$$c(\boldsymbol{x},\tau) := g(d) = g\big(\|[\nabla I^{(\tau)}](\boldsymbol{x})\|\big). \tag{17.51}$$

To preserve edges, the function $g(d) : \mathbb{R} \to [0,1]$ should return high values in areas of low image gradient, enabling smoothing of homogeneous regions, but return low values (and thus inhibit smoothing) where the local brightness changes rapidly. Commonly used conductivity functions $g(d)$ are, for example [48, 182],

$$g_1(d) = e^{-(d/\kappa)^2}, \qquad g_2(d) = \frac{1}{1+(d/\kappa)^2}, \tag{17.52}$$

$$g_3(d) = \frac{1}{\sqrt{1+(d/\kappa)^2}}, \qquad g_4(d) = \begin{cases} (1-(d/2\kappa)^2)^2 & \text{for } d \le 2\kappa, \\ 0 & \text{otherwise,} \end{cases}$$

where $\kappa > 0$ is a constant that is either set manually (typically in the range $[5, 50]$ for 8-bit images) or adjusted to the amount of image noise. Graphs of the four functions in Eqn. (17.52) are shown in Fig. 17.16 for selected values of κ. The Gaussian conductivity function g_1 tends to promote high-contrast edges, whereas g_2 and even more g_3 prefer wide, flat regions over smaller ones. Function g_4, which corresponds to Tuckey's *biweight* function known from robust statistics [205, p. 230], is strictly zero for any argument $d > 2\kappa$. The exact shape of the function $g()$ does not appear to be critical; other functions with similar properties (e.g., with a linear cutoff) are sometimes used instead.

As an approximate discretization of Eqn. (17.50), Perona and Malik [182] proposed the simple iterative scheme

$$I^{(n)}(\boldsymbol{u}) \leftarrow I^{(n-1)}(\boldsymbol{u}) + \alpha \cdot \sum_{i=0}^{3} g\big(|\delta_i(I^{(n-1)},\boldsymbol{u})|\big) \cdot \delta_i(I^{(n-1)},\boldsymbol{u}), \tag{17.53}$$

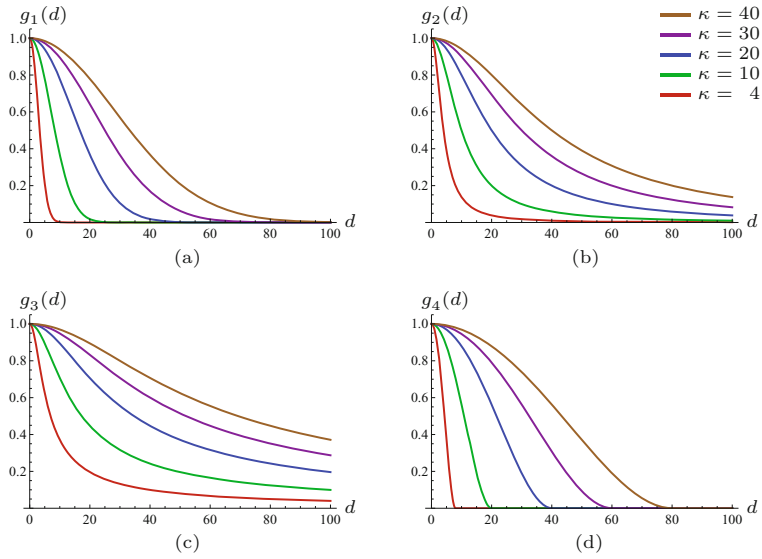

Fig. 17.16
Typical conductivity functions $g_1(), \ldots, g_4()$ for $\kappa = 4, 10, 20, 30, 40$ (see Eqn. (17.52)). If the magnitude of the local gradient d is small (near zero), smoothing amounts to a maximum (1.0), whereas diffusion is reduced where the gradient is high, for example, at or near edges. Smaller values of κ result in narrower curves, thereby restricting the smoothing operation to image areas with only small variations.

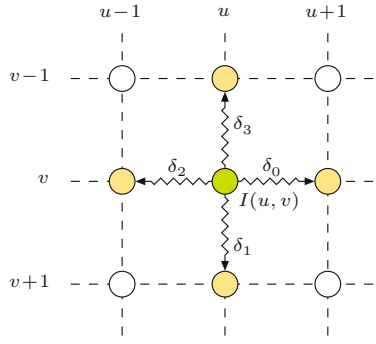

Fig. 17.17
Discrete lattice used for implementing diffusion filters in the Perona-Malik algorithm. The green element represents the current image pixel at position $\boldsymbol{u} = (u, v)$ and the yellow elements are its direct 4-neighbors.

where $I^{(0)} = I$ is the original image and

$$\delta_i(I, \boldsymbol{u}) = I(\boldsymbol{u} + \boldsymbol{d}_i) - I(\boldsymbol{u}) \tag{17.54}$$

denotes the difference between the pixel value $I(\boldsymbol{u})$ and its direct neighbor $i = 0, \ldots, 3$ (see Fig. 17.17), with

$$\boldsymbol{d}_0 = \binom{1}{0}, \quad \boldsymbol{d}_1 = \binom{0}{1}, \quad \boldsymbol{d}_2 = -\binom{1}{0}, \quad \boldsymbol{d}_3 = -\binom{0}{1}. \tag{17.55}$$

The procedure for computing the Perona-Malik filter for scalar-valued images is summarized in Alg. 17.7. The examples in Fig. 17.18 demonstrate how this filter performs along a step edge in a noisy grayscale image compared to isotropic (i.e., Gaussian) filtering.

In summary, the principle operation of this filter is to inhibit smoothing in the direction of strong local gradient vectors. Wherever the local contrast (and thus the gradient) is small, diffusion occurs uniformly in all directions, effectively implementing a Gaussian smoothing filter. However, in locations of high gradients, smoothing is inhibited along the gradient direction and allowed only in the direction perpendicular to it. If viewed as a heat diffusion process, a high-gradient brightness edge in an image acts like an insulating layer between areas of different temperatures. While temperatures

Alg. 17.7 Perona-Malik anisotropic diffusion filter for scalar (grayscale) images. The input image I is assumed to be real-valued (floating-point). Temporary real-valued maps $\mathsf{D}_x, \mathsf{D}_y$ are used to hold the directional gradient values, which are then re-calculated in every iteration. The conductivity function $g(d)$ can be one of the functions defined in Eqn. (17.52), or any similar function.

PeronaMalikGray(I, α, κ, T)

Input: I, a grayscale image of size $M \times N$; α, update rate; κ, smoothness parameter; T, number of iterations. Returns the modified image I.

Specify the conductivity function:

$g(d) := e^{-(d/\kappa)^2}$ ▷ for example, see alternatives in Eq. 17.52

$(M, N) \leftarrow \mathsf{Size}(I)$

Create maps $\mathsf{D}_x, \mathsf{D}_y$: $M \times N \to \mathbb{R}$

for $n \leftarrow 1, \ldots, T$ **do** ▷ perform T iterations

  for all coordinates $(u, v) \in M \times N$ **do** ▷ re-calculate gradients

$$\mathsf{D}_x(u,v) \leftarrow \begin{cases} I(u+1,v) - I(u,v) & \text{if } u < M-1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{D}_y(u,v) \leftarrow \begin{cases} I(u,v+1) - I(u,v) & \text{if } v < N-1 \\ 0 & \text{otherwise} \end{cases}$$

  for all coordinates $(u, v) \in M \times N$ **do** ▷ update the image

    $\delta_0 \leftarrow \mathsf{D}_x(u,v)$

    $\delta_1 \leftarrow \mathsf{D}_y(u,v)$

$$\delta_2 \leftarrow \begin{cases} -\mathsf{D}_x(u-1,v) & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_3 \leftarrow \begin{cases} -\mathsf{D}_y(u,v-1) & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$I(u,v) \leftarrow I(u,v) + \alpha \cdot \sum_{k=0}^{3} g(|\delta_k|) \cdot \delta_k$$

return I

continuously level out in the homogeneous regions on either side of an edge, thermal energy does not diffuse across the edge itself.

