APSIPA Transactions on Signal and Information Processing, 2022, 11, e10

Original Paper

Adjoint Bilateral Filter and Its Application to Optimization-based Image Processing

Keiichiro Shirai[1], Kenjiro Sugimoto[2] and Sei-ichiro Kamata[3]*

[1] *Shinshu University, Nagano-shi 380-8553, Japan*
[2] *Japan R&D Center, Xiaomi, PMOGOTANDA, Tokyo-to 141-0031, Japan*
[3] *Waseda University, Kitakyushu-shi 808-0135, Japan*

ABSTRACT

This study primarily presents efficient tools for optimization-based image processing using a bilateral filter (BF). Generally, for image restoration, e.g., deblurring, a forward operation and its adjoint operation pair are required to solve inverse problems via iterative approaches such as the gradient method. Image data comprise millions of variables; thus, the operations should be performed as image filters rather than matrix products because of the considerable matrix size. This approach is known as a matrix-free approach, i.e., filter form, because it is executed without explicitly generating an enormous matrix. When BF is incorporated into optimization, its matrix-free adjoint BF is required to solve the optimization problem. This study discusses the matrix-free adjoint BF and its constant-time algorithm to solve optimization problems in a practical time frame. The experimental results demonstrate that the proposed method yields sufficient filtering accuracy for solving inverse problems. Furthermore, BF-based optimization improves accuracy by adjusting the image quality of resultant images.

*Corresponding author: Keiichiro Shirai, keiichi@shinshu-u.ac.jp. Keiichiro Shirai and Kenjiro Sugimoto contributed equally to this work.

Received 03 September 2021; Revised 26 January 2022
ISSN 2048-7703; DOI 10.1561/116.00000046

1 Introduction

In image processing, computer vision, and computer graphics, a *bilateral filter* (BF) [1, 31, 38] is a common edge-preserving smoothing filter. Traditional low-pass linear filters, for example, Gaussian filters, determine filter weights (coefficients) from distance in a spatial domain; however, in addition to the spatial domain, the BF uses distance in the pixel intensity domain. Because of its simple concept and structure, BF has been extensively used for various applications, including *intensity tone mapping* [13], *demosaicing* [29], *abstraction* [40], and *optical flow estimation* [41].

In the last two decades, BF has been actively extended to achieve higher smoothing quality along with lower computational complexity. An extensively used extension for high-quality smoothing of noisy images is *cross filtering* (or *joint filtering*) [14, 26]. Cross BF can produce high-quality images compared to the original BF by determining filter weights from an additional guide image captured under different photographic conditions rather than noisy target images. Another algorithm to accelerate BF is the *constant-time* ($O(1)$) BF [6–8, 10, 11, 13, 25, 32–34, 42, 44]. Here, constant time indicates that the per-pixel computational complexity is independent of the filter window size, *i.e.*, it runs in $O(1)$ time per pixel. Regardless of filter window size, this technique allows us to perform BF in nearly the same time.

Recently, BF has played a role in an optimization tool to extract image structures [2–4, 18, 24, 30, 45]. In variational approaches, the desired image is characterized as a minimizer of the sum of a regularization term incorporating a priori knowledge of an image and a data fidelity term that evaluates image consistency. Existing methods incorporate the BF for characterizing the desired image as a tool to separately handle the structure and texture components of an image. This enables us to perform edge-preserving or texture-preserving image processing via optimization. Indeed, compared to conventional formalizations, this technique has demonstrated higher potential relative to resulting quality in certain tasks.

However, existing methods with optimization have faced severe difficulty when directly handling enormous matrices, which occupy vast memory space and incur significant computational time. For variational approaches, an N-pixel image is denoted as an N-dimensional vector, and a filter applied to the image is represented as a vector-matrix product with an $N \times N$ matrix. Indeed, this matrix comprises N^2 elements, which is enormous for image processing, e.g., a 1920×1080 grayscale image ($N = 2,073,600$) requires a matrix with $N^2 = 4,299,816,960,000$ elements. Thus, vector-matrix products in optimization are generally limited to operations that can be run without explicitly generating filter matrices. We refer to this approach as a *matrix-free* operation (inspired from [20]). Note that well-known matrix-free operations include various image filters, e.g., correlation and convolution, fast Fourier transform, wavelet transform, and sparse matrix products.

This paper proposes a matrix-free operation of adjoint BF, which we refer to as BF* in this study. For many optimizers, such as the gradient descent method, the iterative steps comprise certain matrix products (forward operators) and their *transposed* matrix products (adjoint operators). The matrix-free operation of the original BF is obvious. However, to our knowledge, matrix-free operation of the BF* has not been investigated to date; nevertheless, this task seems nontrivial. Although an optimization approach with BF was proposed previously [4], it did not discuss BF* and was limited to equations comprising ℓ_2-norm terms. Moreover, acceleration techniques for the matrix-free BF* has not been discussed to date, e.g., $O(1)$ BF*. Thus, it is necessary to develop efficient algorithms for the BF* to realize realistic computational time for BF-based optimization of image processing applications.

This study tackles this problem by deriving a matrix-free BF*. Our discussion begins from a pair of filters required in optimization, i.e., a (forward) filter and its adjoint filter, which correspond to correlation filtering and convolution filtering in linear filtering. Moreover, we extend this method to cross filtering for better smoothing quality and to $O(1)$ filtering for lower computational complexity. As demonstrated by experimental results, our proposed method can efficiently solve BF-based optimization problems and produce the same or comparable resulting images at acceptable computational complexity while consuming significantly less memory space.

