SIAM J. IMAGING SCIENCES
Vol. 8, No. 3, pp. 1396–1420

© 2015 SIAM. Published by SIAM under the terms
of the Creative Commons 4.0 license

# Dual-Domain Filtering*

C. Knaus† and M. Zwicker†

**Abstract.** We propose *dual-domain filtering*, an image processing paradigm that couples spatial domain with frequency domain filtering. Our dual-domain defined filter removes artifacts like residual noise of other image denoising methods and compression artifacts. Moreover, iterating the filter achieves state-of-the-art image denoising results, but with a much simpler algorithm than competing approaches. The simplicity and versatility of the dual-domain filter makes it an attractive tool for image processing.

**Key words.** image denoising, robust statistics, bilateral filter, Fourier transform

**AMS subject classifications.** 68U10, 94A08, 60G35, 65T50

**DOI.** 10.1137/140978879

## 1. Introduction.

Image enhancement and reconstruction are important tasks in image processing. Images may be degraded by additive white Gaussian noise, by arbitrary method noise, or by compression artifacts. To improve such images, specialized tools are often developed for each type of degradation. Some image processing tools are generally potent for attacking such problems. The bilateral filter (BF) [32] and its variant, the joint-bilateral filter [27], have become popular tools due to their simplicity and effectiveness in removing named artifacts. For example, bilateral filtering can be used for denoising images contaminated with weak noise or for removing unwanted details. Also, adaptive bilateral filtering has been proven effective for JPEG deblocking, as proposed by Zhang and Gunturk [36] and Nath, Hazarika, and Mahanta [25].

However, the BF, due to its spatial definition, makes a trade-off between removal of noise and loss of contrast and detail. Typically, details are better preserved using transform domain methods, which is why some JPEG deblocking methods inspect the discrete cosine transform (DCT) coefficients of the blocks. Sophisticated image denoising methods operate in both spatial and frequency domains. Moreover, it is common practice to cast artifact removal as a denoising problem, by simply using existing denoising methods [15, 31, 3, 16, 9, 8]. However, the best denoising methods are complex to implement and are not part of every image processing engineer's toolbox like the common BF is.

In this work, we introduce a simple but powerful image processing filter that we call the *dual-domain filter* (DDF). We build on *dual-domain image denoising* (DDID), which was recently introduced by Knaus and Zwicker [20] as a simple, but equally powerful, alternative to more complex image denoising methods. At the core, both DDID and DDF combine

*Received by the editors July 23, 2014; accepted for publication (in revised form) April 30, 2015; published electronically July 8, 2015.
http://www.siam.org/journals/siims/8-3/97887.html

†Institute of Computer Science and Applied Mathematics, University of Bern, Bern BE 3012, Switzerland (knaus@iam.unibe.ch, zwicker@iam.unibe.ch).



source: https://doi.org/10.7892/boris.81124 | downloaded: 18.4.2025 Redistribution subject to CCBY license
Downloaded 06/08/16 to 130.92.9.55

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

bilateral filtering with wavelet shrinkage using local, windowed Fourier transforms. Intuitively, the two steps compensate for each other's weaknesses: the bilateral kernel masks out high contrast edges that may lead to ringing in wavelet shrinkage, and the local Fourier transforms detect and preserve low-contrast repetitive structures that the bilateral kernel would tend to blur away. We show that a single pass of dual-domain filtering provides state-of-the-art performance for compression and denoising artifact removal. In addition, by iterating DDF, we obtain an image denoiser with excellent quality that is much simpler to implement than are related approaches.

DDF interprets bilateral filtering and wavelet shrinkage as robust noise estimators in two different domains. Durand and Dorsey have already made the connection of the bilateral filter to robust statistics [13], but they did not consider the bilateral filter as a robust noise estimator. Our approach is also related to a more recent work of Knaus and Zwicker called *progressive image denoising* (PID) [21], where the authors made the connection of wavelet shrinkage to robust estimation of noise differentials. The iteration in PID, however, requires many small steps. In contrast, with DDF we often obtain better results in only a few iterations.

In summary, we make the following contributions: First, we introduce the dual-domain filter (DDF), which performs noise estimation using arbitrary robust kernels in two domains. Second, we offer an extension of DDF to allow guided filtering using a second guide image. Third, we demonstrate new applications of this filter for removing denoising and compression artifacts. Last but not least, we provide a new formulation of an image denoiser based on iterating DDF that achieves state-of-the-art results, but with a much simpler algorithm than in competing techniques.

The remainder of the paper is organized as follows. We first review related work in section 2. We next introduce the DDF in section 3. Then, section 4 demonstrates three applications of DDF, including removal of denoising artifacts, removal of compression artifacts, and high-quality image denoising. Finally, section 5 concludes our work.

## 2. Related work.

We review work in the areas of image denoising and artifact removal of denoising and compression methods that is most relevant to our contribution. For a recent, more comprehensive survey of image denoising techniques we refer to the work by Shao et al. [30]. We restrict the discussion here to selected state-of-the-art techniques, focusing on their relation to our approach. Denoising approaches can be broadly categorized into spatial filtering, transform domain filtering, and dictionary learning-based methods. Spatial filtering techniques are conceptually very simple: they estimate denoised pixels by computing weighted averages of other pixels in the image. The BF [32] implements this idea by weighting pixels in a neighborhood window based on their similarity to the center pixel whose denoised value is estimated. The crux is that the weights of the BF are highly sensitive to the noise in the input; hence bilateral filtering by itself is not a very effective denoising approach, especially for larger noise levels. Recently, Caraffa, Tarel, and Charbonnier [5] described an iterated version of the BF that is robust to outliers, demonstrating how it can be used to remove non-Gaussian noise.

Takeda, Farsiu, and Milanfar [31] observed that the BF is a simple example of kernel regression. In kernel regression one computes local, weighted fits of a regression function to the noisy data, where the weights are provided by a kernel function. Takeda, Farsiu, and Milanfar

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

proposed improved regression-based denoising algorithms with data-adapted, anisotropic kernel functions, which are steered to align with image edges. Because their parametric kernel functions are prone to corruption by the noise in the input, they implement an iterative approach to denoise and re-estimate the kernel parameters in several steps. Bouboulis, Slavakis, and Theodoridis [2] propose a different approach in order to exploit kernels. They formulate denoising as a projection of the noisy input onto a reproducing kernel Hilbert space (RKHS), which is enriched with a semiparametric model that can explicitly represent sharp edges. Our approach is more related to kernel regression [31]. We also use an iterative approach to re-estimate a kernel in several steps. Instead of using a parametric kernel, however, we use a nonparametric bilateral kernel. In addition, instead of denoising using regression, we denoise using a form of wavelet shrinkage, that is, transform domain filtering.

The nonlocal means (NLM) filter generalizes the BF by considering the differences between pairs of small patches around a neighbor pixel and the center pixel instead of just the pixel differences. Comparing patches instead of pixels leads to weights that are much more robust to noise in the input, and NLM is significantly more effective than bilateral filtering for noise removal. NLM was first proposed by Buades, Coll, and Morel [3], and the basic idea has been refined and extended in many ways. An important problem is to estimate parameters of the algorithm in a data-adaptive manner. Kevrann and Boulanger [19] developed a technique to locally adapt the size of the neighborhood window. Van De Ville and Kocher [33, 34] estimate parameters of NLM using Stein's unbiased risk estimate (SURE). The computation of NLM can be accelerated by preselecting contributing patches based on various properties [23, 11, 26]. Denoising performance can be improved by combining it with kernel regression [6], clustering and principal component analysis [7], or spectral analysis [28] and by using more sophisticated patch similarity metrics, such as those based on principal component analysis [1] or exploiting rotational invariance [17].

The BM3D algorithm of Dabov et al. [9] combines the advantages of patch-based techniques like NLM with transform domain filtering. Instead of simply averaging pixels (or patches), the key idea in BM3D is to perform transform domain filtering on 3D (three-dimensional) blocks of similar patches. The denoising quality of BM3D is still considered state-of-the-art today. The success of BM3D inspired many variations of the basic scheme of collecting and jointly denoising similar patches. Several approaches are based on building statistical models of the collected patches. For example, Dabov et al. [10] use shape-adaptive principal component analysis (SAPCA), Chatterjee and Milanfar [8] propose the patch-based locally optimal Wiener filter (PLOW), and Lebrun, Buades, and Morel [22] use a nonlocal Bayes (NLB) approach that assumes a Gaussian distribution of patches and applies maximum a posteriori (MAP) estimation to obtain denoised patches. While interesting from a theoretical perspective, in practice these extensions often provide modest gains over the original BM3D algorithm. In contrast to these approaches, our algorithm is not patch-based. We do not rely on collecting similar patches nor on evaluating patch similarities. Instead, we operate directly on entire 2D neighborhood windows.

Classical transform domain methods rely on image representations using sets of suitable basis functions that are chosen such that the signal can be represented accurately by few coefficients. That is, the image representation in the transform domain is sparse, and noise corrupts mostly the small coefficients. Denoising in the transform domain is the problem of

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

estimating the basis coefficients of the denoised image, where one can exploit the sparsity of the representation. The most popular transforms are the DCT [35] and wavelets [29] and their many variations. Our approach is related to these techniques since it includes a transform domain filtering step based on local, windowed Fourier transforms. We combine this, however, with a bilateral kernel to avoid ringing artifacts, which otherwise often hamper pure transform domain approaches that rely on simple, data independent transforms like the Fourier transform. Our approach is related to denoising using shape adaptive DCT (SA-DCT) from Foi, Katkovnik, and Egiazarian [16]. Key differences between our work and theirs are that they use binary masks restricted to simple polygonal shapes, while we use bilateral kernels with continuous weights and arbitrary support. We directly apply the DFT to the masked data instead of using SA-DCT, and we iteratively refine the bilateral kernels and the denoising filters in several steps.

Learning-based approaches have become popular more recently. Burger, Schuler, and Harmeling [4] train a multilayer perceptron (MLP) for denoising. A disadvantage of this approach is that the perceptron has to be trained individually for each noise level. Dictionary learning-based methods construct patch-based representations by training overcomplete patch dictionaries from natural images. A classical approach is denoising with a dictionary learned using the K-SVD algorithm [15]. Learned dictionaries can also be combined with nonlocal techniques. The idea is to ensure that similar image patches are restored simultaneously using similar dictionary elements [24], which is called learned simultaneous sparse coding (LSSC). Dong, Shi, and Li [12] further build on this approach using low-rank techniques, and they propose spatially adaptive iterative singular-value thresholding (SAIST) for image denoising. Our approach achieves similar denoising performance with a much simpler algorithm that does not require any learning stage.

The problem of compression and denoising artifact removal is highly related to image denoising and often addressed with similar algorithms. For example, adaptive bilateral filtering has been proven effective for JPEG deblocking [36, 25]. It is common to simply cast artifact removal as a denoising problem and use existing denoising methods as discussed above to solve it. Similarly, we will show that our DDF is highly effective for addressing these problems too.

## 3. The dual-domain filter (DDF).

We formulate DDF as a robust noise estimator in two domains, the spatial and frequency domains. Typical image denoising filters estimate a signal $x$ directly from a noisy input $y$, attempting a decomposition $y = x+n$, where $n$ is the noise. In contrast to such filters, our filter first estimates the noise $n$, which is then subtracted from the noisy signal $y$ to obtain $x$. This seemingly subtle difference allows us to directly express noise estimation in both domains in an analogous fashion using robust kernels. While we make no assumptions about the signal, we assume to know the noise statistics. The noise statistics are used to robustly estimate the noise first in the spatial domain, then in the frequency domain.

For every pixel $p$, DDF estimates the noise $\hat{n}_p$ in two steps. DDF first uses a BF in the spatial domain to obtain an intermediate noise estimate $\bar{n}_p$ in the pixel value $y_p$. The BF is defined over a square neighborhood of pixels $q \in \mathcal{N}_p$, where $\mathcal{N}_p$ is a filter window, centered around pixel $p$ and limited by radius $r$. DDF then re-estimates the noise $\hat{n}_p$ in the frequency domain using the frequencies $f \in \mathcal{F}_p$, where $\mathcal{F}_p$ is the frequency domain implied by the neighborhood $\mathcal{N}_p$. The noise estimations in the two domains are described in subsections

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

3.1 and 3.2. In addition, in subsection 3.3 we show a way to let DDF be guided by a second *guide image*. We will leverage guided filtering for our applications of DDF to denoising artifact removal (section 4.1) and iterative image denoising (section 4.3).