Note that the Perona-Malik filter (as defined in Eqn. (17.50)) is formally considered a *nonlinear* filter but not an *anisotropic* diffusion filter because the conductivity function $g()$ is only a scalar and not a (directed) vector-valued function [250]. However, the (inexact) discretization used in Eqn. (17.53), where each lattice direction is attenuated individually, makes the filter appear to perform in an anisotropic fashion.

17.3.3 Perona-Malik Filter for Color Images

The original Perona-Malik filter is not explicitly designed for color images or vector-valued images in general. The simplest way to apply this filter to a color image is (as usual) to treat the color channels as a set of independent scalar images and filter them separately. Edges should be preserved, since they occur only where at least one of the color channels exhibits a strong variation. However, different filters are applied to the color channels and thus new chromaticities may be produced that were not contained in the original image. Nevertheless, the results obtained (see the examples in Fig. 17.19(b–d)) are often satisfactory and the approach is frequently used because of its simplicity.

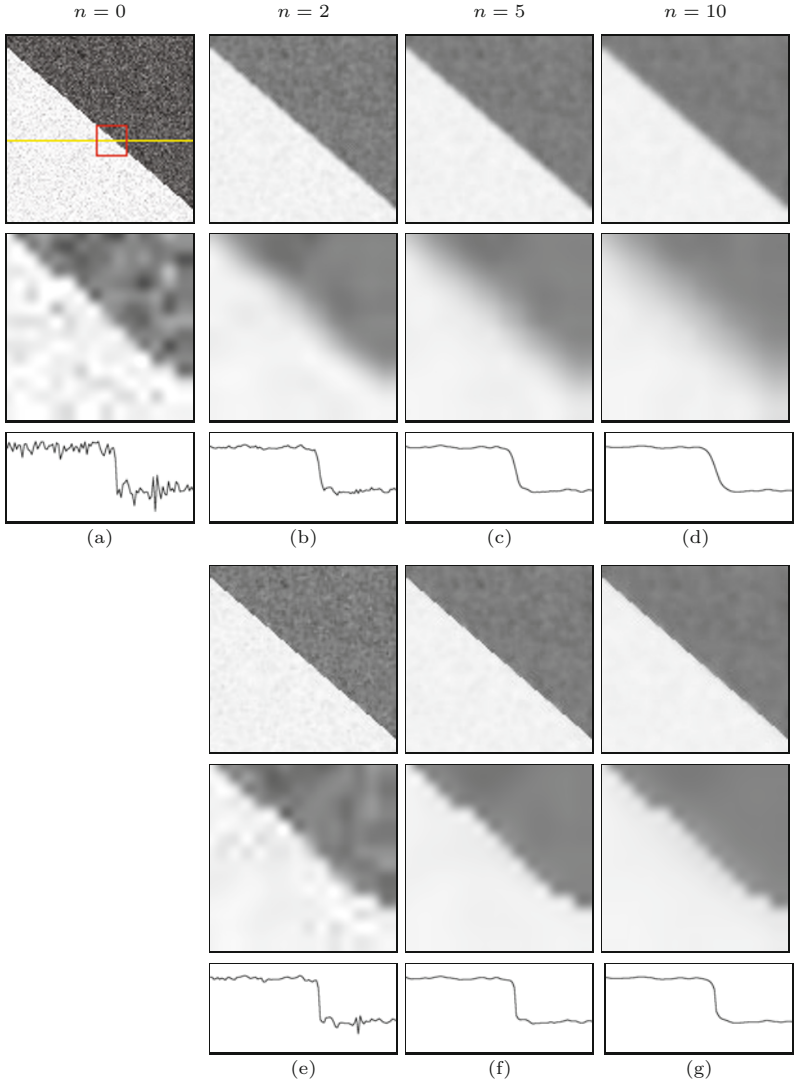

Fig. 17.18
Isotropic vs. anisotropic diffusion applied to a noisy step edge. Original image, enlarged detail, and horizontal profile (a), results of isotropic diffusion (b–d), results of anisotropic diffusion (e–g) after $n = 2, 5, 10$ iterations, respectively ($\alpha = 0.20$, $\kappa = 40$).

Color diffusion based on the brightness gradient

As an alternative to filtering each color channel separately, it has been proposed to use only the brightness (intensity) component to control the diffusion process of all color channels. Given an RGB color image $\boldsymbol{I} = (I_\mathrm{R}, I_\mathrm{G}, I_\mathrm{B})$ and a brightness function $\beta(\boldsymbol{I})$, the iterative scheme in Eqn. (17.53) could be modified to

$$\boldsymbol{I}^{(n)}(\boldsymbol{u}) \leftarrow \boldsymbol{I}^{(n-1)}(\boldsymbol{u}) + \alpha \cdot \sum_{i=0}^{3} g\big(|\beta_i(\boldsymbol{I}^{(n-1)}, \boldsymbol{u})|\big) \cdot \boldsymbol{\delta}_i(\boldsymbol{I}^{(n-1)}, \boldsymbol{u}), \tag{17.56}$$

where $$\beta_i(\boldsymbol{I}, \boldsymbol{u}) = \beta(\boldsymbol{I}(\boldsymbol{u} + \boldsymbol{d}_i)) - \beta(\boldsymbol{I}(\boldsymbol{u})), \tag{17.57}$$

$\boldsymbol{d}_i$ is the local brightness difference (as defined in Eqn. (17.55)) and

$$\boldsymbol{\delta}_i(\boldsymbol{I}, \boldsymbol{u}) = \begin{pmatrix} I_\mathrm{R}(\boldsymbol{u} + \boldsymbol{d}_i) - I_\mathrm{R}(\boldsymbol{u}) \\ I_\mathrm{G}(\boldsymbol{u} + \boldsymbol{d}_i) - I_\mathrm{G}(\boldsymbol{u}) \\ I_\mathrm{B}(\boldsymbol{u} + \boldsymbol{d}_i) - I_\mathrm{B}(\boldsymbol{u}) \end{pmatrix} = \begin{pmatrix} \delta_i(I_\mathrm{R}, \boldsymbol{u}) \\ \delta_i(I_\mathrm{G}, \boldsymbol{u}) \\ \delta_i(I_\mathrm{B}, \boldsymbol{u}) \end{pmatrix} \tag{17.58}$$