Our primary contributions are summarized as follows: (1) We derive the matrix-free BF*, which has not been investigated in the literature to date. (2) We propose a matrix-free $O(1)$ BF*, which is designed based on some existing $O(1)$ BF. Note that preliminary work has been published previously [37], and, in this study, we complement the unclear points of the preliminary work, for example, the differences of boundary conditions between matrix-form and matrix-free operations.

2 Preliminaries

This section describes the fundamentals required to comprehend the importance of BF* and matrix-free BF* in image processing based on variational approaches using BF.

2.1 Motivation of Matrix-free Adjoint Operation

Here, consider the image processing of a grayscale image with N pixels. To facilitate understanding the importance of our work, we present a toy example of the simple inverse problem:

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 \right\}, \tag{1}$$

where the column vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ represent a latent image and observed image, respectively, $\mathbf{A} \in \mathbb{R}^{N \times N}$ represent a matrix that aligns all filter coefficients of a linear (low-pass) filter, and $\|\cdot\|_2$ represents the ℓ_2-norm of a vector. If we estimate latent image $\mathbf{x}$ in a least-squares manner, the analytical solution can be obtained as the pseudo inverse $\mathbf{x} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}$. In this equation, we observe the adjoint (transposed) operator $\mathbf{A}^\top$. In the following additional example, which works in a wider range of situations, the same problem can be solved by an iterative optimizer, e.g., the gradient descent method:

$$\begin{aligned} \mathbf{x}^{(t+1)} &\leftarrow \mathbf{x}^{(t)} - \tau \nabla f(\mathbf{x}^{(t)}) \\ &= \mathbf{x}^{(t)} - \tau \mathbf{A}^\top (\mathbf{A}\mathbf{x}^{(t)} - \mathbf{y}), \end{aligned} \tag{2}$$

where t is an iteration counter and τ is a step-size parameter that must be appropriately set. If we naively compute Equation (2), the filter matrix (linear operator) $\mathbf{A}$ and its transposed matrix (adjoint operator) $\mathbf{A}^\top$ must be explicitly generated in the iterative process. However, $\mathbf{A}$ and $\mathbf{A}^\top$ are enormous in size; thus, it is computationally expensive to directly handle both matrices. Therefore, matrix-free operations are required for image processing via optimization.

2.2 Bilateral Filter and Cross Bilateral Filter

BF [1, 31, 38] is an edge-preserving smoothing filter that determines filter coefficients from both the spatial and range (pixel intensity) distances of a target image. The cross BF [14, 26] is an extension that determines the range distance from a guide image captured under different photographic conditions rather than a target image. This technique can mitigate loss in quality when the target image is very noisy and contains unsharp or pseudo edges. The theoretical differences between the original and cross BF appear trivial; thus, we refer to both as *BF* in the following discussions. By appropriately switching target images and guide images, BFs have played an important role in various smoothing tasks, for example, *color grading* [27, 28], *jaggy smoothing* for *image zooming* [22], and *depth maps* [23].

Here, consider the BF applied to a two-dimensional grayscale image x. Let $\mathbf{p} \in \mathbb{Z}^2$ indicate the position of a target pixel and $x_\mathbf{p} \in \mathbb{R}^1$ indicate its pixel intensity, and let $\mathbf{q} \in \mathcal{N}_\mathbf{p}$ indicate a neighboring pixel of $\mathbf{p}$, where $\mathcal{N}_\mathbf{p} \subset \mathbb{Z}^2$ is the set of neighboring pixels at $\mathbf{p}$, i.e., a rectangular domain supported in a filter window. Using these notations, BF is defined as follows:

$$\mathrm{BF}(\mathbf{p}) := \frac{\sum_{\mathbf{q} \in \mathcal{N}_\mathbf{p}} w_s(\mathbf{p}, \mathbf{q}) \cdot w_r(g_\mathbf{p}, g_\mathbf{q}) \cdot x_\mathbf{q}}{\sum_{\mathbf{q} \in \mathcal{N}_\mathbf{p}} w_s(\mathbf{p}, \mathbf{q}) \cdot w_r(g_\mathbf{p}, g_\mathbf{q})}. \tag{3}$$

where $g_\mathbf{p}$ is a target pixel in the guide image and $g_\mathbf{q}$ is one of its neighboring pixels. This definition is identical to the original BF when it is self-guided, i.e., $g_{\mathbf{p}} = x_{\mathbf{p}}$ and $g_{\mathbf{q}} = x_{\mathbf{q}}$. The former weight $w_s(\mathbf{p}, \mathbf{q})$ defined on the pixel position domain is referred to as the *spatial kernel*, and the latter weight $w_r(g_{\mathbf{p}}, g_{\mathbf{q}})$ defined on the pixel intensity domain is referred to as the *range kernel*. Generally, both are set as an even-symmetric unimodal function, for example, a Gaussian or Lorentzian function. The most common choice is the Gaussian kernel as follows:

$$\begin{aligned} w_s(\mathbf{p}, \mathbf{q}) &:= \exp\left(-\frac{1}{2\sigma_s^2}\|\mathbf{p}-\mathbf{q}\|_2^2\right), \\ w_r(g_{\mathbf{p}}, g_{\mathbf{q}}) &:= \exp\left(-\frac{1}{2\sigma_r^2}(g_{\mathbf{p}}-g_{\mathbf{q}})^2\right). \end{aligned} \tag{4}$$

As shown in Equation (3), the BF smooths a target image using only the neighboring pixels with similar intensity values to that of the target pixel $x_{\mathbf{p}}$. This simple concept can prevent smoothing edges from corresponding to those of the guide image because the intensities or colors around edge regions drastically change.