**3.1. Noise estimation in the spatial domain.** The spatial domain filter is a reformulation of the BF, now designed to estimate the noise $\bar{n}_p$. We first subtract the pixel value $y_p$ from the neighbor pixel values $y_q, q \in \mathcal{N}_p$, forming the differences $d_q$ as

$$d_q = y_q - y_p. \tag{3.1}$$

In the following all symbols with subscripts $q$ are 2D arrays over $q \in \mathcal{N}_p$. We next introduce a bilateral kernel function $k(|d_q|^2, |q-p|^2)$ based on the squared norms of the differences $d_q$ and the distances $q-p$ between pixels. We assume that $k(\cdot,\cdot)$ includes a robust range kernel that normalizes the squared differences according to the known noise statistics (for concrete examples see section 4), and its purpose is to reject large values in $|d_q|^2$ as outliers (that is, signal), and retain small values as our noise estimates. Using the bilateral kernel function, we compute the discrete bilateral kernel

$$k_q = k\left(|d_q|^2, |q-p|^2\right), \tag{3.2}$$

which is a 2D array over all pixels $q \in \mathcal{N}_p$. Finally, we obtain the intermediate noise estimate $\bar{n}_p$ by locally convolving the differences $d_q$ with the normalized, discrete bilateral kernel $k_q / \sum_{q\in\mathcal{N}_p} k_q$,

$$\bar{n}_p = a \sum_{q\in\mathcal{N}_p} d_q\, k_q \Big/ \sum_{q\in\mathcal{N}_p} k_q. \tag{3.3}$$

We additionally introduce a confidence factor $a$ ranging from 0 to 1.

**3.2. Noise re-estimation in the frequency domain.** The frequency domain filter is designed to obtain our final noise estimate $\hat{n}_p$ by exploiting the intermediate results from the previous section. We apply a reformulation of wavelet shrinkage based on the discrete Fourier transform (DFT) for this purpose. First, we leverage the intermediate noise estimate $\bar{n}_p$ and the discrete bilateral kernel $k_q$ to avoid bias when re-estimating noise in the frequency domain. We start by subtracting the spatially estimated noise $\bar{n}_p$ from the differences $d_q$. Intuitively, all differences in $d_q$ are biased by the noise in the center pixel $y_p$. By subtracting the estimated noise $\bar{n}_p$, we remove this bias. Then we mask the resulting signal using the discrete bilateral kernel $k_q$ to remove large differences in $d_q$, which correspond to high contrast edges. They would otherwise bias the spectrum by introducing low-amplitude ringing at high frequencies, which would be confused with noise.

Now we are ready to perform noise estimation using the DFT. We first obtain the DFT by computing inner products of the preprocessed signal $(d_q - \bar{n}_p)\, k_q$ with the Fourier basis functions, yielding the Fourier coefficients $D_f$ as

$$D_f = \sum_{q\in\mathcal{N}_p} (d_q - \bar{n}_p)\, k_q\, e^{-i\frac{2\pi}{2r+1} f\cdot(q-p)}, \tag{3.4}$$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

with frequencies $f \in \mathcal{F}_p$. As before, the subscript $f$ in $D_f$ denotes that the symbol is a 2D array over all frequencies $f \in \mathcal{F}_p$ in the neighborhood window $\mathcal{N}_p$.

Next, we introduce the range kernel $K(\cdot)$ in the frequency domain, which is a function of a frequency coefficient, that is, a complex amplitude. We assume this to be a robust kernel that rejects large-amplitude signals, normalized by the energy according to the known noise statistics, and retains small values as our noise estimates (for concrete examples see section 4). Hence, $K(\cdot)$ serves the same purpose in the frequency domain as the bilateral kernel function $k(\cdot,\cdot)$ does in the spatial domain. Evaluating $K(\cdot)$ for all frequencies $f \in \mathcal{F}_p$ leads to a discrete frequency domain kernel,

$$K_f = K\left(|D_f|^2 \Big/ \sum_{q\in\mathcal{N}_p} k_q^2\right). \tag{3.5}$$

Here we normalize the energy of the Fourier coefficients by the energy of the bilateral kernel $k_q$, since the variance of a scaled signal is proportional to the squared factors. The pointwise product $D_f\, K_f$, where $f \in \mathcal{F}_p$, of the Fourier coefficients and the discrete frequency domain kernel now is devoid of high-amplitude coefficients, and it retains the low-amplitude coefficients as the desired noise estimates.

Finally, we reconstruct the center pixel noise $\hat{n}_p$ by applying an inverse DFT to the noise estimates $D_f\, K_f$. This amounts to taking the dot product in the frequency domain between the Fourier coefficients $D_f$ and the frequency kernel $K_f$ as

$$\hat{n}_p = A \sum_{f\in\mathcal{F}_p} D_f\, K_f \,/\, (2r+1)^2 = \mathrm{DDF}_p(y), \tag{3.6}$$

which we define as the output of DDF at pixel $p$. The normalization factor $1/(2r+1)^2$ corrects for the fact that the DFT is nonunitary. The parameter $A$ is another confidence factor between 0 and 1. Now we have the final noise estimate $\hat{n}_p$, and we can subtract it from the noisy pixel $y_p$ to get the estimate

$$\hat{x}_p = y_p - \mathrm{DDF}_p(y) = y_p - \hat{n}_p. \tag{3.7}$$

Durand and Dorsey [13] have made the connection of the BF to robust statistics and explored the replacement of the Gaussian kernels with other robust estimators, such as the Lorentzian and the Tukey estimator. In addition, Elad [14] showed that the BF is the first step in an iterative minimization of a local cost that is defined by the robust error norm corresponding to the robust kernel. The same space is available for exploration to DDF, to define the bilateral kernel function $k(\cdot,\cdot)$ and the new range kernel $K(\cdot)$ in the frequency domain. Concrete examples of these kernel functions are provided in section 4.

**3.3. Guided DDF.** We can also formulate DDF as a "guided filter," similar to the joint-bilateral filter [27] and the guided image filter [18]. Instead of having a single input image, we have an additional guide image $g$ that defines the filter, which is then applied to the noisy input image $y$. Since the same computations are performed on both images, we can use a trick by using the complex substitution

$$y \to g + i\, y. \tag{3.8}$$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

We only have to make minor adaptations. First, we extract the real part as the guide to define the bilateral kernel, and (3.2) becomes

$$k_q = k\left(|\operatorname{Re} d_q|^2, |q-p|^2\right). \tag{3.9}$$

Second, the Fourier transform now computes two real Fourier transforms simultaneously, one for the guide image $g$ and another for the noisy image $y$. We extract the Fourier coefficients of the real part as the guide with $\frac{D_f + D^*_{-f}}{2}$, and (3.5) becomes

$$K_f = K\left(\left|\frac{D_f + D^*_{-f}}{2}\right|^2 \Big/ \sum_{q\in\mathcal{N}_p} k_q^2\right). \tag{3.10}$$

Finally, the estimated noise is in the imaginary part of the output of guided DDF. Hence we write the noise estimate as $\hat{n}_p = \operatorname{Im}\mathrm{DDF}_p(g+iy)$, and the estimate of the denoised pixel as $x_p = y_p - \operatorname{Im}\mathrm{DDF}_p(g+iy)$. The MATLAB implementation of DDF given by Algorithm 1 (see the appendix) works for both guided and unguided DDF.

## 4. Applications.

We demonstrate three applications using DDF. In subsection 4.1, we remove residual noise from common image denoising algorithms. In subsection 4.2, we perform deblocking of JPEG images. Finally, in subsection 4.3, we iterate the DDF to perform high-quality image denoising. The code for artifact removal and image denoising of grayscale images is given by Algorithms 2 and 3 in the appendix and uses the DDF implementation of Algorithm 1. For all three applications we follow the adaptations made by DDID [20] to process color images (see Algorithms 4, 5, and 6 in the appendix). We perform a color-space transformation using DCT, and the range kernel in the BF relies on normalized Euclidean distances.

### 4.1. Removal of denoising artifacts.

To remove the artifacts of a denoising method, we postprocess the denoised output $g$ with DDF. Specifically, we use $g$ as the guide image to filter the original, noisy input image $y$. Hence, our output $x$ is

$$x = y - \operatorname{Im}\mathrm{DDF}(g+iy). \tag{4.1}$$

We configure DDF using the following confidence factors and kernels:

$$a = A = 1, \tag{4.2}$$

$$k(d^2,\rho^2) = e^{-\frac{d^2}{\gamma_r\,\sigma^2}}\, e^{-\frac{\rho^2}{2\,\sigma_s^2}}, \tag{4.3}$$

$$K(D^2) = \max\left(0, 1 - \frac{D^2}{\gamma_f\,\sigma^2}\right). \tag{4.4}$$

Here, $\sigma^2$ is the noise variance in the noisy input $y$. The bilateral kernel function $k(\cdot,\cdot)$ in the spatial domain is the ordinary bilateral kernel with a range parameter $\gamma_r$ and a scale parameter $\sigma_s$. For the range kernel in the frequency domain $K(\cdot)$, we choose the Epanechnikov estimator, introducing the range parameter $\gamma_f$. Both range kernels normalize their input by dividing it by the noise variance $\sigma^2$. We set the window radius for DDF as $r = 15$, the spatial scale as

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

$\sigma_s = 7$, the spatial range as $\gamma_r = 0.7$, and the frequency range as $\gamma_f = 2.3$ for all denoising methods and independent of input noise levels $\sigma^2$.

For grayscale images, we postprocess the output of the denoising methods K-LLD [7], K-SVD [15], PLOW [8], NLM [3], nonlocal Bayes (NLB) [22], LSSC [24], MLP [4], BM3D [9], BM3D-SAPCA [10], and SAIST [12]. For color images, we use the output of the methods supporting colors: PLOW, NLM, NLB, and BM3D. The noise sigma for grayscale images is $\sigma \in \{10, 25, 40\}$, for color images $\sigma \in \{25, 40\}$.

The top four rows of Figure 1 give visual examples for removing denoising artifacts. Low-frequency noise of K-SVD, graininess of NLM, outliers of NLB, and wavy patterns of LSSC: all these artifacts are reduced or removed by DDF. Tables 1 and 2 show that nearly all grayscale output of most methods can be numerically improved by postprocessing with DDF. Table 3 shows the same for color images. The more smooth regions an image has, the larger the gain is in PSNR. This is not surprising, since most methods excel at denoising natural images, whereas they have difficulties denoising synthetic images where smooth regions dominate. Images denoised by SAIST show almost no artifacts and can be improved only for high-noise situations where the signal is more homogeneous. For grayscale and color images with noise sigma $\sigma = 40$, nearly all images show improvement.

**4.2. JPEG artifact removal.** For JPEG deblocking, we have only the artifact contaminated image, so we use the same image for both the guide image $g$ and the noisy image $y$. Otherwise, we use the same kernel functions as in the previous section to define the DDF. We compressed grayscale and color images using three quality settings in MATLAB, $Q \in \{30, 20, 10\}$, and used empirically found corresponding noise sigma, $\sigma \in \{20, 25, 40\}$. For grayscale images, we use the parameters $r = 15$, $\sigma_s = 7$, $\gamma_r = 1.7$, and $\gamma_f = 1.1$. For color images, we change $\gamma_r = 2.8$ and $\gamma_f = 4.2$. We compare our results against SA-DCT, a state-of-the-art JPEG deblocker by Foi, Katkovnik, and Egiazarian [16]. We also compare against the bilaterally filtered image, using the same parameters as for DDF.

The last row in Figure 1 shows the deblocking of a JPEG image. The removed artifacts are the typical block patterns. The image improves almost everywhere. Table 4 numerically summarizes the results for deblocking JPEG images. For grayscale images, DDF approaches the quality of SA-DCT. For color images, the results are nearly identical.