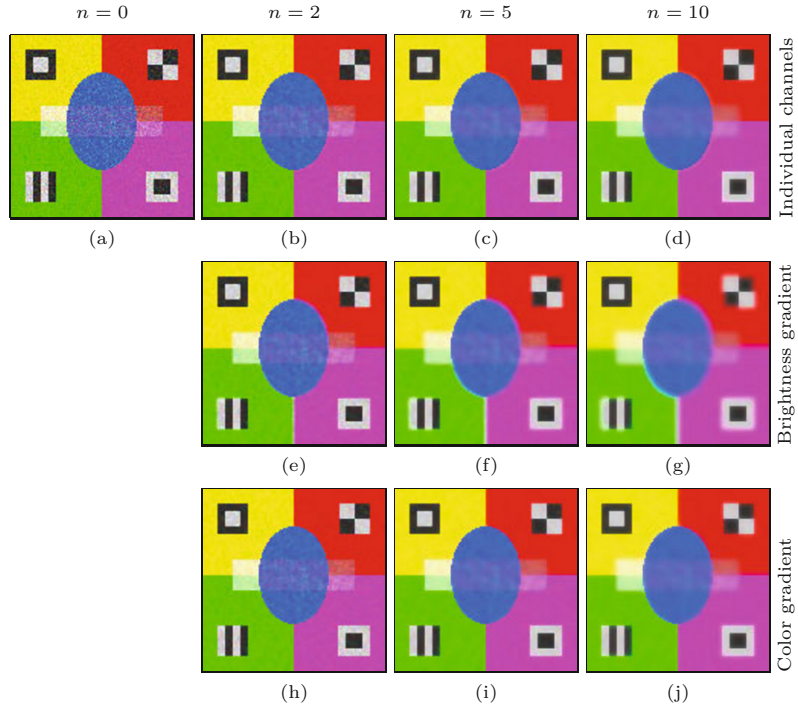

Fig. 17.19 Anisotropic diffusion filter (color). Noisy test image (a). Anisotropic diffusion filter applied separately to *individual color channels* (b–d), diffusion controlled by *brightness gradient* (e–g), diffusion controlled by *color gradient* (h–j), after 2, 5, and 10 iterations, respectively ($\alpha = 0.20$, $\kappa = 40$). With diffusion controlled by the brightness gradient, strong blurring occurs between regions of different color but similar brightness (e–g). The most consistent results are obtained by diffusion controlled by the *color gradient* (h–j). Filtering was performed in linear RGB color space.

is the local color difference vector for the neighboring pixels in directions $i = 0, \ldots, 3$ (see Fig. 17.17). Typical choices for the brightness function $\beta()$ are the *luminance* Y (calculated as a weighted sum of the linear R, G, B components), *luma* Y' (from nonlinear R', G', B' components), or the lightness component L of the CIELAB and CIELUV color spaces (see Chapter 15, Sec. 15.1 for a detailed discussion).

Algorithm 17.7 can be easily adapted to implement this type of color filter. An obvious disadvantage of this method is that it naturally blurs across color edges if the neighboring colors are of similar brightness, as the examples in Fig. 17.19(e–g)) demonstrate. This limits its usefulness for practical applications.

Using the color gradient

A better option for controlling the diffusion process in all three color channels is to use the color gradient (see Ch. 16, Sec. 16.2.1). As defined in Eqn. (16.17), the color gradient

$$(\mathrm{grad}_\theta \boldsymbol{I})(\boldsymbol{u}) = \boldsymbol{I}_x(\boldsymbol{u}) \cdot \cos(\theta) + \boldsymbol{I}_y(\boldsymbol{u}) \cdot \sin(\theta) \tag{17.59}$$

is a 3D vector, representing the combined variations of the color image $\boldsymbol{I}$ at position $\boldsymbol{u}$ in a given direction θ. The squared norm of this vector, $S_\theta(\boldsymbol{I}, \boldsymbol{u}) = \|(\mathrm{grad}_\theta \boldsymbol{I})(\boldsymbol{u})\|^2$, called the *squared local contrast*, is a scalar quantity useful for color edge detection. Along the horizontal and vertical directions of the discrete diffusion lattice (see Fig. 17.17), the angle θ is a multiple of $\pi/2$, and thus one of the cosine/sine terms in Eqn. (17.59) vanishes, that is,

$$\begin{aligned}\|(\mathrm{grad}_\theta \boldsymbol{I})(\boldsymbol{u})\| &= \|(\mathrm{grad}_{i\pi/2} \boldsymbol{I})(\boldsymbol{u})\| \\ &= \begin{cases} \|\boldsymbol{I}_x(\boldsymbol{u})\| & \text{for } i = 0, 2, \\ \|\boldsymbol{I}_y(\boldsymbol{u})\| & \text{for } i = 1, 3. \end{cases}\end{aligned} \tag{17.60}$$

Taking $\boldsymbol{\delta}_i$ (Eqn. (17.58)) as an estimate for the horizontal and vertical derivatives $\boldsymbol{I}_x$, $\boldsymbol{I}_y$, the diffusion iteration (adapted from Eqn. (17.53)) thus becomes

$$\boldsymbol{I}^{(n)}(\boldsymbol{u}) \leftarrow \boldsymbol{I}^{(n-1)}(\boldsymbol{u}) + \alpha \cdot \sum_{i=0}^{3} g\big(\|\boldsymbol{\delta}_i(\boldsymbol{I}^{(n-1)}, \boldsymbol{u})\|\big) \cdot \boldsymbol{\delta}_i(\boldsymbol{I}^{(n-1)}, \boldsymbol{u}), \tag{17.61}$$

with $g()$ chosen from one of the conductivity functions in Eqn. (17.52). Note that this is almost identical to the formulation in Eqn. (17.53), except for the use of vector-valued images and the absolute values $|\cdot|$ being replaced by the vector norm $\|\cdot\|$. The diffusion process is coupled between color channels, because the local diffusion strength depends on the combined color difference vectors. Thus, unlike in the brightness-governed diffusion scheme in Eqn. (17.56), opposing variations in different color do not cancel out and edges between colors of similar brightness are preserved (see the examples in Fig. 17.19(h–j)).

The resulting process is summarized in Alg. 17.8. The algorithm assumes that the components of the color image $\boldsymbol{I}$ are real-valued. In practice, integer-valued images must be converted to floating point before this procedure can be applied and integer results should be recovered by appropriate rounding.

Examples

Figure 17.20 shows the results of applying the Perona-Malik filter to a color image, using different modalities to control the diffusion process. In Fig. 17.20(a) the *scalar* (grayscale) diffusion filter (described in Alg. 17.7) is applied *separately* to each color channel. In Fig. 17.20(b) the diffusion process is coupled over all three color channels and controlled by the *brightness gradient*, as specified in Eqn. (17.56). Finally, in Fig. 17.20(c) the *color gradient* is used to control the common diffusion process, as defined in Eqn. (17.61) and Alg. 17.8. In each case, $T = 10$ diffusion iterations were applied, with update rate $\alpha = 0.20$, smoothness $\kappa = 25$, and conductivity function $g_1(d)$. The example demonstrates that, under otherwise equal conditions, edges and line structures are best preserved by the filter if the diffusion process is controlled by the color gradient.

17.3.4 Geometry Preserving Anisotropic Diffusion

Historically, the seminal publication by Perona and Malik [182] was followed by increased interest in the use of diffusion filters based on partial differential equations. Numerous different schemes were proposed, mainly with the aim to better adapt the diffusion process to the underlying image geometry.

Alg. 17.8 Anistropic diffusion filter for color images based on the color gradient (see Ch. 16, Sec. 16.2.1). The conductivity function $g(d)$ may be chosen from the functions defined in Eqn. (17.52), or any similar function. Note that (unlike in Alg. 17.7) the maps $\mathsf{D}_x, \mathsf{D}_y$ are vector-valued.

PeronaMalikColor$(\boldsymbol{I}, \alpha, \kappa, T)$

Input: $\boldsymbol{I}$, an RGB color image of size $M \times N$; α, update rate; κ, smoothness parameter; T, number of iterations. Returns the modified image $\boldsymbol{I}$.