3 Proposed Methods

This section proposed the matrix-free BF* and its $O(1)$ algorithm.

3.1 Matrix-free Adjoint Bilateral Filter

In existing methods based on variational approaches [2, 24], BF and BF* were implemented as naive, matrix-form operations (rather than matrix-free operations). This naive technique is technically easy to implement; however, it is impractical because of its enormous matrix size, which requires significant memory space and incurs expensive computation. Consequently, it is necessary to develop a matrix-free BF*. Unfortunately, unlike linear filters (such as Gaussian filters), this task appears to be nontrivial.

Here, we mathematically derive the matrix-free BF*. For simplicity, we first introduce the next compound kernel:

$$w_{sr}(\mathbf{p}, \mathbf{q}) := w_s(\mathbf{p}, \mathbf{q}) \cdot w_r(g_{\mathbf{p}}, g_{\mathbf{q}}), \tag{5}$$

Let $\mathbf{B}_g \in \mathbb{R}^{N \times N}$ be a filter matrix of the BF, where the subscript g indicates the guide image to be used. The (i, j)-th element of $\mathbf{B}_g$ is then determined as follows:

$$[\mathbf{B}_g]_{i,j} := \begin{cases} \dfrac{w_{sr}(\mathbf{p}_i, \mathbf{p}_j)}{\sum_{\mathbf{v} \in \mathcal{N}_{\mathbf{p}_i}} w_{sr}(\mathbf{p}_i, \mathbf{v})} & \text{if } \mathbf{p}_j \in \mathcal{N}_{\mathbf{p}_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

where $[\cdot]_{i,j}$ denotes the (i,j)-th element of a matrix. Using this filter matrix and target image $\mathbf{x}$, the BF can be represented as $\mathbf{B}_g\mathbf{x} \in \mathbb{R}^N$, and BF* can be given as $\mathbf{B}_g^\top \mathbf{x} \in \mathbb{R}^N$ in a linear algebra manner. Before deriving the matrix-free BF*, we separate Equation (6) into a denominator and numerator using a matrix $\mathbf{W}_g \in \mathbb{R}^{N\times N}$ comprising the unnormalized filter weights:

$$[\mathbf{W}_g]_{i,j} := \begin{cases} w_{sr}(\mathbf{p}_i, \mathbf{p}_j) & \text{if } \mathbf{p}_j \in \mathcal{N}_{\mathbf{p}_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

If we extensively redefine the division operator as an element-wise operator between two vectors or two matrices, the BF Equation (3) can be rewritten as follows:

$$\mathbf{B}_g\mathbf{x} = \frac{\mathbf{W}_g\mathbf{x}}{\mathbf{W}_g\mathbf{1}} = \left(\frac{\mathbf{W}_g}{\mathbf{W}_g\mathbf{1}\mathbf{1}^\top}\right)\mathbf{x}, \tag{8}$$

where $\mathbf{1} \in \mathbb{R}^N$ is a column all-one vector. Note that $\mathbf{W}_g\mathbf{1}\mathbf{1}^\top \in \mathbb{R}^{N\times N}$ is a matrix whose columns all are $\mathbf{W}_g\mathbf{1} \in \mathbb{R}^N$ because each element of $\mathbf{W}_g\mathbf{1}$ indicates the sum of the filter weights in each local window region. Then, the matrix-free BF* can be derived from Equation (8) as follows:

$$\mathbf{B}_g^\top\mathbf{x} = \left(\frac{\mathbf{W}_g^\top}{\mathbf{1}\mathbf{1}^\top\mathbf{W}_g^\top}\right)\mathbf{x} = \mathbf{W}_g^\top\left(\frac{\mathbf{x}}{\mathbf{W}_g\mathbf{1}}\right). \tag{9}$$

We use the following formula that we found:

$$\left(\frac{\mathbf{A}}{\mathbf{1}\mathbf{c}^\top}\right)\mathbf{x} = \mathbf{A}\left(\frac{\mathbf{x}}{\mathbf{c}}\right), \tag{10}$$

where $\mathbf{A} := \mathbf{W}_g^\top$ and $\mathbf{c} := \mathbf{W}_g\mathbf{1}$.

Proof. $\mathbf{A} \in \mathbb{R}^{M\times N}$, $\mathbf{c} \in \mathbb{R}^N$, and $\mathbf{x} \in \mathbb{R}^N$. Here, both sides are M-dimensional column vectors; thus, we can rewrite the m-th element of the left-hand side to that of the right-hand side as follows:

$$\begin{aligned} \left[\left(\frac{\mathbf{A}}{\mathbf{1}\mathbf{c}^\top}\right)\mathbf{x}\right]_m &= \sum_n \left[\frac{\mathbf{A}}{\mathbf{1}\mathbf{c}^\top}\right]_{m,n}[\mathbf{x}]_n \\ &= \sum_n \left(\frac{[\mathbf{A}]_{m,n}}{[\mathbf{1}\mathbf{c}^\top]_{m,n}}\right)[\mathbf{x}]_n = \sum_n \frac{[\mathbf{A}]_{m,n}[\mathbf{x}]_n}{[\mathbf{1}]_m[\mathbf{c}]_n} \\ &= \sum_n \frac{[\mathbf{A}]_{m,n}[\mathbf{x}]_n}{[\mathbf{c}]_n} = \sum_n [\mathbf{A}]_{m,n}\left(\frac{[\mathbf{x}]_n}{[\mathbf{c}]_n}\right) \\ &= \sum_n [\mathbf{A}]_{m,n}\left[\frac{\mathbf{x}}{\mathbf{c}}\right]_n = \left[\mathbf{A}\left(\frac{\mathbf{x}}{\mathbf{c}}\right)\right]_m \end{aligned} \tag{11}$$

where $[\cdot]_{m,n}$ and $[\cdot]_m$ denote the (m,n) element of a matrix and m-th element of a (column) vector, respectively. □

Similar to the original BF Equation (3) and its matrix form Equation (8), we can obtain the matrix-free BF* from Equation (9) as follows:

$$\mathrm{BF}^*(\mathbf{p}) := \sum_{\mathbf{q}\in\mathcal{N}_\mathbf{p}} w_{sr}(\mathbf{p},\mathbf{q}) \left(\frac{x_\mathbf{q}}{\sum_{\mathbf{v}\in\mathcal{N}_\mathbf{q}} w_{sr}(\mathbf{q},\mathbf{v})} \right), \tag{12}$$

where we employ the symmetry $\mathbf{W}_g = \mathbf{W}_g^\top$ because $w_{sr}(\mathbf{p},\mathbf{q}) = w_{sr}(\mathbf{q},\mathbf{p})$ for even-symmetric kernels. From Equations (8) and (9), the BF and BF* differ in that the BF finally divides the resulting image of $\mathbf{W}_g\mathbf{x}$ by $\mathbf{W}_g\mathbf{1}$; however, BF* divides the target image of $\mathbf{W}_g\mathbf{x}$ by $\mathbf{W}_g\mathbf{1}$.

3.2 Boundary Padding

In practical image processing implementations, we must incorporate boundary padding into the matrix-form BF Equation (8) and matrix-form BF* Equation (9). This indicates that the image boundary must be appropriately expanded for a protruding filter window, for example, by zero padding, replication padding, and odd symmetric padding. To fill the gap between matrix-form and matrix-free operations with boundary padding, we complete the matrix-free BF* with appropriate boundary padding in a more practical form.

Matrix-form image filtering requires two boundary-operating matrices, i.e., the boundary padding and boundary trimming operators. For better understanding, we provide some examples of one-dimensional filtering. Let $\mathbf{x} \in \mathbb{R}^3$ be a target signal and $\mathbf{y} \in \mathbb{R}^5$ be a both-ends-padded signal. The boundary padding operator $\mathbf{P}$ produces a padded signal from the target signal by appropriately assuming the boundary condition. In a replication padding case, the operator can be computed as follows:

$$\mathbf{Px} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ x \\ y \\ z \\ z \end{pmatrix}. \tag{13}$$

The adjoint operator performs cumulative summation of the pixel values at the padded positions as follows:

$$\mathbf{P}^\top\mathbf{y} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} a+b \\ c \\ d+e \end{pmatrix}. \tag{14}$$

Then, the boundary trimming operator $\mathbf{Q}$ reduces the size of the padded signal to the original size by truncating the padded positions as follows:

$$\mathbf{Q}\mathbf{y} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} b \\ c \\ d \end{pmatrix}. \tag{15}$$

The adjoint operator of the boundary trimming operator simply works as zero padding as follows:

$$\mathbf{Q}^\top \mathbf{x} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ x \\ y \\ z \\ 0 \end{pmatrix}. \tag{16}$$

Obviously, all boundary operations are matrix-free operations because they are very sparse binary matrices.

Next, we consider a two-dimensional BF with a boundary operation in a matrix-form manner. Because of the size of a target image $H \times W$ and padding radius $r \in \mathbb{N}$, the boundary padding and boundary trimming operators are defined as follows:

$$\begin{aligned} \mathbf{P} &\in \{0,1\}^{(H+2r)(W+2r) \times HW}, \\ \mathbf{Q} &\in \{0,1\}^{HW \times (H+2r)(W+2r)}, \end{aligned} \tag{17}$$

Using the above boundary padding and trimming operators, we can decompose the unnormalized BF operation as $\mathbf{W}_g = \mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}$, where $\widehat{\mathbf{W}}_g$ indicates a unnormalized BF for expanded images of size $(H+2r) \times (W+2r)$. Using this representation, the matrix-form BF (Equation 8) should be reformulated with boundary padding as follows:

$$\mathbf{B}_g\mathbf{x} = \frac{\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{x}}{\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}} = \left(\frac{\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}}{\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}\mathbf{1}^\top} \right)\mathbf{x}. \tag{18}$$

Similarly, the matrix-form BF* Equation (9) can be redefined as follows:

$$\mathbf{B}_g^\top\mathbf{x} = \mathbf{P}^\top\widehat{\mathbf{W}}_g^\top\mathbf{Q}^\top \left(\frac{\mathbf{x}}{\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}} \right). \tag{19}$$

In the padding version, the symmetry $\widehat{\mathbf{W}}_g = \widehat{\mathbf{W}}_g^\top$ holds but $\mathbf{W}_g \neq \mathbf{W}_g^\top$. Note that all of the abovementioned matrix products in Equations (18) and (19) can be performed as filtering, binary permutation, and summation with low memory consumption.