**4.3. Image denoising.** Here we formulate an iterative image denoiser with DDF using $N$ iteration steps. We initialize our estimate $x_N$ with the noisy input $y$ and then perform the guided iteration

$$x_{n-1} = y - \operatorname{Im} \mathrm{DDF}(x_n + iy), \tag{4.5}$$

where $n$ counts down from $N$ to 1. A key idea is that we parameterize both the bilateral kernel function $k_n(\cdot, \cdot)$ in the spatial domain and the range kernel $K_n(\cdot)$ in the frequency domain depending on the iteration number $n$. Specifically, for iterative denoising we define them as

$$k_n(d^2, \rho^2) = \cos\left(\min\left(\frac{\pi}{2}, \sqrt{\frac{d^2}{T_n\, n}}\right)\right)^n e^{-\frac{\rho^2}{S_n}}, \tag{4.6}$$

$$K_n(D^2) = \cos\left(\min\left(\frac{\pi}{2}, \sqrt{\frac{D^2}{V\, n}}\right)\right)^n. \tag{4.7}$$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

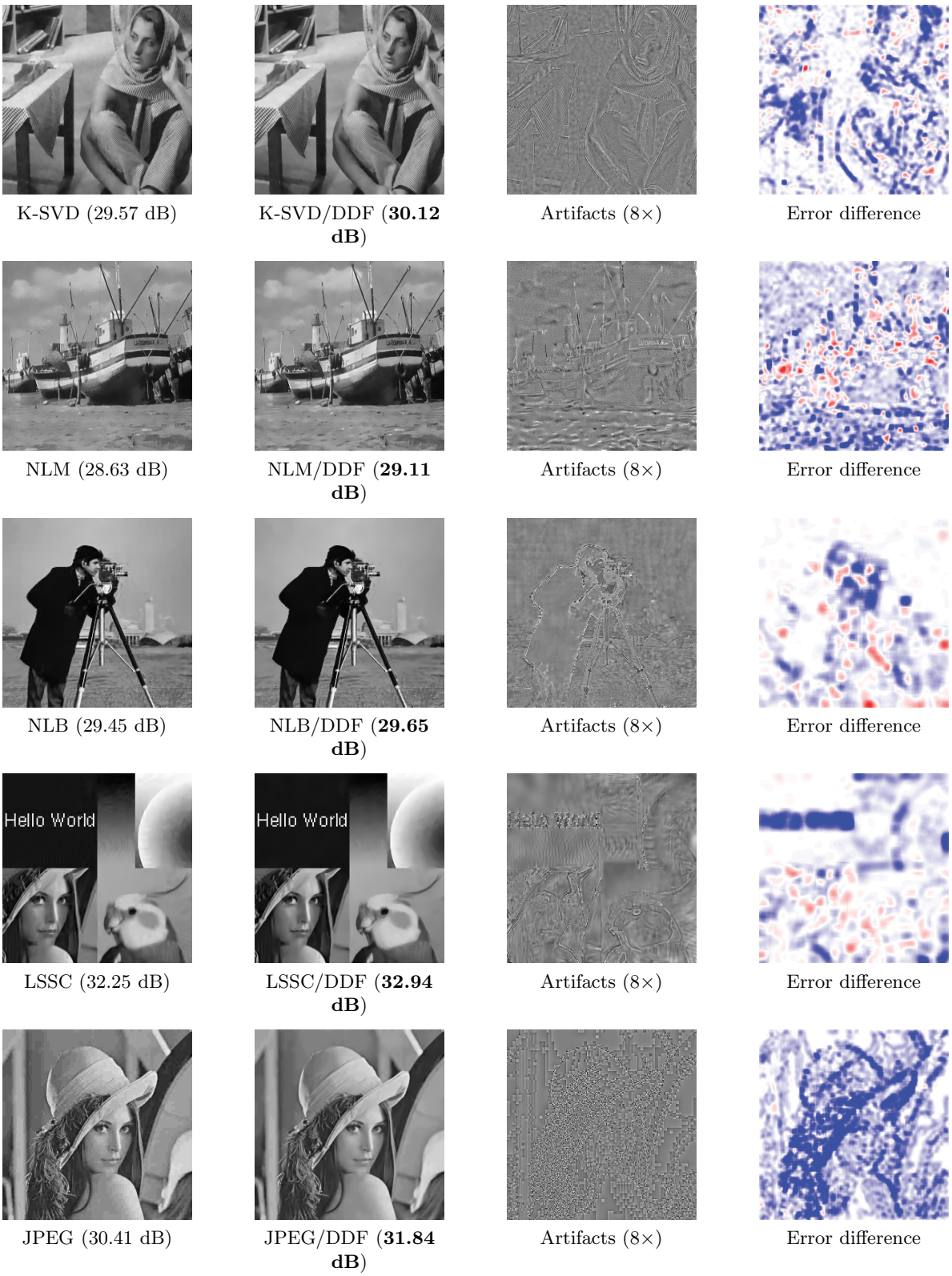


**Figure 1.** *Denoising and compression artifact removal: DDF removes artifacts like low-frequency noise, graininess, outliers, ringing, and blockiness. The noise sigma for the first four images was $\sigma = 25$. The JPEG compression of the last image used MATLAB quality $Q = 10$. The artifacts images are the difference images between the states before and after processing with DDF. In the error difference images, red and blue mark where the error increased and decreased, respectively. Better peak signal-to-noise ratio (PSNR) values are set in bold.*

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

**Table 1**

*Denoising artifact removal* (1/3): *PSNR (in dB) values for grayscale images before and after postprocessing with DDF. For state-of-the-art methods, with exception of SAIST and BM3D-SAPCA, and low-noise scenarios, DDF consistently removes artifacts and improves PSNR values. The MLP implementation does not provide weights for* $\sigma = 40$. *Better PSNR values are set in bold.*

| Grayscale | SAIST | BM3D-SAPCA | BM3D | MLP | LSSC |
|---|---|---|---|---|---|
| Barbara | **35.17** → 34.92 | **35.08** → 34.89 | **34.96** → 34.82 | **34.05** → 34.00 | **34.98** → 34.97 |
| Boats | **33.92** → 33.79 | **34.09** → 33.99 | **33.91** → 33.86 | **33.80** → **33.80** | **34.01** → 34.00 |
| Cameraman | **34.22** → 34.09 | **34.57** → 34.47 | 34.17 → **34.27** | 34.16 → **34.21** | 34.23 → **34.31** |
| Couple | **33.99** → 33.91 | **34.16** → 34.13 | 34.03 → **34.04** | 33.89 → **33.97** | 34.00 → **34.04** |
| Finger Print | **32.67** → 32.47 | **32.63** → 32.41 | **32.45** → 32.32 | **32.56** → 32.44 | 32.56 → **32.59** |
| Hill | **33.68** → 33.60 | **33.82** → 33.80 | 33.61 → **33.66** | 33.58 → **33.63** | 33.66 → **33.70** |
| House | **36.79** → 36.70 | **36.98** → 36.91 | 36.69 → **36.75** | 35.95 → **36.11** | 36.93 → **36.95** |
| Lena | **35.83** → 35.69 | **36.05** → 35.95 | **35.91** → 35.87 | 35.83 → **35.85** | 35.83 → **35.91** |
| Man | **34.12** → 34.04 | **34.23** → **34.23** | 33.97 → **34.06** | 34.09 → **34.16** | 34.09 → **34.14** |
| Montage | 37.17 → **37.24** | **37.81** → 37.80 | 37.32 → **37.52** | 36.48 → **37.11** | 37.23 → **37.50** |
| Pepper | **34.78** → 34.72 | **34.92** → 34.89 | 34.67 → **34.74** | 34.70 → **34.81** | 34.78 → **34.84** |

$\sigma = 10$

| Grayscale | SAIST | BM3D-SAPCA | BM3D | MLP | LSSC |
|---|---|---|---|---|---|
| Barbara | **31.22** → 31.07 | **30.99** → 30.91 | 30.71 → **30.74** | 29.54 → **29.74** | 30.48 → **30.70** |
| Boats | **29.96** → 29.95 | **30.02** → 30.01 | 29.90 → **29.95** | 29.97 → **30.00** | 29.89 → **30.00** |
| Cameraman | 29.41 → **29.55** | 29.81 → **29.88** | 29.44 → **29.68** | 29.61 → **29.85** | 29.50 → **29.84** |
| Couple | 29.74 → **29.80** | 29.81 → **29.89** | 29.71 → **29.83** | 29.73 → **29.84** | 29.66 → **29.84** |
| Finger Print | **27.93** → 27.73 | **27.80** → 27.61 | **27.70** → 27.61 | **27.65** → 27.49 | **27.62** → 27.60 |
| Hill | **29.89** → 29.88 | 29.95 → **29.96** | 29.85 → **29.89** | 29.87 → **29.88** | 29.82 → **29.92** |
| House | **33.16** → 33.10 | 32.95 → **33.00** | 32.85 → **33.02** | 32.56 → **32.80** | 33.11 → **33.15** |
| Lena | **32.25** → **32.25** | 32.22 → **32.23** | 32.07 → **32.19** | **32.25** → 32.22 | 31.85 → **32.16** |
| Man | 29.74 → **29.80** | 29.81 → **29.87** | 29.61 → **29.76** | 29.88 → **29.95** | 29.69 → **29.83** |
| Montage | 32.40 → **32.83** | 32.96 → **33.25** | 32.36 → **32.99** | 32.04 → **32.68** | 32.24 → **32.94** |
| Pepper | 30.39 → **30.58** | 30.43 → **30.52** | 30.16 → **30.44** | 30.30 → **30.64** | 30.22 → **30.49** |

$\sigma = 25$

| Grayscale | SAIST | BM3D-SAPCA | BM3D | MLP | LSSC |
|---|---|---|---|---|---|
| Barbara | **28.62** → 28.61 | **28.68** → 28.66 | 27.99 → **28.20** | n/a | 28.17 → **28.44** |
| Boats | 27.62 → **27.69** | 27.92 → **27.97** | 27.74 → **27.84** | n/a | 27.77 → **27.91** |
| Cameraman | 27.30 → **27.64** | 27.57 → **27.70** | 27.17 → **27.50** | n/a | 27.34 → **27.79** |
| Couple | 27.33 → **27.46** | 27.58 → **27.65** | 27.48 → **27.60** | n/a | 27.40 → **27.58** |
| Finger Print | **25.55** → 25.33 | **25.53** → 25.30 | **25.30** → 25.22 | n/a | **25.30** → **25.30** |
| Hill | 27.86 → **27.90** | 28.08 → **28.11** | 27.98 → **28.04** | n/a | 27.99 → **28.07** |
| House | **31.37** → 31.28 | 30.74 → **30.98** | 30.64 → **30.90** | n/a | 31.10 → **31.19** |
| Lena | 30.03 → **30.21** | 30.10 → **30.22** | 29.86 → **30.12** | n/a | 29.90 → **30.16** |
| Man | 27.57 → **27.70** | 27.82 → **27.90** | 27.64 → **27.79** | n/a | 27.64 → **27.83** |
| Montage | 29.57 → **30.38** | 30.01 → **30.67** | 29.52 → **30.41** | n/a | 29.43 → **30.51** |
| Pepper | 27.93 → **28.23** | 28.10 → **28.23** | 27.70 → **28.02** | n/a | 27.85 → **28.24** |

$\sigma = 40$

The bilateral kernel function $k_n(\cdot,\cdot)$ uses a clamped cosine raised to the $n$th power as its range and a Gaussian as its spatial kernel. It has a scale $S_n$ and a range parameter $T_n$ that depend on the iteration step $n$ as

$$S_n = 2\,\sigma_s^2\,\alpha^{\frac{1-n}{2N}}, \tag{4.8}$$

$$T_n = \gamma_r\,\sigma^2\,\alpha^{\frac{n-1}{N}}, \tag{4.9}$$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

**Table 2**

*Denoising artifact removal (2/3): PSNR (dB) values for grayscale images before and after postprocessing with DDF. For this set of methods, except for NLB at low noise sigma, DDF consistently removes artifacts and improves PSNR values.*

| Grayscale | NLB | NLM | PLOW | K-SVD | K-LLD |
|---|---|---|---|---|---|
| Barbara | **34.80** → 34.67 | **33.14** → 33.12 | 33.79 → **34.16** | 34.43 → **34.56** | 33.11 → **33.34** |
| Boats | **33.87** → 33.80 | **32.89** → 32.88 | 32.97 → **33.31** | 33.63 → **33.78** | 33.00 → **33.33** |
| Cameraman | **34.39** → 34.31 | 33.40 → **33.52** | 33.17 → **33.61** | 33.75 → **33.98** | 32.81 → **33.18** |
| Couple | **33.97** → **33.97** | 32.88 → **32.89** | 33.12 → **33.51** | 33.54 → **33.79** | 33.10 → **33.47** |
| Finger Print | **32.41** → 32.21 | **30.98** → 30.77 | 31.03 → **31.63** | 32.39 → **32.50** | 31.65 → **31.70** |
| Hill | **33.70** → 33.68 | **32.81** → 32.79 | 32.62 → **32.98** | 33.36 → **33.55** | 32.78 → **33.02** |
| House | **36.26** → 36.25 | 34.90 → **35.13** | 36.22 → **36.58** | 35.95 → **36.27** | 35.24 → **35.59** |
| Lena | 35.73 → **35.74** | 34.29 → **34.45** | 35.30 → **35.57** | 35.48 → **35.70** | 35.26 → **35.43** |
| Man | **34.11** → 34.09 | 33.07 → **33.09** | 32.95 → **33.42** | 33.59 → **33.87** | 33.18 → **33.56** |
| Montage | 37.20 → **37.39** | 35.23 → **35.61** | 36.12 → **36.99** | 36.08 → **36.78** | 35.43 → **36.44** |
| Pepper | **34.80** → 34.77 | 33.44 → **33.53** | 33.56 → **34.16** | 34.24 → **34.51** | 33.85 → **34.20** |