Specify the conductivity function:

$g(d) := e^{-(d/\kappa)^2}$ ▷ for example, see alternatives in Eq. 17.52

$(M, N) \leftarrow \mathsf{Size}(\boldsymbol{I})$

Create maps $\mathsf{D}_x, \mathsf{D}_y$: $M \times N \to \mathbb{R}^3$; $\mathsf{S}_x, \mathsf{S}_y : M \times N \to \mathbb{R}$

for $n \leftarrow 1, \ldots, T$ **do** ▷ perform T iterations

  for all $(u, v) \in M \times N$ **do** ▷ re-calculate gradients

$$\mathsf{D}_x(u,v) \leftarrow \begin{cases} \boldsymbol{I}(u+1,v) - \boldsymbol{I}(u,v) & \text{if } u < M-1 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\mathsf{D}_y(u,v) \leftarrow \begin{cases} \boldsymbol{I}(u,v+1) - \boldsymbol{I}(u,v) & \text{if } v < N-1 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

    $\mathsf{S}_x(u,v) \leftarrow (\mathsf{D}_x(u,v))^2$ ▷ $= I_{R,x}^2 + I_{G,x}^2 + I_{B,x}^2$

    $\mathsf{S}_y(u,v) \leftarrow (\mathsf{D}_y(u,v))^2$ ▷ $= I_{R,y}^2 + I_{G,y}^2 + I_{B,y}^2$

  for all $(u, v) \in M \times N$ **do** ▷ update the image

    $s_0 \leftarrow \mathsf{S}_x(u,v)$, $\boldsymbol{\Delta}_0 \leftarrow \mathsf{D}_x(u,v)$

    $s_1 \leftarrow \mathsf{S}_y(u,v)$, $\boldsymbol{\Delta}_1 \leftarrow \mathsf{D}_y(u,v)$

    $s_2 \leftarrow 0$, $\boldsymbol{\Delta}_2 \leftarrow \mathbf{0}$

    $s_3 \leftarrow 0$, $\boldsymbol{\Delta}_3 \leftarrow \mathbf{0}$

    if $u > 0$ **then**

      $s_2 \leftarrow \mathsf{S}_x(u-1,v)$

      $\boldsymbol{\Delta}_2 \leftarrow -\mathsf{D}_x(u-1,v)$

    if $v > 0$ **then**

      $s_3 \leftarrow \mathsf{S}_y(u,v-1)$

      $\boldsymbol{\Delta}_3 \leftarrow -\mathsf{D}_y(u,v-1)$

    $\boldsymbol{I}(u,v) \leftarrow \boldsymbol{I}(u,v) + \alpha \cdot \sum_{k=0}^{3} g(|s_k|) \cdot \boldsymbol{\Delta}_k$

return $\boldsymbol{I}$

Generalized divergence-based formulation

Weickert [249, 250] generalized the divergence-based formulation of the Perona-Malik approach (see Eqn. (17.49)), that is,

$$\frac{\partial I}{\partial t} = \operatorname{div}\big(c \cdot \nabla I\big),$$

by replacing the time-varying, scalar diffusivity field $c(\boldsymbol{x}, \tau) \in \mathbb{R}$ by a *diffusion tensor* field $\boldsymbol{D}(\boldsymbol{x}, \tau) \in \mathbb{R}^{2\times 2}$ in the form

$$\frac{\partial I}{\partial t} = \operatorname{div}\big(\boldsymbol{D} \cdot \nabla I\big). \tag{17.62}$$

The time-varying tensor field $\boldsymbol{D}(\boldsymbol{x}, \tau)$ specifies a symmetric, positive-definite 2×2 matrix for each 2D image position $\boldsymbol{x}$ and time τ (i.e., $\boldsymbol{D} : \mathbb{R}^3 \to \mathbb{R}^{2\times 2}$ in the continuous case). Geometrically, $\boldsymbol{D}$ specifies an oriented, stretched ellipse which controls the local diffusion process. $\boldsymbol{D}$ may be independent of the image I but is typically derived from it. For example, the original Perona-Malik diffusion equation could be (trivially) written in the form[12]

[12] $\mathbf{I}_2$ denotes the 2×2 identity matrix.

(a) Color channels filtered separately

(b) Diffusion controlled by the local brightness gradient

(c) Diffusion controlled by the local color gradient

Fig. 17.20
Perona-Malik color example. Scalar diffusion filter applied separately to each color channel (a); diffusion controlled by the brightness gradient (b); diffusion controlled by color gradient (c). Common settings are $T = 10$, $\alpha = 0.20$, $g(d) = g_1(d)$, $\kappa = 25$; original image in Fig. 17.3(a).

$$\frac{\partial I}{\partial t} = \operatorname{div}\Big[\underbrace{(c \cdot \mathbf{I}_2)}_{\boldsymbol{D}} \cdot \nabla I\Big] = \operatorname{div}\Big[\begin{pmatrix} c & 0 \\ 0 & c \end{pmatrix} \cdot \nabla I\Big], \tag{17.63}$$

where $c = g\left(\|\nabla I(\boldsymbol{x}, t)\|\right)$ (see Eqn. (17.51)), and thus $\boldsymbol{D}$ is coupled to the image content. In Weickert's approach, $\boldsymbol{D}$ is constructed from the eigenvalues of the local "image structure tensor" [251], which we have encountered under different names in several places. This approach was also adapted to work with color images [252].

Trace-based formulation

Similar to the work of Weickert, the approach proposed by Tschumperlé and Deriche [233, 235] also pursues a geometry-oriented generalization of anisotropic diffusion. The approach is directly aimed at vector-valued (color) images, but can also be applied to single-channel (scalar-valued) images. For a vector-valued image $\boldsymbol{I} = (I_1, \ldots, I_n)$, the smoothing process is specified as

$$\frac{\partial I_k}{\partial t} = \operatorname{trace}\left(\boldsymbol{A} \cdot \mathbf{H}_k\right), \tag{17.64}$$

for each channel k, where $\mathbf{H}_k$ denotes the *Hessian* matrix of the scalar-valued image function of channel I_k, and $\boldsymbol{A}$ is a square (2×2 for 2D images) matrix that depends on the complete image $\boldsymbol{I}$ and

adapts the smoothing process to the local image geometry. Note that $\boldsymbol{A}$ is the same for all image channels. Since the trace of the Hessian matrix[13] is the Laplacian of the corresponding function (i.e., $\text{trace}(\mathbf{H}_I) = \nabla^2 I$) the diffusion equation for the Perona-Malik filter (Eqn. (17.49)) can be written as

$$\begin{aligned}\frac{\partial I}{\partial t} &= c \cdot (\nabla^2 I) = \text{div}(c \cdot \nabla I) \\ &= \text{trace}\left((c \cdot \mathbf{I}_2) \cdot \mathbf{H}_I\right) = \text{trace}\left(c \cdot \mathbf{H}_I\right).\end{aligned} \tag{17.65}$$

In this case, $\boldsymbol{A} = c \cdot \mathbf{I}_2$, which merely applies the constant scalar factor c to the Hessian matrix $\mathbf{H}_I$ (and thus to the resulting Laplacian) that is derived from the local image (since $c = g(\|\nabla I(\boldsymbol{x}, t)\|)$) and does not represent any geometric information.