3.3 Constant-time Adjoint BF

Here, we demonstrate how to reduce the per-pixel computational complexity of BF* from linear to constant time. The per-pixel computational complexity of Equation (12) is proportional to the filter window size, which is significantly high for iterative optimizers. To make per-pixel computational complexity more practical, we must design an $O(1)$ algorithm of the matrix-free BF*. For BF, this improvement has been discussed extensively [6–8, 10, 11, 13, 25, 32–34, 42, 44] to improve the performance tradeoffs between computation complexity and approximation accuracy. Here, we clarify that BF* can be extended to an $O(1)$ algorithm by applying certain concepts from the existing $O(1)$ BF.

Most existing $O(1)$ BFs share a general framework based on separability. Here, we consider the approximation of the range kernel by the separable form:

$$w_r(g_{\mathbf{p}}, g_{\mathbf{q}}) \approx \sum_{k=1}^{K} \phi_k(g_{\mathbf{p}}) \cdot \psi_k(g_{\mathbf{q}}), \tag{20}$$

where $\phi_k : \mathbb{R} \to \mathbb{R}$, $\psi_k : \mathbb{R} \to \mathbb{R}$, and K is an approximation order. Multiple functions for $\phi_k(\cdot)$ and $\psi_k(\cdot)$ have been previously proposed, e.g., trigonometric functions [6, 10, 34] for the Gaussian range kernel and a hat function [13] and low-rank approximation [11, 32, 33] for an arbitrary range kernel. If Equation (20) is substituted for the numerator in Equation (3), we obtain the following:

$$\begin{aligned} &\sum_{\mathbf{q}\in\mathcal{N}_{\mathbf{p}}} w_s(\mathbf{p},\mathbf{q}) \left\{ \sum_{k=1}^{K} \phi_k(g_{\mathbf{p}}) \cdot \psi_k(g_{\mathbf{q}}) \right\} x_{\mathbf{q}} \\ &= \sum_{k=1}^{K} \phi_k(g_{\mathbf{p}}) \left[\sum_{\mathbf{q}\in\mathcal{N}_{\mathbf{p}}} w_s(\mathbf{p},\mathbf{q}) \cdot \underbrace{\psi_k(g_{\mathbf{q}}) \cdot x_{\mathbf{q}}}_{x'_{\mathbf{q}}} \right]. \end{aligned} \tag{21}$$

The denominator of Equation (3) shows the same structure by replacing the rightmost $x_{\mathbf{q}}$ to by 1, i.e., $x'_{\mathbf{q}} := \phi_k(g_{\mathbf{q}})$. Note that the $[\cdot]$ part in Equation (21) is a linear filter with spatial kernel $w_s(\mathbf{p},\mathbf{q})$ to an intermediate image x' newly generated by a pixel-wise product, i.e., $x'_{\mathbf{q}} := \psi_k(g_{\mathbf{q}}) \cdot x_{\mathbf{q}}$. Certain spatial kernels, such as Gaussian kernels, can be efficiently convolved in $O(1)$ time per pixel via recursive approximation [12, 39] or truncated-cosine approximation [5, 34, 36]. Thus, the BF can be approximated by the weighted sum of K intermediate images generated by $O(1)$ filtering. Importantly, the proposed $O(1)$ BF* can use this framework because both the denominator and numerator of the matrix-free BF* Equation (12) show the nested structure of Equation (21). Thus, if we appropriately set the approximation order K, our $O(1)$ BF* produces the sufficiently accurate output of the naive BF*.

Algorithm 1 Standard BF

```
1: function BF$(\mathbf{x}, \mathbf{g})$                ▷ $\mathbf{x}$: target image, $\mathbf{g}$: guide image.
2:     $\mathbf{w} \leftarrow \mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}$        ▷ In a matrix-free and/or $O(1)$ manner.
3:     $\mathbf{t} \leftarrow \mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{x}$        ▷ In a matrix-free and/or $O(1)$ manner.
4:     $\mathbf{x} \leftarrow \frac{\mathbf{t}}{\mathbf{w}}$                        ▷ Element-wise division.
5:     return $\mathbf{x}$
6: end function
```

Algorithm 2 Adjoint bilateral filter (BF*)

```
1: function BF$^*(\mathbf{x}, \mathbf{g})$               ▷ $\mathbf{x}$: target image, $\mathbf{g}$: guide image.
2:     $\mathbf{w} \leftarrow \mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}$        ▷ In a matrix-free and/or $O(1)$ manner.
3:     $\mathbf{t} \leftarrow \frac{\mathbf{x}}{\mathbf{w}}$                        ▷ Element-wise division.
4:     $\mathbf{x} \leftarrow \mathbf{P}^\top\widehat{\mathbf{W}}_g\mathbf{Q}^\top\mathbf{t}$      ▷ In a matrix-free and/or $O(1)$ manner.
5:     return $\mathbf{x}$
6: end function
```

In Section 3.2, we did not consider whether the signal length is finite or infinite. Moreover, for simplicity, we handled $\mathbf{W}$ as a symmetric matrix. Here, we consider actual finite length signals, and $\mathbf{W}$ (including boundary processing) is assumed to be an asymmetric matrix. However, when representing $\mathbf{W} = \mathbf{Q}\widehat{\mathbf{W}}\mathbf{P}$, the symmetry $\widehat{\mathbf{W}} = \widehat{\mathbf{W}}^\top$ holds because the remaining part after trimming by $\mathbf{Q}$ is not influenced by whether $\widehat{\mathbf{W}}$ is symmetric.