$\sigma = 10$

| Grayscale | NLB | NLM | PLOW | K-SVD | K-LLD |
|---|---|---|---|---|---|
| Barbara | 30.25 → **30.30** | 28.95 → **29.44** | 30.20 → **30.45** | 29.56 → **30.12** | 27.69 → **28.26** |
| Boats | 29.67 → **29.77** | 28.63 → **29.11** | 29.53 → **29.76** | 29.31 → **29.64** | 29.26 → **29.65** |
| Cameraman | 29.45 → **29.65** | 28.70 → **29.13** | 28.65 → **29.23** | 28.90 → **29.50** | 28.42 → **29.08** |
| Couple | 29.38 → **29.57** | 28.27 → **28.88** | 29.35 → **29.64** | 28.89 → **29.38** | 29.14 → **29.49** |
| Finger Print | **27.53** → 27.33 | **26.12** → **26.12** | 27.05 → **27.13** | 27.25 → **27.50** | **26.97** → **26.97** |
| Hill | 29.62 → **29.76** | 28.67 → **29.21** | 29.60 → **29.75** | 29.22 → **29.52** | 29.25 → **29.54** |
| House | 32.40 → **32.57** | 31.18 → **31.91** | 32.72 → **33.03** | 32.07 → **32.76** | 31.49 → **32.35** |
| Lena | 31.79 → **31.93** | 30.41 → **31.13** | 31.90 → **32.13** | 31.35 → **31.84** | 31.42 → **31.90** |
| Man | 29.62 → **29.76** | 28.60 → **29.13** | 29.33 → **29.62** | 29.11 → **29.49** | 29.27 → **29.63** |
| Montage | 31.97 → **32.53** | 30.70 → **31.71** | 30.95 → **32.71** | 31.16 → **32.30** | 30.52 → **31.82** |
| Pepper | 30.12 → **30.31** | 28.69 → **29.46** | 29.63 → **30.23** | 29.70 → **30.30** | 29.56 → **30.20** |

$\sigma = 25$

| Grayscale | NLB | NLM | PLOW | K-SVD | K-LLD |
|---|---|---|---|---|---|
| Barbara | 28.07 → **28.26** | 26.65 → **27.76** | 28.10 → **28.37** | 26.89 → **27.60** | 24.84 → **25.87** |
| Boats | 27.39 → **27.61** | 26.27 → **27.15** | 27.63 → **27.86** | 27.06 → **27.44** | 26.33 → **27.23** |
| Cameraman | 27.17 → **27.53** | 26.49 → **27.12** | 26.66 → **27.32** | 26.76 → **27.46** | 25.58 → **26.63** |
| Couple | 27.18 → **27.46** | 25.66 → **26.80** | 27.32 → **27.58** | 26.39 → **26.91** | 26.19 → **27.01** |
| Finger Print | **25.39** → 25.27 | 24.07 → **24.64** | **25.20** → 25.12 | 24.69 → **25.04** | 24.00 → **24.28** |
| Hill | 27.75 → **27.95** | 26.45 → **27.53** | 27.89 → **28.04** | 27.21 → **27.58** | 26.53 → **27.43** |
| House | 30.26 → **30.67** | 28.83 → **30.18** | 30.55 → **31.05** | 29.58 → **30.50** | 27.44 → **29.04** |
| Lena | 29.81 → **30.09** | 28.23 → **29.53** | 29.86 → **30.20** | 29.05 → **29.73** | 27.63 → **29.02** |
| Man | 27.51 → **27.74** | 26.35 → **27.34** | 27.52 → **27.80** | 27.04 → **27.47** | 26.42 → **27.34** |
| Montage | 29.11 → **30.07** | 27.67 → **29.48** | 27.90 → **30.06** | 28.64 → **29.93** | 26.71 → **28.40** |
| Pepper | 27.69 → **28.10** | 25.73 → **27.27** | 27.50 → **28.09** | 27.40 → **28.03** | 26.15 → **27.23** |

$\sigma = 40$

where $\sigma^2$ again is the noise variance. Intuitively, as $n$ counts down from $N$ to 1, the scale $S_n$ becomes larger; that is, the bilateral kernel function considers a larger and larger neighborhood of pixels. On the other hand, the range parameter $T_n$ becomes smaller, which means that as the support of the bilateral kernel function grows, it becomes more sensitive to pixel differences. We visualize this behavior in Figure 2 on the left. The definition implies that at the end of the iteration the scale is $S_1 = 2\sigma_s^2$ and the range is $T_1 = \gamma_r \sigma^2$, where $\sigma_s$ and $\gamma_r$ are parameters that will need to be specified. The base $\alpha$ controls the initial values $T_N$ and $S_N$. The range kernel in the frequency domain $K_n(\cdot)$ is also defined as a raised cosine. Its range parameter

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

**Table 3**

*Denoising artifact removal* (3/3): *PSNR (dB) values for color images before and after postprocessing with DDF. For color images with noise sigma $\sigma = 40$, the PSNR values almost always improve.*

| Color | BM3D | NLB | NLM | PLOW |
|---|---|---|---|---|
| Baboon | 25.95 → **26.10** | **26.53** → 26.44 | 25.65 → **25.98** | 24.22 → **24.96** |
| F-16 | 32.77 → **33.00** | **33.03** → 33.00 | 31.31 → **32.21** | 30.97 → **32.23** |
| House | **33.02** → 32.81 | **32.65** → 32.59 | 31.39 → **31.99** | 31.91 → **32.53** |
| Kodak 1 | 29.12 → **29.14** | 29.29 → **29.32** | 27.49 → **28.43** | 25.68 → **26.60** |
| Kodak 2 | **32.44** → 32.26 | 32.28 → **32.30** | 30.69 → **31.49** | 29.86 → **31.21** |
| Kodak 3 | **34.56** → 34.51 | 34.49 → **34.50** | 32.33 → **33.39** | 30.46 → **32.51** |
| Kodak 12 | **33.76** → 33.52 | **33.37** → 33.28 | 31.62 → **32.36** | 30.02 → **31.92** |
| Lena | 32.27 → 32.27 | **32.25** → 32.22 | 30.88 → **31.53** | 31.00 → **31.73** |
| Pepper | 31.22 → **31.27** | 31.23 → 31.23 | 30.28 → **30.83** | 30.37 → **31.00** |
| Lake | 28.68 → **28.87** | **29.10** → 29.08 | 28.29 → **28.58** | 27.60 → **28.27** |
| Tiffany | 32.31 → **32.41** | 32.37 → **32.53** | 31.13 → **31.91** | 31.51 → **32.08** |

$\sigma = 25$

| Color | BM3D | NLB | NLM | PLOW |
|---|---|---|---|---|
| Baboon | 23.87 → **24.07** | **24.50** → 24.47 | 23.22 → **23.91** | 22.56 → **23.17** |
| F-16 | 30.24 → **30.84** | 30.87 → **30.94** | 28.61 → **30.07** | 29.05 → **30.37** |
| House | 30.59 → **31.00** | 30.85 → **30.99** | 29.05 → **30.43** | 30.04 → **30.91** |
| Kodak 1 | 26.59 → **26.68** | 26.84 → **26.97** | 24.76 → **25.67** | 24.32 → **25.29** |
| Kodak 2 | 30.30 → **30.41** | 30.24 → **30.37** | 28.36 → **29.54** | 28.22 → **29.75** |
| Kodak 3 | 31.59 → **31.88** | 32.08 → **32.13** | 29.96 → **31.03** | 28.46 → **30.69** |
| Kodak 12 | 31.32 → **31.38** | **31.26** → 31.20 | 29.41 → **30.49** | 28.04 → **30.29** |
| Lena | 30.11 → **30.47** | 30.48 → **30.54** | 28.90 → **29.85** | 29.26 → **30.15** |
| Pepper | 29.32 → **29.77** | 29.65 → **29.76** | 28.19 → **29.34** | 28.72 → **29.64** |
| Lake | 26.88 → **27.22** | 27.43 → **27.44** | 26.04 → **26.63** | 26.03 → **26.74** |
| Tiffany | 30.24 → **30.50** | 30.40 → **30.68** | 28.71 → **30.00** | 29.84 → **30.44** |

$\sigma = 40$

$V$, however, is constant over the iterations and defined as $V = \gamma_f \sigma^2$, where the parameter $\gamma_f$ will need to be specified.

By raising the cosines to the $n$th power, we change the shape of the range kernels over the iterations. We found that Gaussians work better in the beginning, and functions with strong outlier rejection like the clamped cosine kernel work better at the end of the iterations. Since the Gaussian can be approximated by powers of cosines, we use this relationship to dynamically adjust the shapes of our range kernels in both domains. We also scale the cosines by a factor $1/\sqrt{n}$ such that the exponent $n$ affects mostly the shapes of the functions but not their overall widths. As visualized in Figure 2 in the middle and on the right, the range kernels in both the spatial and frequency domains start as approximate Gaussians, become steeper over time, and end as clamped cosines. The range kernel in the spatial domain additionally becomes narrower as $T_n$ gets smaller over the iterations, while the width of the range kernel in the frequency domain stays practically constant.

We also specify the confidence factors $a$ and $A$ dynamically. In the beginning, due to the small scale parameters, the spatially estimated noise cannot be trusted. Over time, the noise estimate becomes more accurate until, in the end, it can be fully trusted. The same applies for the estimated noise in the frequency domain. We therefore specify that the confidence factors $a_n$ and $A_n$ follow a sine ramp from 0 to 1, expressed as

$$a_n = A_n = \cos\left(\frac{n-1}{N}\frac{\pi}{2}\right). \tag{4.10}$$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license



**Table 4**

*JPEG artifact removal: PSNR (dB) values for JPEG grayscale and color images before and after postprocessing with DDF. For grayscale images, DDF approaches SA-DCT in quality. For color images, their results are similar.*

| Grayscale | JPEG | BF | DDF | SA-DCT | JPEG | BF | DDF | SA-DCT | JPEG | BF | DDF | SA-DCT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Barbara | 30.16 | 29.65 | **31.09** | 30.72 | 28.25 | 27.83 | **29.26** | 28.90 | 25.70 | 25.15 | **26.95** | 26.54 |
| Boats | 31.83 | 30.25 | 32.48 | **32.54** | 30.49 | 28.88 | 31.23 | **31.28** | 28.13 | 26.34 | 29.09 | **29.14** |
| Cameraman | 29.93 | 29.96 | **30.70** | 30.69 | 28.59 | 28.62 | 29.37 | **29.39** | 26.47 | 26.20 | 27.33 | **27.48** |
| Couple | 31.75 | 30.02 | 32.39 | **32.44** | 30.41 | 28.55 | 31.18 | **31.24** | 28.05 | 25.81 | 28.98 | **29.04** |
| Finger Print | 31.16 | 29.61 | 31.71 | **32.05** | 29.49 | 27.61 | 30.20 | **30.54** | 26.57 | 23.61 | 27.52 | **27.80** |
| Hill | 32.04 | 30.27 | **32.55** | 32.53 | 30.82 | 28.95 | **31.45** | 31.42 | 28.61 | 26.55 | **29.43** | 29.40 |
| House | 34.20 | 33.18 | **35.19** | 35.07 | 33.02 | 31.92 | 34.09 | **34.10** | 30.56 | 28.82 | 31.93 | **32.09** |
| Lena | 34.28 | 32.42 | 35.09 | **35.12** | 32.96 | 31.19 | 34.01 | **34.04** | 30.41 | 28.66 | **31.84** | **31.84** |
| Man | 31.80 | 30.33 | 32.50 | **32.55** | 30.55 | 29.00 | 31.34 | **31.39** | 28.27 | 26.62 | 29.25 | **29.30** |
| Montage | 32.76 | 33.18 | 34.13 | **34.15** | 31.24 | 31.53 | 32.61 | **32.63** | 28.56 | 28.51 | 30.04 | **30.14** |
| Pepper | 31.63 | 30.91 | 32.64 | **32.74** | 30.29 | 29.50 | 31.38 | **31.52** | 27.82 | 26.69 | 29.10 | **29.30** |
| | | | $Q = 30$ | | | | $Q = 20$ | | | | $Q = 10$ | |

| Color | JPEG | BF | DDF | SA-DCT | JPEG | BF | DDF | SA-DCT | JPEG | BF | DDF | SA-DCT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Baboon | 23.85 | 24.08 | **24.18** | 24.07 | 23.07 | 23.27 | **23.44** | 23.38 | 21.63 | 21.74 | **22.15** | 22.13 |
| F-16 | 30.06 | 30.99 | **31.20** | 31.08 | 28.90 | 29.92 | **30.17** | 30.12 | 26.87 | 28.01 | **28.33** | 28.30 |
| House | 28.96 | **29.91** | 29.85 | 29.78 | 27.87 | **28.83** | 28.81 | 28.77 | 26.25 | 27.22 | 27.48 | **27.53** |
| Kodak 1 | 28.21 | 28.34 | 28.77 | **28.83** | 26.94 | 27.05 | 27.57 | **27.63** | 24.77 | 24.56 | 25.48 | **25.52** |
| Kodak 2 | 31.37 | 31.15 | **31.99** | 31.83 | 30.01 | 29.91 | **30.70** | 30.64 | 27.85 | 27.84 | **28.67** | 28.63 |
| Kodak 3 | 32.86 | 33.21 | **34.00** | **34.00** | 31.44 | 31.96 | **32.68** | **32.68** | 28.56 | 29.10 | 29.81 | **29.85** |
| Kodak 12 | 32.81 | 32.60 | **33.62** | 33.61 | 31.33 | 31.39 | **32.30** | 32.26 | 28.71 | 29.18 | **29.81** | 29.76 |
| Lake | 26.84 | 27.43 | **27.57** | 27.38 | 26.07 | 26.76 | **26.92** | 26.79 | 24.39 | 25.02 | **25.37** | 25.34 |
| Lena | 30.91 | 31.20 | **31.87** | 31.79 | 29.83 | 30.29 | **31.01** | 31.00 | 27.53 | 28.20 | 29.00 | **29.06** |
| Pepper | 28.40 | 29.01 | **29.19** | 29.14 | 27.57 | 28.32 | **28.54** | **28.54** | 25.77 | 26.72 | 27.03 | **27.12** |
| Tiffany | 29.21 | 29.50 | **29.80** | 29.64 | 28.40 | 28.74 | **29.11** | 29.00 | 26.83 | 27.37 | **27.79** | 27.78 |
| | | | $Q = 30$ | | | | $Q = 20$ | | | | $Q = 10$ | |