17.3.5 Tschumperlé-Deriche Algorithm

This is different in the trace-based approach proposed by Tschumperlé and Deriche [233, 235], where the matrix $\boldsymbol{A}$ in Eqn. (17.64) is composed by the expression

$$\boldsymbol{A} = f_1(\lambda_1, \lambda_2) \cdot (\hat{\boldsymbol{q}}_2 \cdot \hat{\boldsymbol{q}}_2^\mathsf{T}) \; + \; f_2(\lambda_1, \lambda_2) \cdot (\hat{\boldsymbol{q}}_1 \cdot \hat{\boldsymbol{q}}_1^\mathsf{T}), \tag{17.66}$$

where λ_1, λ_2 and $\hat{\boldsymbol{q}}_1, \hat{\boldsymbol{q}}_2$ are the eigenvalues and normalized eigenvectors, respectively, of the (smoothed) 2×2 structure matrix

$$\boldsymbol{G} = \sum_{k=1}^{K} (\nabla I_k) \cdot (\nabla I_k)^\mathsf{T}, \tag{17.67}$$

with ∇I_k denoting the local gradient vector in image channel I_k. The functions $f_1(), f_2()$, which are defined in Eqn. (17.79), use the two eigenvalues to control the diffusion strength along the dominant direction of the contours (f_1) and perpendicular to it (f_2). Since the resulting algorithm is more involved than most previous ones, we describe it in more detail than usual.

Given a vector-valued image $\boldsymbol{I}\colon M \times N \to \mathbb{R}^n$, the following steps are performed in each iteration of the algorithm:

Step 1:

Calculate the gradient at each image position $\boldsymbol{u} = (u, v)$,

$$\nabla I_k(\boldsymbol{u}) = \begin{pmatrix} \frac{\partial I_k}{\partial x}(\boldsymbol{u}) \\ \frac{\partial I_k}{\partial y}(\boldsymbol{u}) \end{pmatrix} = \begin{pmatrix} I_{k,\mathrm{x}}(\boldsymbol{u}) \\ I_{k,\mathrm{y}}(\boldsymbol{u}) \end{pmatrix} = \begin{pmatrix} (I_k * H_x^\nabla)(\boldsymbol{u}) \\ (I_k * H_y^\nabla)(\boldsymbol{u}) \end{pmatrix}, \tag{17.68}$$

for each color channel $k = 1, \ldots, K$.[14] The first derivatives of the gradient vector ∇I_k are estimated by convolving the image with the kernels

[13] See Sec. C.2.6 in the Appendix for details.

[14] Note that $\nabla I_k(\boldsymbol{u})$ in Eqn. (17.68) is a 2D, vector-valued function, that is, a dedicated vector is calculated for every image position $\boldsymbol{u} = (u, v)$. For better readability, we omit the spatial coordinate $(\boldsymbol{u})$ in the following and simply write ∇I_k instead of $\nabla I_k(\boldsymbol{u})$. Analogously, all related vectors and matrices defined below (including the vectors $\mathbf{e}_1, \mathbf{e}_2$ and the matrices $\boldsymbol{G}$, $\bar{\boldsymbol{G}}$, $\boldsymbol{A}$, and $\mathbf{H}_k$) are also calculated for each image point $\boldsymbol{u}$, without the spatial coordinate being explicitly given.

$$H_x^{\nabla} = \begin{bmatrix} -a & 0 & a \\ -b & 0 & b \\ -a & 0 & a \end{bmatrix} \quad \text{and} \quad H_y^{\nabla} = \begin{bmatrix} -a & -b & -a \\ 0 & 0 & 0 \\ a & b & a \end{bmatrix}, \tag{17.69}$$

with $a = (2 - \sqrt{2})/4$ and $b = (\sqrt{2} - 1)/2$ (such that $2a + b = 1/2$).[15]

Step 2:

Smooth the channel gradients $I_{k,\mathrm{x}}, I_{k,\mathrm{y}}$ with an isotropic 2D Gaussian filter kernel $H^{\mathrm{G},\sigma_\mathrm{d}}$ of radius σ_d,

$$\overline{\nabla I}_k = \begin{pmatrix} \bar{I}_{k,\mathrm{x}} \\ \bar{I}_{k,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} I_{k,\mathrm{x}} * H^{\mathrm{G},\sigma_\mathrm{d}} \\ I_{k,\mathrm{y}} * H^{\mathrm{G},\sigma_\mathrm{d}} \end{pmatrix}, \tag{17.70}$$

for each image channel $k = 1, \ldots, K$. In practice, this step is usually skipped by setting $\sigma_\mathrm{d} = 0$.

Step 3:

Calculate the *Hessian matrix* (see Sec. C.2.6 in the Appendix) for each image channel I_k, $k = 1, \ldots, K$, that is,

$$\mathbf{H}_k = \begin{pmatrix} \frac{\partial^2 I_k}{\partial x^2} & \frac{\partial^2 I_k}{\partial x \partial y} \\ \frac{\partial^2 I_k}{\partial y \partial x} & \frac{\partial^2 I_k}{\partial y^2} \end{pmatrix} = \begin{pmatrix} I_{k,\mathrm{xx}} & I_{k,\mathrm{xy}} \\ I_{k,\mathrm{xy}} & I_{k,\mathrm{yy}} \end{pmatrix} = \begin{pmatrix} I_k * H_{\mathrm{xx}}^{\nabla} & I_k * H_{\mathrm{xy}}^{\nabla} \\ I_k * H_{\mathrm{xy}}^{\nabla} & I_k * H_{\mathrm{yy}}^{\nabla} \end{pmatrix}, \tag{17.71}$$

using the filter kernels

$$H_{\mathrm{xx}}^{\nabla} = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}, \quad H_{\mathrm{yy}}^{\nabla} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad H_{\mathrm{xy}}^{\nabla} = \frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}. \tag{17.72}$$

Step 4:

Calculate the local variation (structure) matrix as

$$\begin{aligned} \boldsymbol{G} &= \begin{pmatrix} G_0 & G_1 \\ G_1 & G_2 \end{pmatrix} = \sum_{k=1}^{K} (\overline{\nabla I}_k) \cdot (\overline{\nabla I}_k)^{\mathsf{T}} \qquad (17.73) \\ &= \sum_{k=1}^{K} \begin{pmatrix} \bar{I}_{k,\mathrm{x}}^2 & \bar{I}_{k,\mathrm{x}} \cdot \bar{I}_{k,\mathrm{y}} \\ \bar{I}_{k,\mathrm{x}} \cdot \bar{I}_{k,\mathrm{y}} & \bar{I}_{k,\mathrm{y}}^2 \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{K} \bar{I}_{k,\mathrm{x}}^2 & \sum_{k=1}^{K} \bar{I}_{k,\mathrm{x}} \cdot \bar{I}_{k,\mathrm{y}} \\ \sum_{k=1}^{K} \bar{I}_{k,\mathrm{x}} \cdot \bar{I}_{k,\mathrm{y}} & \sum_{k=1}^{K} \bar{I}_{k,\mathrm{y}}^2 \end{pmatrix}, \end{aligned}$$

for each image position $\boldsymbol{u}$. Note that the matrix $\boldsymbol{G}$ is symmetric (and positive semidefinite). In particular, for a RGB color image this is (coordinates $\boldsymbol{u}$ again omitted)

$$\begin{aligned} \boldsymbol{G} &= \begin{pmatrix} \bar{I}_{\mathrm{R},x}^2 & \bar{I}_{\mathrm{R},x} \bar{I}_{\mathrm{R},y} \\ \bar{I}_{\mathrm{R},x} \bar{I}_{\mathrm{R},y} & \bar{I}_{\mathrm{R},y}^2 \end{pmatrix} + \begin{pmatrix} \bar{I}_{\mathrm{G},x}^2 & \bar{I}_{\mathrm{G},x} \bar{I}_{\mathrm{G},y} \\ \bar{I}_{\mathrm{G},x} \bar{I}_{\mathrm{G},y} & \bar{I}_{\mathrm{G},y}^2 \end{pmatrix} + \begin{pmatrix} \bar{I}_{\mathrm{B},x}^2 & \bar{I}_{\mathrm{B},x} \bar{I}_{\mathrm{B},y} \\ \bar{I}_{\mathrm{B},x} \bar{I}_{\mathrm{B},y} & \bar{I}_{\mathrm{B},x}^2 \end{pmatrix} \\ &= \begin{pmatrix} \bar{I}_{\mathrm{R},x}^2 + \bar{I}_{\mathrm{G},x}^2 + \bar{I}_{\mathrm{B},x}^2 & \bar{I}_{\mathrm{R},x} \bar{I}_{\mathrm{R},y} + \bar{I}_{\mathrm{G},x} \bar{I}_{\mathrm{G},y} + \bar{I}_{\mathrm{B},x} \bar{I}_{\mathrm{B},y} \\ \bar{I}_{\mathrm{R},x} \bar{I}_{\mathrm{R},y} + \bar{I}_{\mathrm{G},x} \bar{I}_{\mathrm{G},y} + \bar{I}_{\mathrm{B},x} \bar{I}_{\mathrm{B},y} & \bar{I}_{\mathrm{R},y}^2 + \bar{I}_{\mathrm{G},y}^2 + \bar{I}_{\mathrm{B},y}^2 \end{pmatrix}. \end{aligned} \tag{17.74}$$