3.4 Algorithm Procedure

We summarize the procedure of the existing BF and proposed BF* with boundary operations in the pseudocode given in Algorithms (1) and (2). In BF, the denominator image $\mathbf{w}$ is generated by $\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}$(line 2), the numerator image $\mathbf{t}$ is generated by $\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{x}$ in a matrix-free manner (line 3), and then we produce the resulting image by their element-wise division (line 4). However, in BF*, the denominator image $\mathbf{w}$ is generated by $\mathbf{Q}\widehat{\mathbf{W}}_g\mathbf{P}\mathbf{1}$ (line 2) and we generate the intermediate image $\mathbf{t}$ via element-wise division by $\mathbf{w}$ (line 3), and then the resulting image is generated by $\mathbf{P}^\top\widehat{\mathbf{W}}_g\mathbf{Q}^\top\mathbf{t}$ (recall $\widehat{\mathbf{W}}_g = \widehat{\mathbf{W}}_g^\top$) (line 4). For both these filters, we must perform all matrix products of $\widehat{\mathbf{W}}_g$ in a matrix-free manner, and we can compute them approximately as an $O(1)$ algorithm, as discussed in the preceding section.

We can reduce the computational time of BF and BF* in optimization by exploiting the following concepts. Although both filters comprise similar operations, they differ in that BF connects lines 2 and 3 in parallel; however, BF* connects lines 2 and 4 in serial. Thus, the filter coefficients can be reused in BF by simultaneously operating the coefficients but cannot BF*. Consequently, the computational time of BF* is slightly greater than that of BF; however, in the $O(1)$ case, the matrix products share some parts, e.g., $\phi_k(g_\mathbf{p})$ and $\psi_k(g_\mathbf{q})$ in Equation (21). Moreover, they are precomputable and can be considered as a few intermediate images. This reduces the computational time of BF*, which results in nearly the same filtering time as the existing BF.

4 Results and Discussion

Here, we confirm the fundamental performance of the proposed BF* and its $O(1)$ algorithm in several experiments conducted to evaluate computational time and approximation accuracy. The test image was the standard "Lenna" image (Figure 2), which is a 24-bit RGB color image (512×512 pixels). Each RGB channel was normalized in advance to $[0, 1]$ as 64-bit floating-point values, and we independently processed each RGB channel. Note that processing time depends only on image size (not pixel values). The test environment comprised an Intel Core i7-9700K @ 3.60 GHz CPU with 16 GB DDR4 @ 2,666 MHz main memory. We wrote all implementations in MATLAB R2019b with MEX (C++) compiled by Microsoft Visual Studio 2019.

The implementation and parameter setting details of the proposed methods are summarized as follows. Both the spatial and range kernels were set to Gaussian kernels in all cases. The proposed $O(1)$ BF* internally requires an $O(1)$ BF, and it was decomposed into an $O(1)$ Gaussian filter. We adopted the compressive BF [34] for the former and the $O(1)$ Gaussian filter based on DCT-5 [35] for the latter, where their tolerance parameters were set to $\varepsilon_{\mathrm{BF}} \in \{0.01, 0.1\}$ for the compressive BF and $\varepsilon_{\mathrm{GF}} = 0.5$ for the $O(1)$ Gaussian filter. The parameter $\varepsilon_{\mathrm{BF}}$ primarily affected the tradeoff between computational time and approximation accuracy, i.e., larger values yield faster computational time but lower accuracy. For the boundary padding in Equation (19), the compressive BF [34] uses replication padding in the original implementation: it is implicitly performed in its inside as $\mathbf{P}$ is replication padding and $\mathbf{Q}$ is trimming. However, we explicitly set for the BF* as $\mathbf{Q}^\top$ is zero padding and $\mathbf{P}^\top$ is cumulative summation, which somewhat increases the computational time.

4.1 Approximation Accuracy

We confirmed the approximation accuracy of the proposed $O(1)$ BF* against the naive BF* Equation (12), which is assumed to be the ideal output. We confirm that the matrix-form BF* and the proposed matrix-free BF* show

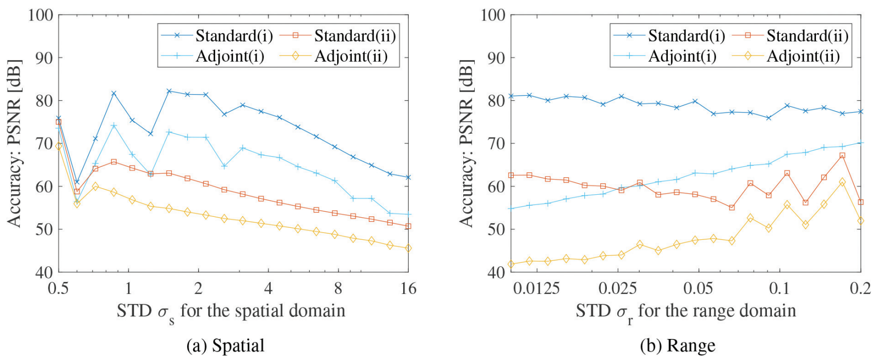


Figure 1: Approximation accuracy (PSNR [dB]) versus spatial scale σ_s and range scale σ_r.