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

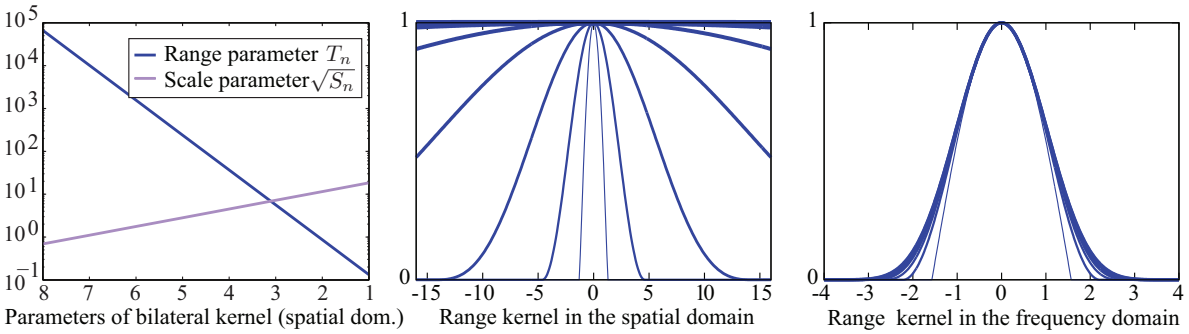


**Figure 2.** *Evolution of the kernels in DDID2. On the left we show semilogarithmic plots of the scale and range parameters $\sqrt{S_n}$ and $T_n$ of the bilateral kernel in the spatial domain over the iteration steps $n = \{8, \dots, 1\}$, illustrating how the scale increases exponentially and the range similarly decreases. In the middle we show the shapes of the range kernels in the spatial domain, from thick to thin lines as $n$ counts down from $n = 8$ to 1. These are the raised cosines from (4.6) as functions of differences $d/\sigma$, normalized by the noise variance $\sigma$, and it is apparent that they become narrower over the iterations. On the right we similarly plot the range kernels in the frequency domain from (4.7) as functions of amplitudes $D/\sigma$ normalized by noise variance. The range kernels become steeper over the iterations, leading to stronger outlier rejection as the iteration progresses. The plots correspond to the default parameters $N = 8$, $\gamma_r = 5.3/N$, $\gamma_f = 13/N$, and $\alpha = e^{15}$.*

Finally, for improved computational performance, we dynamically adapt the window radius. We define the window radius $r_n$ to be twice the spatial standard deviation $\sqrt{S_n/2}$ and at least 4 pixels large. The window radius $r_n$ is thus

$$r_n = \max\left(4, \operatorname{round}\left(2\sqrt{S_n/2}\right)\right). \tag{4.11}$$

**4.3.1. Discussion.** We visualize the last four steps $n = 4, 3, 2, 1$ in the DDID2 iteration in Figure 3. In the top row we show the intermediate denoising results after each iteration. Below, we visualize the bilateral kernels $k_n$ (from (4.6)) and the overall DDF kernels for three center pixels. In the leftmost column we show the corresponding neighborhood windows (see also red squares in the top left image) with the center pixels marked white, before and after denoising. The bilateral weights are always between 0 and 1, visualized using black and white, respectively. The overall DDF kernels consist of filter weights that correspond to (4.5). We compute these weights by expressing the window around a center pixel as a vector (by unrolling the window), and writing the application of DDF on this window as a sequence of matrix multiplications. For example, applying $k_n$ (from (4.6)) and $K_n$ (from (4.7)) are multiplications with diagonal matrices, and taking the DFT and its inverse can also be expressed as matrices, etc. We extract the filter weights for the center pixel as the central row of this matrix. The DDF weights can be negative; hence in the visualizations the gray background represents 0, brighter values are positive, and darker ones negative. The kernels are normalized such that the largest magnitude appears as white (or black). The visualizations show how, over the DDID2 iterations, the DDF kernels more and more accurately detect and follow the structures in the input image.

We show similar visualizations of modified versions of DDID2 in Figures 4 and 5 to provide more intuition. In Figure 4 we show DDID2 using DDF without the bilateral masking step; that is, we set $k_n(\cdot, \cdot) \equiv 1$ and we ignore the noise estimate in the spatial domain by setting

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

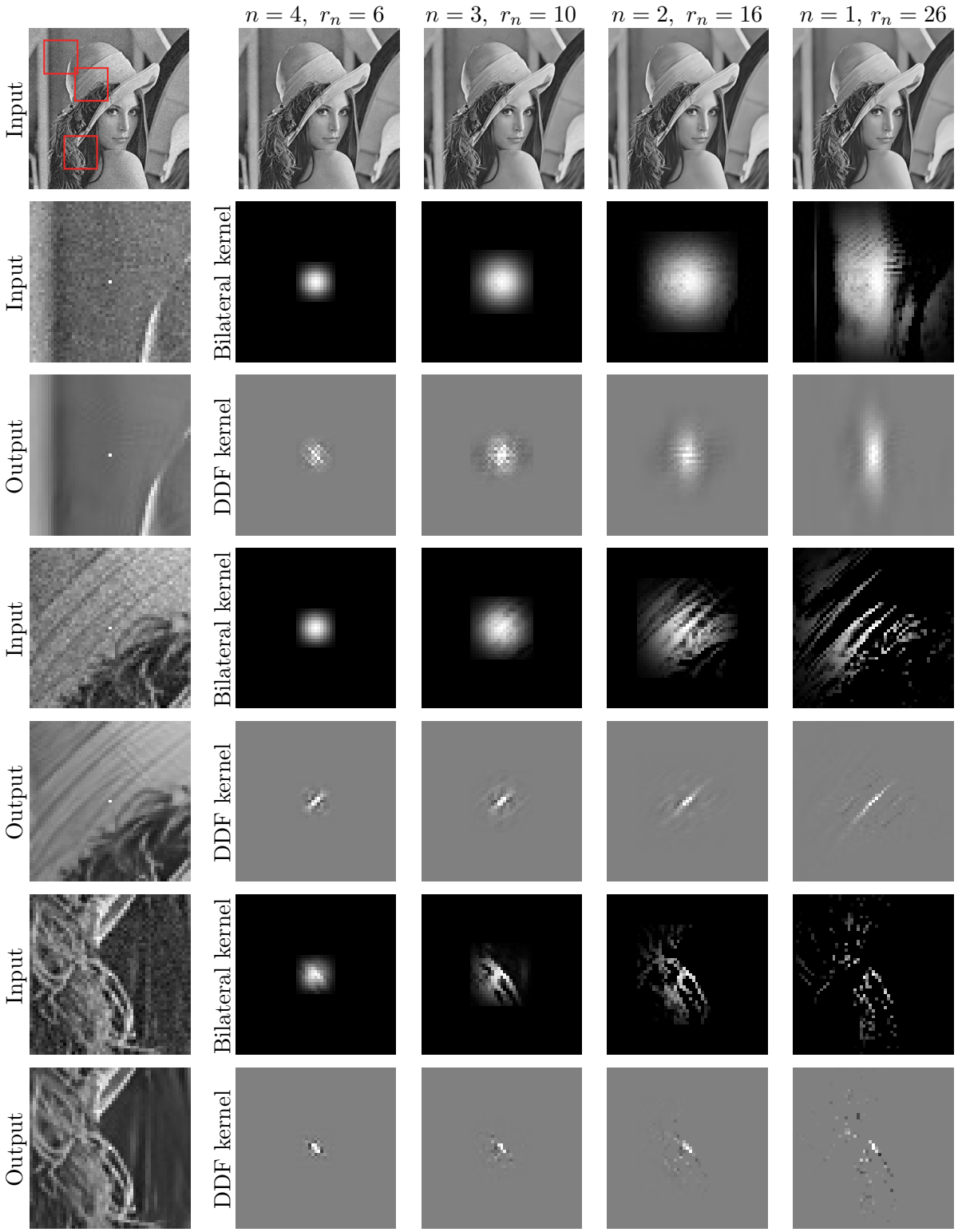


**Figure 3.** *Visualization of DDF kernels in DDID2 for iterations $n = \{4, 3, 2, 1\}$. For the visualization we zero-pad the windows with radius $r_n$ to match the largest window size at $n = 1$. Over the iterations, the DDF kernels more and more accurately capture detailed image structures, even at low contrast.*

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

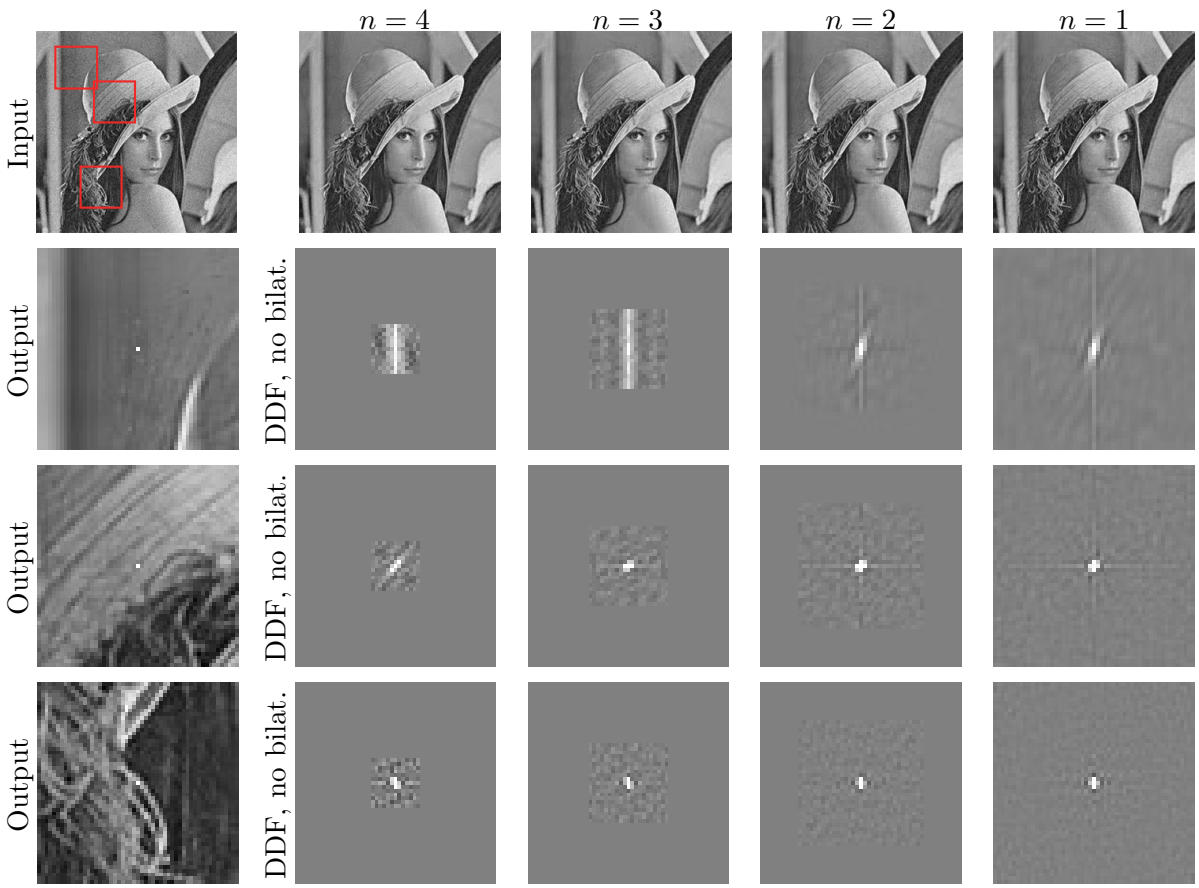


**Figure 4.** *Visualization of DDID2 using DDF kernels without the bilateral mask in the spatial domain. We zero-pad the windows for iterations $n = \{4, 3, 2\}$ to match the window size at $n = 1$. Without the bilateral mask we observe ringing artifacts and residual noise. In addition, the modified DDF kernels do not adapt to the image structures.*

$\bar{n}_p = 0$. This means that denoising relies only on robust noise estimation in the Fourier domain. Without bilateral masking, however, ringing artifacts appear. Also, the resulting kernels do not adapt to structures in the image well. On the other hand, Figure 5 illustrates DDID2 using DDF without noise estimation in the Fourier domain; that is, we rely only on the spatial domain noise estimate $\bar{n}_p$ (from (3.3)). In this case DDID2 is a form of iterative joint-bilateral filtering, similar to the "rolling guidance filter" recently proposed by Zhang et al. [37]. While the filter adapts to and preserves strong edges, we lose most low-contrast image detail. The combination of both steps, however, is surprisingly effective (Figure 3).