[15] Any other common set of x/y gradient kernels (e.g., Sobel masks) could be used instead, but these filters have better rotation invariance than their traditional counterparts. Similar kernels (with $a = 3/32$, $b = 10/32$) were proposed by Jähne in [126, p. 353].

Step 5:

Smooth the elements of the structure matrix $\boldsymbol{G}$ using an isotropic Gaussian filter kernel $H^{\mathrm{G},\sigma_\mathrm{g}}$ of radius σ_g, that is,

$$\bar{\boldsymbol{G}} = \begin{pmatrix} \bar{G}_0 & \bar{G}_1 \\ \bar{G}_1 & \bar{G}_2 \end{pmatrix} = \begin{pmatrix} G_0 * H^{\mathrm{G},\sigma_\mathrm{g}} & G_1 * H^{\mathrm{G},\sigma_\mathrm{g}} \\ G_1 * H^{\mathrm{G},\sigma_\mathrm{g}} & G_2 * H^{\mathrm{G},\sigma_\mathrm{g}} \end{pmatrix}. \tag{17.75}$$

Step 6:

For each image position $\boldsymbol{u}$, calculate the eigenvalues λ_1, λ_2 for the smoothed 2×2 matrix $\bar{\boldsymbol{G}}$, such that $\lambda_1 \geq \lambda_2$, and the corresponding normalized eigenvectors[16]

$$\hat{\boldsymbol{q}}_1 = \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix}, \quad \hat{\boldsymbol{q}}_2 = \begin{pmatrix} \hat{x}_2 \\ \hat{y}_2 \end{pmatrix},$$

such that $\|\hat{\boldsymbol{q}}_1\| = \|\hat{\boldsymbol{q}}_2\| = 1$. Note that $\hat{\boldsymbol{q}}_1$ points in the direction of maximum change and $\hat{\boldsymbol{q}}_2$ points in the perpendicular direction, that is, along the edge tangent. Thus, smoothing should occur predominantly along $\hat{\boldsymbol{q}}_2$. Since $\hat{\boldsymbol{q}}_1$ and $\hat{\boldsymbol{q}}_2$ are normal to each other, we can express $\hat{\boldsymbol{q}}_2$ in terms of $\hat{\boldsymbol{q}}_1$, for example,

$$\hat{\boldsymbol{q}}_2 \;\equiv\; \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \hat{\boldsymbol{q}}_1 = \begin{pmatrix} -\hat{y}_1 \\ \hat{x}_1 \end{pmatrix}. \tag{17.76}$$

Step 7:

From the eigenvalues (λ_1, λ_2) and the normalized eigenvectors $(\hat{\boldsymbol{q}}_1, \hat{\boldsymbol{q}}_2)$ of $\bar{\boldsymbol{G}}$, compose the symmetric matrix $\boldsymbol{A}$ in the form

$$\begin{aligned} \boldsymbol{A} = \begin{pmatrix} A_0 & A_1 \\ A_1 & A_2 \end{pmatrix} &= \underbrace{f_1(\lambda_1, \lambda_2)}_{c_1} \cdot (\hat{\boldsymbol{q}}_2 \cdot \hat{\boldsymbol{q}}_2^\mathsf{T}) \;+\; \underbrace{f_2(\lambda_1, \lambda_2)}_{c_2} \cdot (\hat{\boldsymbol{q}}_1 \cdot \hat{\boldsymbol{q}}_1^\mathsf{T}) \\ &= c_1 \cdot \begin{pmatrix} \hat{y}_1^2 & -\hat{x}_1 \cdot \hat{y}_1 \\ -\hat{x}_1 \cdot \hat{y}_1 & \hat{x}_1^2 \end{pmatrix} + c_2 \cdot \begin{pmatrix} \hat{x}_1^2 & \hat{x}_1 \cdot \hat{y}_1 \\ \hat{x}_1 \cdot \hat{y}_1 & \hat{y}_1^2 \end{pmatrix} \qquad (17.77) \\ &= \begin{pmatrix} c_1 \cdot \hat{y}_1^2 + c_2 \cdot \hat{x}_1^2 & (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 \\ (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 & c_1 \cdot \hat{x}_1^2 + c_2 \cdot \hat{y}_1^2 \end{pmatrix}, \qquad (17.78) \end{aligned}$$

using the conductivity coefficients

$$\begin{aligned} c_1 &= f_1(\lambda_1, \lambda_2) = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_1}}, \\ c_2 &= f_2(\lambda_1, \lambda_2) = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_2}}, \end{aligned} \tag{17.79}$$

with fixed parameters $a_1, a_2 > 0$ to control the non-isotropy of the filter: a_1 specifies the amount of smoothing along contours, a_2 in perpendicular direction (along the gradient). Small values of a_1, a_2 facilitate diffusion in the corresponding direction, while larger values inhibit smoothing. With a_1 close to zero, diffusion is practically unconstrained along the tangent direction. Typical default values are $a_1 = 0.5$ and $a_2 = 0.9$; results from other settings are shown in the examples.

[16] See Sec. B.4.1 in the Appendix for details on calculating the eigensystem of a 2×2 matrix.

Step 8:

Finally, each image channel I_k is updated using the recurrence relation

$$I_k \leftarrow I_k + \alpha \cdot \mathrm{trace}\,(\boldsymbol{A}\cdot\mathbf{H}_k) = I_k + \alpha \cdot \beta_k \tag{17.80}$$

$$= I_k + \alpha \cdot \big(A_0 \cdot I_{k,\mathrm{xx}} + A_1 \cdot I_{k,\mathrm{xy}} + A_1 \cdot I_{k,yx} + A_2 \cdot I_{k,\mathrm{yy}}\big) \tag{17.81}$$

$$= I_k + \alpha \cdot \underbrace{\big(A_0 \cdot I_{k,\mathrm{xx}} + 2 \cdot A_1 \cdot I_{k,\mathrm{xy}} + A_2 \cdot I_{k,\mathrm{yy}}\big)}_{\beta_k} \tag{17.82}$$

(since $I_{k,\mathrm{xy}} = I_{k,\mathrm{yx}}$). The term $\beta_k = \mathrm{trace}\,(\boldsymbol{A}\cdot\mathbf{H}_k)$ represents the local image *velocity* in channel k. Note that, although a separate Hessian matrix $\mathbf{H}_k$ is calculated for each channel, the structure matrix $\boldsymbol{A}$ is the same for all image channels. The image is thus smoothed along a common image geometry which considers the correlation between color channels, since $\boldsymbol{A}$ is derived from the joint structure matrix $\boldsymbol{G}$ (Eqn. (17.74)) and therefore combines all K color channels.