equivalent results. The approximation accuracy is quantified as the PSNR [dB] between the output of the naive and $O(1)$ methods.[1]

Note that higher PSNR means less approximation error. Figure 1(a) shows the PSNR under gradually changing spatial scale σ_s with fixed range scale $\sigma_r = 0.1$, and Figure 1(b) shows the PSNR under gradually changing range scale σ_r with fixed spatial scale $\sigma_s = 2.0$. Moreover, Standard(i) and Adjoint(i) correspond to parameter $\varepsilon_{\mathrm{BF}} = 0.01$, and Standard(ii) and Adjoint(ii) correspond to $\varepsilon_{\mathrm{BF}} = 0.1$. For both these results, the approximation accuracy of the proposed $O(1)$ BF achieves at least 42 dB and higher than 50 dB at a small spatial scale. This indicates that the approximation accuracy is sufficient and the difference is visually imperceptible; thus, the proposed methods experimentally produce sufficient accuracy for multiple practical applications.

Furthermore, qualitative differences are an important evaluation factor. Figure 2 shows the input and resulting images of each implementation of BF*. On the basis of the results, we observe that the BF simply smooths an image. On the other hand, the proposed BF* appears to emphasize edges. Compared with (c) with (d), the approximation accuracy of the proposed $O(1)$ BF* exceeds 50 dB in PSNR. If we amplify the difference ten times, we can observe some errors around edges; however, the difference is practically negligible. In applications shown later, we show that the difference does not influence the performance of the applications.

[1]We compute the PSNR for color images as follows. Let $X_{i,j,k} \in [0,1]$ and $Y_{i,j,k} \in [0,1]$ be the pixel values of the two images, where (i,j) is the pixel position and k indicates a color channel index. Using the mean square error $\mathrm{MSE} = \frac{1}{N}\sum_{i,j,k}(X_{i,j,k} - Yi,j,k)^2$, we calculate the PSNR as $\mathrm{PSNR} = 10\log_{10}\left(\frac{\mathrm{MAX}}{\mathrm{MSE}}\right)$, where $\mathrm{MAX} = 1$ indicates the maximum value of the dynamic range.

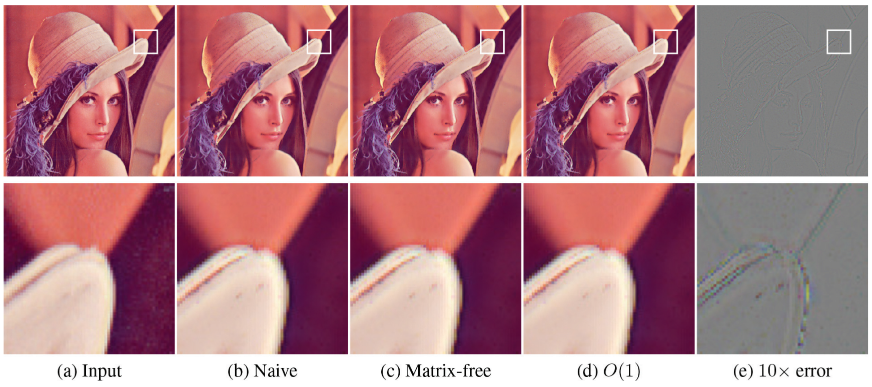


Figure 2: The top row shows the input "Lenna" image and corresponding images filtered by each BF*. The bottom row shows magnified images of the corresponding upper images, inside the white square frame. Here, $\sigma_s = 2.0$, $\sigma_r = 0.1$, $\varepsilon_{\mathrm{BF}} = 0.1$, and $\varepsilon_{\mathrm{GF}} = 0.5$. In (e), we demonstrate the difference between (c) and (d) amplified $10\times$ to facilitate visualization. The peak signal-to-noise ratio (PSNR) of the error was 53 dB.

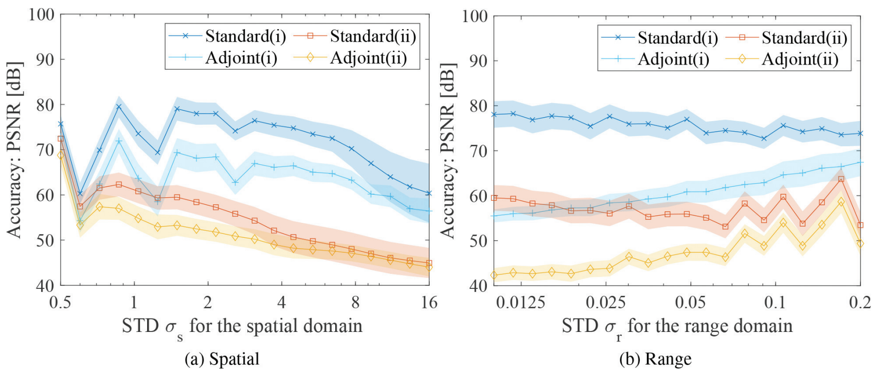
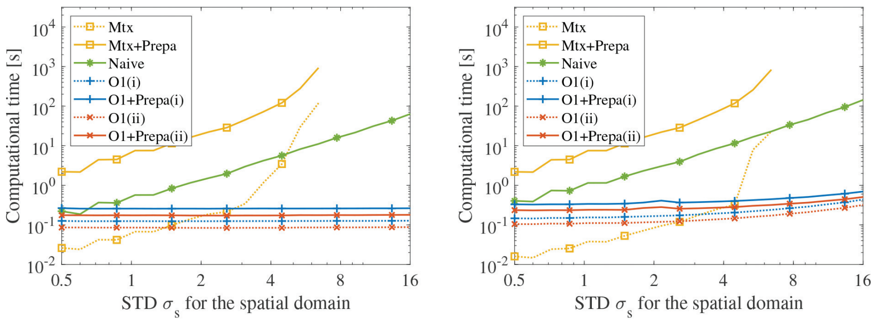