Similar denoising filters have been proposed previously, like DDID [20] and the more recent PID [21]. The differences between their approaches and ours are the following: DDID relies on three steps of guided DDF steps with somewhat less effective kernels than DDID2. It uses a fixed scale $\sigma_s$ for the spatial Gaussian, and Gaussians instead of raised cosines with dynamically changing shapes for the range kernels. In DDID the spatial and frequency range parameters $\gamma_r$ and $\gamma_f$ are set manually for each of the three iterations. In contrast, DDID2

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

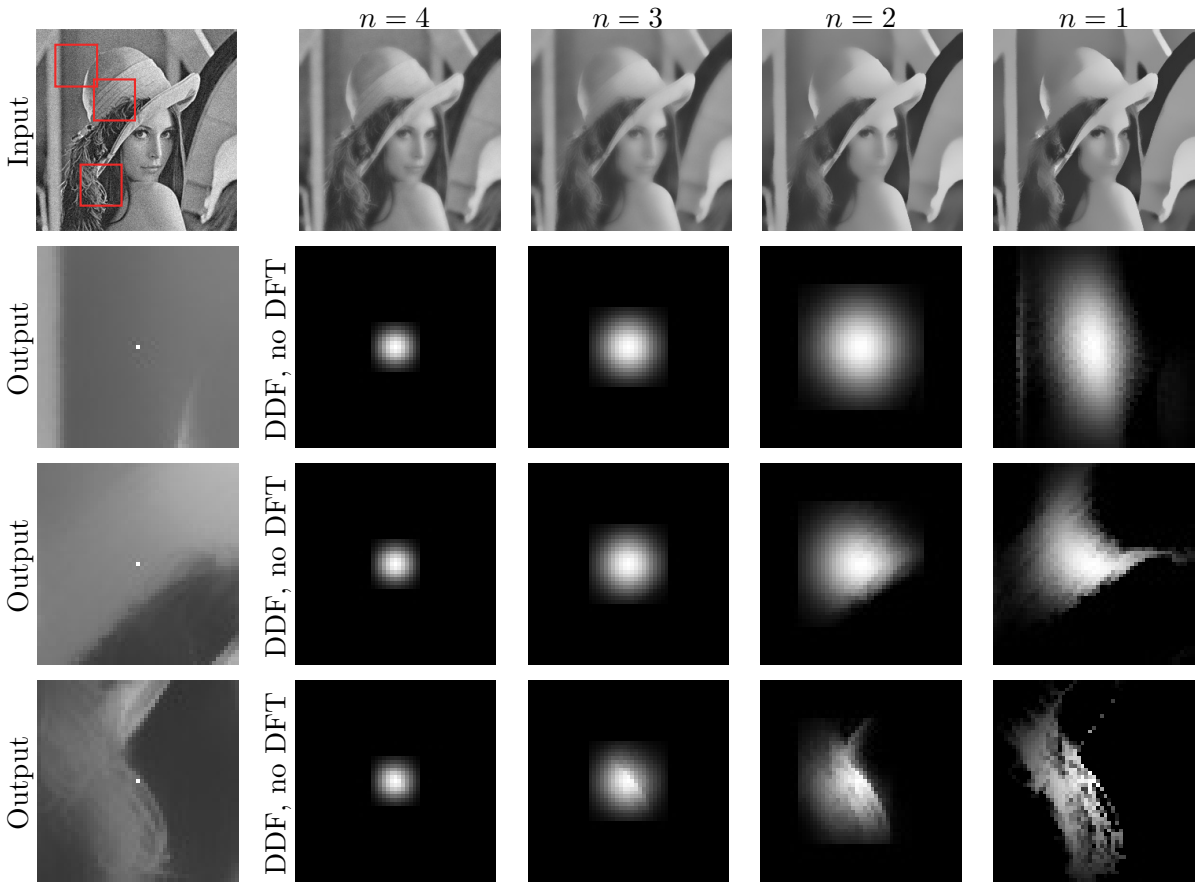


**Figure 5.** *Visualization of DDID2 using DDF kernels without noise estimation in the Fourier domain using the DFT. In this case DDID2 amounts to a form of iterative cross-bilateral filtering. We zero-pad the windows for iterations $n = \{4, 3, 2\}$ to match the window size at $n = 1$. While the bilateral kernel adapts to high-contrast edges, it loses most low-contrast details.*

computes its range parameters as a function of the iteration number. A key improvement of DDID2 over DDID is that DDID2 allows an arbitrary choice of the number of iteration steps $N$, and using more than three iteration steps leads to significant improvements, as can be seen in Figure 6. PID uses a bilateral kernel with range and scale parameters similar to (4.8) and (4.9) due to deterministic annealing. However, PID does not change the shape of the range kernels as we propose in (4.6) and (4.7). More importantly, the PID formulation is not guided and needs at least 30 iterations and an additional guided DDID step for high-quality results. Our formulation requires only 8 guided iterations and is as simple as DDID, but it achieves consistently higher quality results. In addition, neither of these previous works investigated the effectiveness of the DDF framework for other tasks such as compression and denoising artifact removal.

**4.3.2. Results.** We achieved the best results by using the constants $N = 8$, $\sigma_s = 13$, $\gamma_r = 5.3/N$, $\gamma_f = 13/N$, and $\alpha = e^{15}$. All constants are fixed, independent of the noise

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

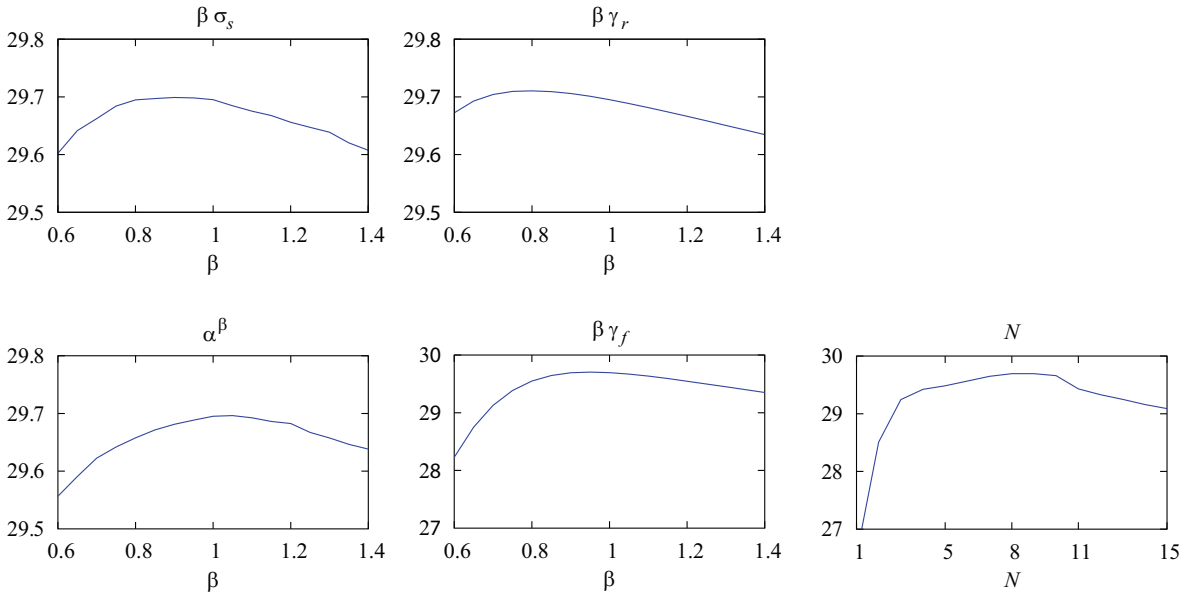


**Figure 6.** *Image denoising using DDID2: We plot PSNR (dB) values when denoising the Cameraman image against parameter changes using a perturbation factor $\beta$ applied to the default values. We include the scale and range parameters $\sigma_s$ and $\gamma_r$, respectively, of the bilateral kernel in the spatial domain, the base value $\alpha$, and the range parameter $\gamma_f$ in the frequency domain. We also plot PSNR as a function of the number of iteration steps $N$. The plots show that DDID2 is robust with respect to changes of most parameters. The most influential parameters are the range parameter $\gamma_f$ in the frequency domain, which controls the noise estimation directly, and the number of iterations $N$. The default values for the less influential parameters $\alpha$, $\sigma_s$, and $\gamma_r$ are close to optimal in this example. Changing them has little impact on the PSNR in general, in the order of* 0.1 *dB.*

level $\sigma$. These parameters lead to a sequence $\{4, 4, 4, 4, 6, 10, 16, 26\}$ of window radii $r_n$ for $n = 8, \ldots, 1$. In Figure 6 we illustrate the robustness of our method, which we call DDID2, with respect to parameter changes. The plots show PSNR values as functions either of a perturbation value $\beta$ that modifies the default parameter values, or as a function of the number $N$ of iteration steps. For the analysis, we used the Cameraman image and fixed the noise sigma to $\sigma = 25$. The plots for other images and noise levels are similar. Changing the number of iterations $N$ or the frequency range parameter $\gamma_f$ has the biggest influence on the PSNR. The other parameters are robust against change, and PSNR values remain close to the optimum within a range on the order of 0.1 dB.

We provide MATLAB code for DDF and its applications to artifact removal and DDID2 denoising for both color and grayscale in the appendix. The MATLAB implementation of DDID2 takes about 100 seconds to process a $512 \times 512$ grayscale image using twelve threads on a Xeon E5-2630 CPU at 2.3 GHz, independent of the noise level.