In each iteration, the factor α in Eqn. (17.82) is adjusted dynamically to the maximum current velocity β_k in all channels in the form

$$\alpha = \frac{d_\mathrm{t}}{\max \beta_k} = \frac{d_\mathrm{t}}{\max\limits_{k,\boldsymbol{u}} |\mathrm{trace}(\boldsymbol{A}\cdot\mathbf{H}_k)|}\,, \tag{17.83}$$

where d_t is the (constant) "time increment" parameter. Thus the time step α is kept small as long as the image gradients (vector field velocities) are large. As smoothing proceeds, image gradients are reduced and thus α typically increases over time. In the actual implementation, the values of I_k (in Eqn. (17.82)) are hard-limited to the initial minimum and maximum.

The steps (1–8) we have just outlined are repeated for the specified number of iterations. The complete procedure is summarized in Alg. 17.9 and a corresponding Java implementation can be found on the book's website (see Sec. 17.4).

Beyond this baseline algorithm, several variations and extensions of this filter exist, including the use of spatially-adaptive, oriented smoothing filters.[17] This type of filter has also been used with good results for *image inpainting* [234], where diffusion is applied to fill out only selected (masked) parts of the image where the content is unknown or should be removed.

Examples

The example in Fig. 17.21 demonstrates the influence of image geometry and how the non-isotropy of the Tschumperlé-Deriche filter can be controlled by varying the diffusion parameters a_1, a_2 (see Eqn. (17.79)). Parameter a_1, which specifies the diffusion in the direction of contours, is changed while a_2 (controlling the diffusion in the gradient direction) is held constant. In Fig. 17.21(a), smoothing along contours is modest and very small across edges with the default settings $a_1 = 0.5$ and $a_2 = 0.9$. With lower values of a_1, increased

[17] A recent version was released by the original authors as part of the "GREYC's Magic Image Converter" open-source framework, which is also available as a GIMP plugin (http://gmic.sourceforge.net).

Alg. 17.9 Tschumperlé-Deriche anisotropic diffusion filter for vector-valued (color) images. Typical settings are $T = 5, \ldots, 20$, $d_t = 20$, $\sigma_g = 0$, $\sigma_s = 0.5$, $a_1 = 0.5$, $a_2 = 0.9$. See Sec. B.4.1 for a description of the procedure RealEigenValues2x2 (used in line 12).

TschumperleDericheFilter$(\boldsymbol{I}, T, d_t, \sigma_g, \sigma_s, a_1, a_2)$

Input: $\boldsymbol{I} = (I_1, \ldots, I_K)$, color image of size $M \times N$ with K channels; T, number of iterations; d_t, time increment; σ_g, width of the Gaussian kernel for smoothing the gradient; σ_s, width of the Gaussian kernel for smoothing the structure matrix; a_1, a_2, diffusion parameters for directions of min./max. variation, respectively. Returns the modified image $\boldsymbol{I}$.

2: Create maps:

$\mathsf{D} : K \times M \times N \to \mathbb{R}^2$ ▷ $\mathsf{D}(k,u,v) \equiv \nabla I_k(u,v)$, grad. vector

$\mathsf{H} : K \times M \times N \to \mathbb{R}^{2\times 2}$ ▷ $\mathsf{H}(k,u,v) \equiv \mathbf{H}_k(u,v)$, Hess. matrix

$\mathsf{G} : M \times N \to \mathbb{R}^{2\times 2}$ ▷ $\mathsf{G}(u,v) \equiv \boldsymbol{G}(u,v)$, structure matrix

$\mathsf{A} : M \times N \to \mathbb{R}^{2\times 2}$ ▷ $\mathsf{A}(u,v) \equiv \boldsymbol{A}(u,v)$, geometry matrix

$\mathsf{B} : K \times M \times N \to \mathbb{R}$ ▷ $\mathsf{B}(k,u,v) \equiv \beta_k(u,v)$, velocity

3: **for** $t \leftarrow 1, \ldots, T$ **do** ▷ perform T iterations

4: **for** $k \leftarrow 1, \ldots, K$ and **all** coordinates $(u,v) \in M \times N$ **do**

5: $\mathsf{D}(k,u,v) \leftarrow \begin{pmatrix} (I_k * H_x^{\nabla})(u,v) \\ (I_k * H_y^{\nabla})(u,v) \end{pmatrix}$ ▷ Eq. 17.68–17.69

6: $\mathsf{H}(k,u,v) \leftarrow \begin{pmatrix} (I_k * H_{xx}^{\nabla})(u,v) & (I_k * H_{xy}^{\nabla})(u,v) \\ (I_k * H_{xy}^{\nabla})(u,v) & (I_k * H_{yy}^{\nabla})(u,v) \end{pmatrix}$ ▷ Eq. 17.71–17.72

7: $\mathsf{D} \leftarrow \mathsf{D} * H_G^{\sigma_d}$ ▷ smooth elements of D over (u,v)

8: **for all** coordinates $(u,v) \in M \times N$ **do**

9: $\mathsf{G}(u,v) \leftarrow \sum_{k=1}^{K} \begin{pmatrix} (\mathsf{D}_x(k,u,v))^2 & \mathsf{D}_x(k,u,v) \cdot \mathsf{D}_y(k,u,v) \\ \mathsf{D}_x(k,u,v) \cdot \mathsf{D}_y(k,u,v) & (\mathsf{D}_y(k,u,v))^2 \end{pmatrix}$

10: $\mathsf{G} \leftarrow \mathsf{G} * H_G^{\sigma_g}$ ▷ smooth elements of G over (u,v)

11: **for all** coordinates $(u,v) \in M \times N$ **do**

12: $(\lambda_1, \lambda_2, \boldsymbol{q}_1, \boldsymbol{q}_2) \leftarrow \mathsf{RealEigenValues2x2}(\mathsf{G}(u,v))$ ▷ p. 724

13: $\hat{\boldsymbol{q}}_1 \leftarrow \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix} = \frac{\boldsymbol{q}_1}{\|\boldsymbol{q}_1\|}$ ▷ normalize 1st eigenvector ($\lambda_1 \geq \lambda_2$)

14: $c_1 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_1}}, \quad c_2 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_2}}$ ▷ Eq. 17.79

15: $\mathsf{A}(u,v) \leftarrow \begin{pmatrix} c_1 \cdot \hat{y}_1^2 + c_2 \cdot \hat{x}_1^2 & (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 \\ (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 & c_1 \cdot \hat{x}_1^2 + c_2 \cdot \hat{y}_1^2 \end{pmatrix}$ ▷ Eq. 17.78

16: $\beta_{\max} \leftarrow -\infty$

17: **for** $k \leftarrow 1, \ldots, K$ and **all** $(u,v) \in M \times N$ **do**

18: $\mathsf{B}(k,u,v) \leftarrow \mathrm{trace}(\mathsf{A}(u,v) \cdot \mathsf{H}(k,u,v))$ ▷ β_k, Eq. 17.82

19: $\beta_{\max} \leftarrow \max(\beta_{\max}, |\mathsf{B}(k,u,v)|)$

20: $\alpha \leftarrow d_t / \beta_{\max}$ ▷ Eq. 17.83

21: **for** $k \leftarrow 1, \ldots, K$ and **all** $(u,v) \in M \times N$ **do**

22: $I_k(u,v) \leftarrow I_k(u,v) + \alpha \cdot \mathsf{B}(k,u,v)$ ▷ update the image

23: **return** $\boldsymbol{I}$

blurring occurs in the direction of the contours, as shown in Figs. 17.21(b, c).