Figures 7 and 8 visually compare our new denoiser, DDID2, against other state-of-the-art denoising methods that support color images. DDID2 and PID produce the cleanest and smoothest results. Table 5 summarizes the numerical results, showing PSNR values for several test images. We also rank all methods for each image and provide the median rank over all images as an aggregate statistic for each method (note that several methods can have the

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license


**Figure 7.** *Image denoising: PSNR (dB) comparison for a natural color image with $\sigma = 40$. DDID2 and PID produce the most aesthetically pleasing results.*

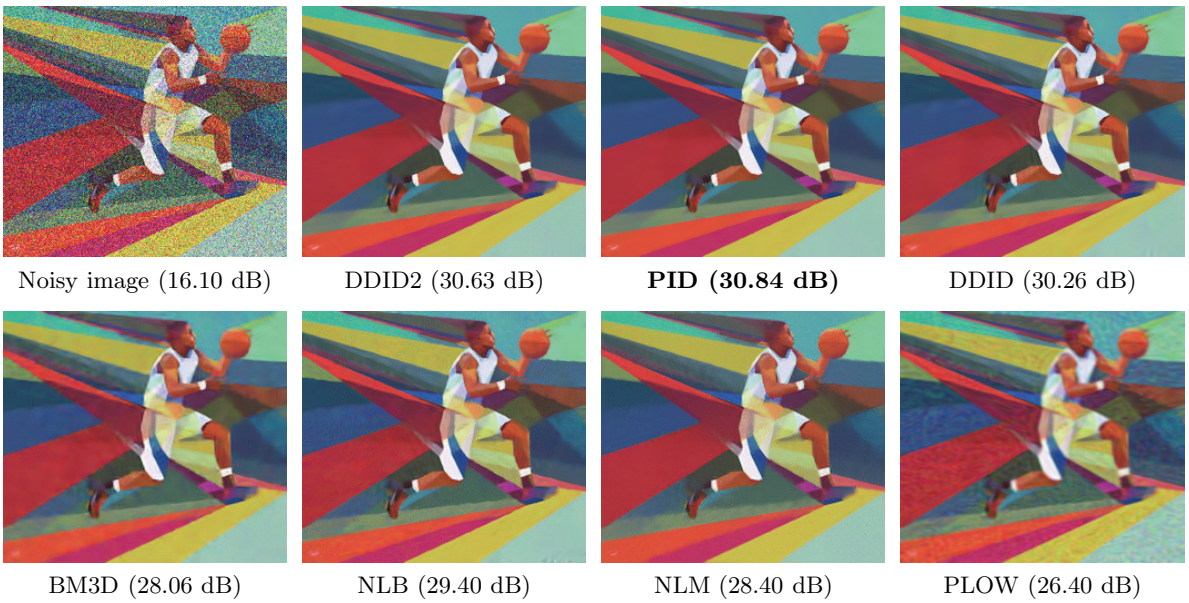


**Figure 8.** *Image denoising: PSNR (dB) comparison for a synthetic color image with $\sigma = 40$. DDID2 and PID preserve edges better and produce cleaner results.*

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

**Table 5**

*Image denoising: PSNR (dB) values for denoising grayscale and color images. We also indicate the median rank for each method over all images. For grayscale images, DDID2 competes with the best denoisers, MLP, BM3D, BM3D-SAPCA, and SAIST. For color images, DDID2 has the highest PSNR values and the best median rank of all methods. The MLP implementation does not provide weights for $\sigma = 40$.*

| Grayscale | DDID2 | PID | DDID | SAIST | SAPCA | BM3D | MLP | LSSC | NLB | NLM | PLOW |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Barbara | 30.82 | 30.56 | 30.80 | **31.23** | 31.00 | 30.72 | 29.55 | 30.49 | 30.25 | 28.95 | 30.21 |
| Boats | 29.88 | 29.80 | 29.79 | 29.97 | **30.03** | 29.91 | 29.97 | 29.90 | 29.67 | 28.63 | 29.54 |
| Cameraman | 29.69 | 29.68 | 29.47 | 29.41 | **29.81** | 29.45 | 29.61 | 29.50 | 29.45 | 28.70 | 28.66 |
| Couple | 29.67 | 29.65 | 29.56 | 29.74 | **29.82** | 29.72 | 29.74 | 29.67 | 29.38 | 28.27 | 29.36 |
| Finger Print | 27.34 | 27.15 | 27.32 | **27.93** | 27.81 | 27.70 | 27.65 | 27.62 | 27.53 | 26.12 | 27.05 |
| Hill | 29.80 | 29.77 | 29.71 | 29.90 | **29.96** | 29.85 | 29.88 | 29.83 | 29.62 | 28.67 | 29.61 |
| House | 32.90 | 32.84 | 32.66 | **33.17** | 32.96 | 32.86 | 32.57 | 33.13 | 32.40 | 31.16 | 32.73 |
| Lena | **32.27** | 32.12 | 32.14 | 32.26 | 32.23 | 32.08 | 32.26 | 31.86 | 31.79 | 30.41 | 31.91 |
| Man | 29.71 | 29.68 | 29.62 | 29.75 | 29.81 | 29.62 | **29.89** | 29.70 | 29.62 | 28.60 | 29.34 |
| Montage | **33.08** | 32.76 | 32.61 | 32.35 | 32.97 | 32.37 | 32.04 | 32.25 | 31.97 | 30.68 | 30.96 |
| Pepper | **30.46** | 30.37 | 30.29 | 30.40 | 30.43 | 30.16 | 30.31 | 30.23 | 30.12 | 28.69 | 29.64 |
| Median rank | 4 | 6 | 7 | 2 | 2 | 5 | 4 | 5 | 9 | 11 | 10 |

$\sigma = 25$

| Grayscale | DDID2 | PID | DDID | SAIST | SAPCA | BM3D | MLP | LSSC | NLB | NLM | PLOW |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Barbara | 28.59 | 28.38 | 28.51 | 28.62 | **28.68** | 27.99 | n/a | 28.17 | 28.07 | 26.65 | 28.10 |
| Boats | 27.75 | 27.71 | 27.65 | 27.62 | **27.92** | 27.74 | n/a | 27.77 | 27.39 | 26.27 | 27.63 |
| Cameraman | 27.55 | **27.60** | 27.32 | 27.29 | 27.57 | 27.18 | n/a | 27.34 | 27.17 | 26.49 | 26.66 |
| Couple | 27.39 | 27.40 | 27.30 | 27.33 | **27.58** | 27.48 | n/a | 27.41 | 27.18 | 25.66 | 27.33 |
| Finger Print | 25.15 | 24.98 | 25.04 | **25.55** | 25.54 | 25.30 | n/a | 25.30 | 25.39 | 24.07 | 25.21 |
| Hill | 27.93 | 27.92 | 27.83 | 27.87 | **28.08** | 27.99 | n/a | 28.00 | 27.75 | 26.45 | 27.89 |
| House | 30.63 | 30.76 | 30.41 | **31.38** | 30.75 | 30.65 | n/a | 31.10 | 30.26 | 28.83 | 30.55 |
| Lena | **30.22** | 30.14 | 30.07 | 30.03 | 30.10 | 29.86 | n/a | 29.91 | 29.81 | 28.23 | 29.86 |
| Man | 27.67 | 27.66 | 27.60 | 27.58 | **27.83** | 27.65 | n/a | 27.64 | 27.51 | 26.35 | 27.52 |
| Montage | **30.26** | 30.25 | 29.82 | 29.50 | 30.02 | 29.52 | n/a | 29.43 | 29.11 | 27.67 | 27.90 |
| Pepper | **28.10** | **28.10** | 27.94 | 27.94 | **28.10** | 27.70 | n/a | 27.86 | 27.69 | 25.73 | 27.50 |
| Median rank | 3 | 3 | 6 | 6 | 1 | 5 | n/a | 4 | 9 | 10 | 7 |

$\sigma = 40$

| Color | DDID2 | PID | DDID | BM3D | NLB | NLM | PLOW |
|---|---|---|---|---|---|---|---|
| Baboon | 26.29 | 26.12 | 26.17 | 25.95 | **26.53** | 25.65 | 24.22 |
| F-16 | **33.06** | 33.02 | 32.88 | 32.78 | 33.03 | 31.31 | 30.98 |
| House | 32.88 | 32.90 | 32.69 | **33.03** | 32.65 | 31.39 | 31.92 |
| Kodak 1 | **29.29** | 29.18 | 29.09 | 29.13 | **29.29** | 27.49 | 25.68 |
| Kodak 2 | **32.49** | 32.40 | 32.29 | 32.44 | 32.28 | 30.69 | 29.86 |
| Kodak 3 | **34.72** | 34.70 | 34.55 | 34.54 | 34.49 | 32.33 | 30.46 |
| Kodak 12 | 33.64 | 33.55 | 33.46 | **33.76** | 33.37 | 31.63 | 30.00 |
| Lake | 28.99 | 28.93 | 28.85 | 28.68 | **29.10** | 28.30 | 27.60 |
| Lena | **32.45** | 32.41 | 32.30 | 32.27 | 32.25 | 30.88 | 31.01 |
| Pepper | **31.40** | 31.36 | 31.25 | 31.20 | 31.23 | 30.28 | 30.35 |
| Tiffany | **32.65** | 32.61 | 32.49 | 32.23 | 32.37 | 31.13 | 31.42 |
| Median rank | 1 | 3 | 4 | 4 | 4 | 6 | 7 |

$\sigma = 25$

| Color | DDID2 | PID | DDID | BM3D | NLB | NLM | PLOW |
|---|---|---|---|---|---|---|---|
| Baboon | 24.33 | 24.29 | 24.19 | 23.87 | **24.50** | 23.22 | 22.56 |
| F-16 | 31.07 | **31.09** | 30.84 | 30.25 | 30.87 | 28.61 | 29.05 |
| House | 31.25 | **31.35** | 30.93 | 30.60 | 30.85 | 29.05 | 30.05 |
| Kodak 1 | **26.97** | 26.91 | 26.77 | 26.59 | 27.84 | 24.76 | 24.32 |
| Kodak 2 | **30.66** | 30.60 | 30.46 | 30.30 | 30.24 | 28.36 | 28.22 |
| Kodak 3 | 32.42 | **32.46** | 32.22 | 31.57 | 32.08 | 29.96 | 28.45 |
| Kodak 12 | **31.69** | 31.65 | 31.46 | 31.31 | 31.26 | 29.41 | 28.02 |
| Lake | 27.39 | 27.36 | 27.23 | 26.88 | **27.43** | 26.04 | 26.02 |
| Lena | **30.87** | **30.87** | 30.68 | 30.11 | 30.48 | 28.90 | 29.26 |
| Pepper | **30.05** | 30.04 | 29.88 | 29.27 | 29.65 | 28.19 | 28.68 |
| Tiffany | 30.93 | **30.95** | 30.73 | 30.13 | 30.40 | 28.71 | 29.73 |
| Median rank | 2 | 2 | 3 | 5 | 4 | 6 | 7 |

$\sigma = 40$

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

same median rank). Our method matches PID in quality but requires only a third of the iterations without a separate guided DDID step. For grayscale images, DDID2 competes with the best denoisers, and only BM3D-SAPCA [10] provides a consistently better median rank over different noise levels. For color images, DDID2 and PID tend to outperform the previous state of the art, such as the work by Lebrun, Buades, and Morel [22], and DDID2 has the best median rank. Some algorithms like BM3D-SAPCA [10], however, are not available for color image denoising.

It is interesting to compare the results on denoising artifact removal using DDF from section 4.1 with the denoising quality of DDID2. In some cases, denoising using one of the third-party methods and a DDF artifact-removal step outperforms DDID2. More often this is the case when the third party method on its own outperforms DDID2, but in rare cases adding DDF helps a third-party method that is slightly inferior on its own to overtake DDID2. The differences in PSNR values are typically small, however, and it is hard to make a systematic argument when this happens.

## 5. Conclusions.

We have introduced dual-domain filtering (DDF), a generalization of the spatial bilateral filter (BF) by including a frequency domain filter. A key idea in DDF is to estimate noise in both the spatial and frequency domains using robust kernels. We have demonstrated that DDF can improve most images with denoising and compression artifacts. By using DDF iteratively, we also implemented a new state-of-the-art image denoiser, DDID2. The simplicity and quality of DDF suggest that it may have the potential to become a universal tool for image enhancement and restoration like the BF.

In this work we have empirically determined the choice of robust kernels and their parameters for different applications such as artifact removal or denoising. We have investigated the robustness of DDID2 denoising with respect to changes of its parameters and found that it performs well under changes of most parameter values, with only two parameters having a strong influence on the output quality. Nonetheless, a more principled approach for obtaining optimal parameter values and kernel functions would be preferable and an interesting avenue for future work. Finally, we would like to investigate whether DDF could also be exploited as a higher-quality alternative to conventional (joint-)bilateral filtering in other applications, such as upsampling, HDR tone mapping, or contrast adjustment and detail enhancement.