17.4 Java Implementation

Implementations of the filters described in this chapter are available as part of the `imagingbook`[18] library at the book's website. The associated classes `KuwaharaFilter`, `NagaoMatsuyamaFilter`, `PeronaMalikFilter` and `TschumperleDericheFilter` are based on

[18] Package `imagingbook.pub.edgepreservingfilters`.

(a) $a_1 = 0.50$

(b) $a_1 = 0.25$

(c) $a_1 = 0.00$

Fig. 17.21
Tschumperlé-Deriche filter example. The non-isotropy of the filter can be adjusted by changing parameter a_1, which controls the diffusion along contours (see Eqn. (17.79)): $a_1 = 0.50, 0.25, 0.00$ (a–c). Parameter $a_2 = 0.90$ (constant) controls the diffusion in the direction of the gradient (perpendicular to contours). Remaining settings are $T = 20$, $d_t = 20$, $\sigma_g = 0.5$, $\sigma_s = 0.5$ (see the description of Alg. 17.9); original image in Fig. 17.3(a).

the common super-class `GenericFilter`[19] and define the following constructors:

`KuwaharaFilter (Parameters p)`
: Creates a Kuwahara-type filter for grayscale and color images, as described in Sec. 17.1 (Alg. 17.2), with radius `r` (default 2) and variance threshold `tsigma` (denoted t_σ in Alg. 17.2, default 0.0). The size of the resulting filter is $(2r + 1) \times (2r + 1)$.

`BilateralFilter (Parameters p)`
: Creates a bilateral filter for grayscale and color images using Gaussian kernels, as described in Sec. 17.2 (seeAlgs. 17.4 and 17.5). Parameters `sigmaD` (σ_d, default 2.0) and `sigmaR` (σ_r, default 50.0) specify the widths of the domain and the range kernels, respectively. The type of norm for measuring color distances is specified by `colorNormType` (default is `NormType.L2`).

`BilateralFilterSeparable (Parameters p)`
: Creates a x/y-separable bilateral filter for grayscale and color images, (see Alg. 17.6). Constructor parameters are the same as for the class `BilateralFilter` above.

[19] Package `imagingbook.lib.filters`. Filters of this type can be applied to images using the method `applyTo(ImageProcessor ip)`, as described in Chapter 15, Sec. 15.3.

`PeronaMalikFilter(Parameters p)`
: Creates an anisotropic diffusion filter for grayscale and color images (see Algs. 17.7 and 17.8). The key parameters and their default values are `iterations` ($T = 10$), `alpha` ($\alpha = 0.2$), `kappa` ($\kappa = 25$), `smoothRegions` (`true`), `colorMode` (`SeparateChannels`). With `smoothRegions` = `true`, function $g_\kappa^{(2)}$ is used to control conductivity, otherwise $g_\kappa^{(1)}$ (see Eqn. (17.52)). For filtering color images, three different color modes can be specified for diffusion control: `SeparateChannels`, `BrightnessGradient`, or `ColorGradient`. See Prog. 17.1 for an example of using this class in a simple ImageJ plugin.

`TschumperleDericheFilter(Parameters p)`
: Creates an anisotropic diffusion filter for color images, as described in Sec. 17.3.4 (Alg. 17.9). Parameters and default values are `iterations` ($T = 20$), `dt` ($d_t = 20$), `sigmaG` ($\sigma_g = 0.0$), `sigmaS` ($\sigma_s = 0.5$), `a1` ($a_1 = 0.25$), `a2` ($a_2 = 0.90$). Otherwise the usage of this class is analogous to the example in Prog. 17.1.

All default values pertain to the parameterless constructors that are also available. Note that these filters are generic and can be applied to grayscale and color images without any modification.

17.5 Exercises

Exercise 17.1. Implement a pure *range filter* (Eqn. (17.17)) for grayscale images, using a 1D Gaussian kernel

$$H_r(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot\exp(-\frac{x^2}{2\sigma^2}).$$

Investigate the effects of this filter upon the image and its histogram for $\sigma = 10$, 20, and 25.

Exercise 17.2. Modify the Kuwahara-type filter for color images in Alg. 17.3 to use the *norm of the color covariance matrix* (as defined in Eqn. (17.12)) for quantifying the amount of variation in each subregion. Estimate the number of additional calculations required for processing each image pixel. Implement the modified algorithm, compare the results and execution times.

Exercise 17.3. Modify the separable bilateral filter algorithm (given in Alg. 17.6) to handle color images, using Alg. 17.5 as a starting point. Implement and test your algorithm, compare the results (see also Fig. 17.14) and execution times.

Exercise 17.4. Verify (experimentally) that n iterations of the diffusion process defined in Eqn. (17.45) have the same effect as a Gaussian filter of width σ_n, as stated in Eqn. (17.48). To determine the impulse response of the resulting diffusion filter, use an "impulse" test image, that is, a black (zero-valued) image with a single bright pixel at the center.

```
1  import ij.ImagePlus;
2  import ij.plugin.filter.PlugInFilter;
3  import ij.process.ImageProcessor;
4  import imagingbook...PeronaMalikFilter;
5  import imagingbook...PeronaMalikFilter.ColorMode;
6  import imagingbook...PeronaMalikFilter.Parameters;
7
8  public class Perona_Malik_Demo implements PlugInFilter {
9
10   public int setup(String arg0, ImagePlus imp) {
11     return DOES_ALL + DOES_STACKS;
12   }
13
14   public void run(ImageProcessor ip) {
15     // create a parameter object:
16     Parameters params = new Parameters();
17
18     // modify filter settings if needed:
19     params.iterations = 20;
20     params.alpha = 0.15f;
21     params.kappa = 20.0f;
22     params.smoothRegions = true;
23     params.colorMode = ColorMode.ColorGradient;
24
25     // instantiate the filter object:
26     PeronaMalikFilter filter =
27         new PeronaMalikFilter(params);
28
29     // apply the filter:
30     filter.applyTo(ip);
31   }
32
33 }
```

17.5 Exercises

Prog. 17.1
Perona-Malik filter (complete ImageJ plugin). Inside the `run()` method, a parameter object (instance of class `PeronaMalikFilter.Parameters`) is created in line 16. Individual parameters may then be modified, as shown in lines 19–23. This would typically be done be querying the user (e.g., with ImageJ's `GenericDialog` class). In line 27, a new instance of `PeronaMalikFilter` is created, the parameter object (`params`) being passed to the constructor as the only argument. Finally, in line 30, the filter is (destructively) applied to the input image, that is, `ip` is modified. `ColorMode` (in line 23) is implemented as an enumeration type within class `PeronaMalikFilter`, providing the options `SeparateChannels` (default), `BrightnessGradient` and `ColorGradient`. Note that, as specified in the `setup()` method, this plugin works for any type of image and image stacks.

Exercise 17.5. Use the signal-to-noise ratio (SNR) to measure the effectiveness of noise suppression by edge-preserving smoothing filters on grayscale images. Add synthetic Gaussian noise (see Sec. D.4.3 in the Appendix) to the original image I to create a corrupted image $\tilde{I}$. Then apply the filter to $\tilde{I}$ to obtain $\bar{I}$. Finally, calculate $\mathrm{SNR}(I, \bar{I})$ as defined in Eqn. (13.2). Compare the SNR values obtained with various types of filters and different parameter settings, for example, for the *Kuwahara filter* (Alg. 17.2), the *bilateral filter* (Alg. 17.4), and the *Perona-Malik* anisotropic diffusion filter (Alg. 17.7). Analyze if and how the SNR values relate to the perceived image quality.