## Appendix. MATLAB code.

**Algorithm 1**

*MATLAB code of DDF for grayscale images.*

```
function E = DDF(z, r, flip, a, A, k, K)

    d = z - z(1+r, 1+r);                                    % (3.1)
    k = k(real(d).^2);                                      % (3.9)
    e = a * sum(sum(d .* k)) / sum(k(:));                   % (3.3)

    D = fft2(ifftshift((d - e) .* k));                      % (3.4)
    K = K(abs((D + conj(D(flip)))/2).^2 / sum(k(:).^2));    % (3.10)
    E = A * sum(sum(D .* K)) / numel(K);                    % (3.6)
end
```

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

**Algorithm 2**

*MATLAB code for removing artifacts in grayscale images.*

```
function x = deart(g, y, sigma2, r, sigma_s, gamma_r, gamma_f)

    [dy dx] = ndgrid(-r:r);
    flip    = circshift(reshape(numel(h):-1:1, size(h)), [1 1]);

    h = exp(- (dx.^2 + dy.^2) / (2 * sigma_s^2));
    k = @(d2) h .* exp(- d2 / (gamma_r * sigma2));          % (4.3)
    K = @(d2) max(0, 1 - d2 / (gamma_f * sigma2));          % (4.4)

    z = padarray(g + 1i * y, [r r], 'symmetric');
    f = @(b) DDF(b.data, r, flip, 1, 1, k, K);              % (4.2)
    n = nlfilter(z, size(h), f);
    x = imag(z - n(1+r:end-r, 1+r:end-r));                  % (4.1)
end
```

**Algorithm 3**

*MATLAB code for denoising grayscale images.*

```
function x = DDID2(y, sigma2)

    N       = 8;
    sigma_s = 13;
    gamma_r = 5.3 / N;
    gamma_f = 13 / N;
    alpha   = exp(15);

    x = (1 + 1i) * y;                                       % (3.8)
    for n = N:-1:1, t = (n - 1) / N;

        S =    2    * sigma_s^2 * alpha^(-t/2);             % (4.6)
        T = gamma_r * sigma2    * alpha^t;                  % (4.7)
        V = gamma_f * sigma2;                               % (4.8)

        r       = max(4, round(2 * sqrt(S/2)));             % (4.12)
        [dy dx] = ndgrid(-r:r);
        flip    = circshift(reshape(numel(dx):-1:1, size(dx)), [1 1]);

        a = cos(t * pi/2);                                  % (4.11)
        h = exp(- (dx.^2 + dy.^2) / S);
        k = @(d2) cos(min(pi/2, sqrt(d2/(T*n)))).^n .* h;   % (4.9)
        K = @(D2) cos(min(pi/2, sqrt(D2/(V*n)))).^n;        % (4.10)

        f = @(b) DDF(b.data, r, flip, a, a, k, K);
        x = (1 + 1i) * y - imag(blockproc(x, [1 1], f, 'BorderSize', [r r], ...
          'PadMethod', 'symmetric', 'TrimBorder', 0, 'UseParallel', 1));    % (4.5)
    end
    x = real(x);
end
```

**Algorithm 4**

*MATLAB code of DDF for color images.*

```
function E = DDF_c(z, r, flip, a, A, k, K)

    d = bsxfun(@minus, z, z(1+r, 1+r, :));                  % (3.1)
    k = k(sum(real(d).^2, 3));                              % (3.9)
    e = a * sum(sum(bsxfun(@times, d, k))) / sum(k(:));     % (3.3)

    D = fft2(circshift(bsxfun(@times, ...                   % (3.4)
```

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

```
        bsxfun(@minus, d, e), k), -[r r]));
    K = K(abs((D + conj(D(flip)))/2).^2 / sum(k(:).^2));    % (3.10)
    E = A * sum(sum(D .* K)) / numel(k);                    % (3.6)
end
```

**Algorithm 5**

*MATLAB code for removing artifacts in color images.*

```
function x = deart_c(g, y, sigma2, r, sigma_s, gamma_r, gamma_f)

    [height width depth] = size(y);
    s = [height * width, depth];
    M = dctmtx(depth)';
    g = reshape(reshape(g, s) * M, size(y));
    y = reshape(reshape(y, s) * M, size(y));

    [dy dx] = ndgrid(-r:r);
    flip    = circshift(flipdim(reshape(numel(dx)*depth:-1:1, ...
              [size(dx) depth]), 3), [1 1]);

    h = exp(- (dx.^2 + dy.^2) / (2 * sigma_s^2));
    k = @(d2) h .* exp(- d2 / (gamma_r * sigma2));          % (4.3)
    K = @(d2) max(0, 1 - d2 / (gamma_f * sigma2));          % (4.4)

    f = @(b) DDF_c(b.data, r, flip, 1, 1, k, K);            % (4.2)
    x = y - imag(blockproc(g + 1i * y, [1 1], f, ...        % (4.1)
      'BorderSize', [r r], 'PadMethod',   'symmetric', ...
      'TrimBorder', 0,     'UseParallel', 1));

    x = reshape(reshape(x, s) / M, size(x));
end
```

**Algorithm 6**

*MATLAB code for denoising color images.*

```
function x = DDID2_c(y, sigma2)

    N       = 8;
    sigma_s = 13;
    gamma_r = 5.3 / N;
    gamma_f = 13 / N;
    alpha   = exp(15);

    [height width depth] = size(y);
    s = [height * width, depth];
    M = dctmtx(depth)';
    y = reshape(reshape(y, s) * M, size(y));

    x = (1 + 1i) * y;                                       % (3.8)
    for n = N:-1:1, t = (n - 1) / N;

        S =    2    * sigma_s^2 * alpha^(-t/2);             % (4.6)
        T = gamma_r * sigma2    * alpha^t;                  % (4.7)
        V = gamma_f * sigma2;                               % (4.8)

        r       = max(4, round(2 * sqrt(S/2)));             % (4.12)
        [dy dx] = ndgrid(-r:r);
        flip    = circshift(flipdim(reshape(numel(dx)*depth:-1:1, ...
                  [size(dx) depth]), 3), [1 1]);

        a = cos(t * pi/2);                                  % (4.11)
        h = exp(- (dx.^2 + dy.^2) / S);
```

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

```
        k = @(d2) cos(min(pi/2, sqrt(d2/(T*n)))).^n .* h;  % (4.9)
        K = @(D2) cos(min(pi/2, sqrt(D2/(V*n)))).^n;       % (4.10)

        f = @(b) ddf_c(b.data, r, flip, a, a, k, K);
        x = (1 + 1i) * y - imag(blockproc(x, [1 1], f, 'BorderSize', [r r], ...
          'PadMethod', 'symmetric', 'TrimBorder', 0, 'UseParallel', 1));    % (4.5)
    end
    x = real(x);

    x = reshape(reshape(x, s) / M, size(x));
end
```

**Acknowledgment.** We thank Charis Tsevis for allowing us to use his artwork in Figure 8.

## REFERENCES

[1] N. Azzabou, N. Paragios, and F. Guichard, *Image denoising based on adapted dictionary computation*, in Proceedings of the IEEE International Conference on Image Processing (ICIP 2007), IEEE, Piscataway, NJ, 2007, Vol. 3, pp. III-109–III-112.

[2] P. Bouboulis, K. Slavakis, and S. Theodoridis, *Adaptive kernel-based image denoising employing semi-parametric regularization*, IEEE Trans. Image Process., 19 (2010), pp. 1465–1479.

[3] A. Buades, B. Coll, and J. M. Morel, *A review of image denoising algorithms, with a new one*, Multiscale Model. Simul., 4 (2005), pp. 490–530.

[4] H. C. Burger, C. J. Schuler, and S. Harmeling, *Image denoising: Can plain neural networks compete with BM3D?*, in Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Piscataway, NJ, 2012, pp. 2392–2399.

[5] L. Caraffa, J.-P. Tarel, and P. Charbonnier, *The guided bilateral filter: When the joint/cross bilateral filter becomes robust*, IEEE Trans. Image Process., 24 (2015), pp. 1199–1208.

[6] P. Chatterjee and P. Milanfar, *A generalization of non-local means via kernel regression*, in Proc. SPIE 6814, SPIE, Bellingham, WA, 2008, 68140P.

[7] P. Chatterjee and P. Milanfar, *Clustering-based denoising with locally learned dictionaries*, IEEE Trans. Image Process., 18 (2009), pp. 1438–1451.

[8] P. Chatterjee and P. Milanfar, *Patch-based near-optimal image denoising*, IEEE Trans. Image Process., 21 (2012), pp. 1635–1649.

[9] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *Image denoising by sparse 3-d transform-domain collaborative filtering*, IEEE Trans. Image Process., 16 (2007), pp. 2080–2095.

[10] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *BM3D image denoising with shape-adaptive principal component analysis*, in Proceedings of the Workshop on Signal Processing with Adaptive Sparse Structured Representations (SPARS'09), 2009.

[11] A. Dauwe, B. Goossens, H. Q. Luong, and W. Philips, *A fast non-local image denoising algorithm*, in Proc. SPIE 6812, SPIE, Bellingham, WA, 2008, 681210.

[12] W. Dong, G. Shi, and X. Li, *Nonlocal image restoration with bilateral variance estimation: A low-rank approach*, IEEE Trans. Image Process., 22 (2013), pp. 700–711.

[13] F. Durand and J. Dorsey, *Fast bilateral filtering for the display of high-dynamic-range images*, ACM Trans. Graphics (TOG), 21 (2002), pp. 257–266.

[14] M. Elad, *On the origin of the bilateral filter and ways to improve it*, IEEE Trans. Image Process., 11 (2002), pp. 1141–1151.

[15] M. Elad and M. Aharon, *Image denoising via sparse and redundant representations over learned dictionaries*, IEEE Trans. Image Process., 15 (2006), pp. 3736–3745.

[16] A. Foi, V. Katkovnik, and K. Egiazarian, *Pointwise shape-adaptive DCT for high-quality denoising and deblocking of grayscale and color images*, IEEE Trans. Image Process., 16 (2007), pp. 1395–1411.

[17] S. Grewenig, S. Zimmer, and J. Weickert, *Rotationally invariant similarity measures for nonlocal image denoising*, J. Vis. Commun. Image Represent., 22 (2011), pp. 117–130.

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

[18] K. He, J. Sun, and X. Tang, *Guided image filtering*, IEEE Trans. Pattern Anal. Machine Intell., 35 (2013), pp. 1397–1409.
[19] C. Kervrann and J. Boulanger, *Optimal spatial adaptation for patch-based image denoising*, IEEE Trans. Image Process., 15 (2006), pp. 2866–2878.
[20] C. Knaus and M. Zwicker, *Dual-domain image denoising*, in Proceedings of the 20th IEEE International Conference on Image Processing (ICIP 2013), IEEE, Piscataway, NJ, 2013, pp. 440–444.
[21] C. Knaus and M. Zwicker, *Progressive image denoising*, IEEE Trans. Image Process., 23 (2014), pp. 3114–3125.
[22] M. Lebrun, A. Buades, and J. M. Morel, *A nonlocal Bayesian image denoising algorithm*, SIAM J. Imaging Sci., 6 (2013), pp. 1665–1688.
[23] M. Mahmoudi and G. Sapiro, *Fast image and video denoising via nonlocal means of similar neighborhoods*, IEEE Signal Process. Lett., 12 (2005), pp. 839–842.
[24] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, *Non-local sparse models for image restoration*, in Proceedings of the 12th IEEE International Conference on Computer Vision (ICCV), IEEE, Piscataway, NJ, 2009, pp. 2272–2279.
[25] V. K. Nath, D. Hazarika, and A. Mahanta, *Blocking artifacts reduction using adaptive bilateral filtering*, in Proceedings of the 2010 International Conference on Signal Processing and Communications (SPCOM), IEEE, Piscataway, NJ, 2010, pp. 1–5.
[26] J. Orchard, M. Ebrahimi, and A. Wong, *Efficient nonlocal-means denoising using the SVD*, in Proceedings of the 15th IEEE International Conference on Image Processing (ICIP 2008), IEEE Press, Piscataway, NJ, 2008, pp. 1732–1735.
[27] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, *Digital photography with flash and no-flash image pairs*, ACM Trans. Graph., 23 (2004), pp. 664–672.
[28] G. Peyré, *Image processing with nonlocal spectral bases*, Multiscale Model. Simul., 7 (2008), pp. 703–730.
[29] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli, *Image denoising using scale mixtures of Gaussians in the wavelet domain*, IEEE Trans. Image Process., 12 (2003), pp. 1338–1351.
[30] L. Shao, R. Yan, X. Li, and Y. Liu, *From heuristic optimization to dictionary learning: A review and comprehensive comparison of image denoising algorithms*, IEEE Trans. Cybernet., 44 (2014), pp. 1001–1013.
[31] H. Takeda, S. Farsiu, and P. Milanfar, *Kernel regression for image processing and reconstruction*, IEEE Trans. Image Process., 16 (2007), pp. 349–366.
[32] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in Proceedings of the 6th International Conference on Computer Vision (ICCV), IEEE, Piscataway, NJ, 1998, pp. 839–846.
[33] D. Van De Ville and M. Kocher, *SURE-based non-local means*, IEEE Signal Process. Lett. 16 (2009), pp. 973–976.
[34] D. Van De Ville and M. Kocher, *Nonlocal means with dimensionality reduction and SURE-based parameter selection*, IEEE Trans. Image Process., 20 (2011), pp. 2683–2690.
[35] G. Yu and G. Sapiro, *DCT image denoising: A simple and effective image denoising algorithm*, IPOL Journal. Image Processing On Line, 1 (2011), http://dx.doi.org/10.5201/ipol.2011.ys-dot.
[36] M. Zhang and B. K. Gunturk, *Compression artifact reduction with adaptive bilateral filtering*, in Visual Communications and Image Processing 2009, Vol. 7257, SPIE, Bellingham, WA, 2009.
[37] Q. Zhang, X. Shen, L. Xu, and J. Jia, *Rolling guidance filter*, in Proceedings of the 13th European Conference on Computer Vision (ECCV 2014), Lecture Notes in Comput. Sci. 8691, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds., Springer, New York, 2014, pp. 815–830.

Downloaded 06/08/16 to 130.92.9.55. Redistribution subject to CCBY license

© 2015 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license