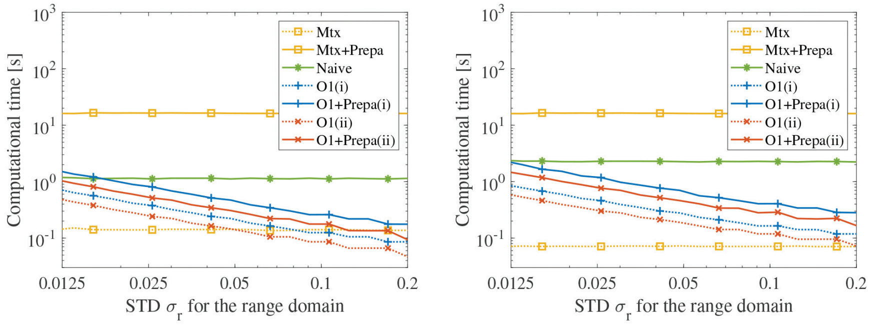
Figure 3: Mean values and standard deviations of the 24 images in the Kodak image data set: approximation accuracy (PSNR [dB]) versus spatial scale σ_s and range scale σ_r.

We also show the results using multiple images, i.e., a Kodak image data set [17] and its 24 images. Figure 3 shows the average result of each setting. The same parameters as in Figure 1 are used. The transition curves are the mean values of the 24 images. The belt-like light-colored region and its half the width in the vertical direction show the standard deviations of the 24 images, i.e., the transition curves obtained from the images mainly pass inside the region. One can see that the same tendencies as in Figure 1 are observed in Figure 3. This indicates that the approximation accuracy is sufficient and the difference is visually imperceptible.



(a) BF (b) BF*

Figure 4: Computational time versus spatial scale σ_s in log–log plot with range scale $\sigma_r = 0.1$.



(a) BF (b) BF*

Figure 5: Computational time versus range scale σ_r in log–log plot with spatial scale $\sigma_s = 2$.

4.2 Computational Complexity

Here, confirm the computational time of BF Equation (3), the proposed naive BF* Equation (12), and the proposed $O(1)$ BF*. Figures 4 and 5 show the relationship between computational time and the spatial scale σ_s and range scale σ_r of the Gaussian kernels, respectively. The parameter setting for $O(1)$ BF/BF* is $\varepsilon_{\mathrm{BF}} = 0.01$ and 0.1.

Figure 4 shows the computational time [s] for fixed $\sigma_r = 0.1$ and gradually changing σ_s. The domain of $1 \leq \sigma_s$ covers practical cases because the Gaussian shape with $\sigma_s < 1$ is non-negligibly distorted on discrete grids. Note that the filter window size for the Gaussian spatial kernel is commonly set to $M = 2\lceil 3\sigma_s \rceil + 1$ to support the Gaussian shape, e.g., $1 \leq \sigma_s$ corresponds to $7 \leq M$. This figure shows seven plots of the following three methods: (I) Mtx (Matrix) shows the time of matrix product of matrix-form BF/BF* and Mtx+Prepa (Matrix+Preparation) adds the time of explicitly generating three BF matrices for RGB colors to Mtx; (II) Naive corresponds to naive BF and the proposed matrix-free BF*; and (III) O1(i)/(ii) and O1+Prepa(i)/(ii) show the results of the $O(1)$ BF where Prepa (Preparation) indicates that the time of generating the common intermediate images. From these results, we observe that the Mtx/Mtx+Prepa and naive BF methods demonstrate per-pixel time that is proportional to $O(M)$ and $O(\sigma_s)$; however, the proposed $O(1)$ methods show obvious $O(1)$ per-pixel time. For the BF* case, the naive implementation is slightly slower than the BF because the BF coefficients must be computed twice (Section 3.4). Moreover, O1 and O1+Prepa demonstrate slightly longer times than the standard ones because the BF* must explicitly manage boundary operations. The fastest one around $\sigma_s \in [0.5, 1.5]$ is Mtx if the BF matrix can be physically allocated; however, $1.5 < \sigma_s$, the proposed $O(1)$ BF* outperforms the other methods. Note that Mtx causes a memory shortage around $\sigma_s = 6.5$ ($3 \leq \sigma_s$; it becomes slow because of a memory swap). Moreover, with significantly larger images, e.g., 2624×3936 pixels, the memory shortage occurs around $\sigma_s \in [1, 2]$ (Section 5.2).

Figure 5 shows computational time obtained with fixed $\sigma_s = 2.0$ and gradually changing σ_r. The domain $0.05 \leq \sigma_r \leq 0.2$ will cover most practical cases. Moreover, the plots show similar results to those given in Figure 4. As can be seen, the proposed $O(1)$ BF and BF* reduce computational time compared to other methods. Furthermore, the proposed $O(1)$ BF* runs considerably faster than our naive BF* for most situations.

4.3 Memory Usage

Here, we compare the memory usage of the matrix-form and matrix-free BF/BF*. Let M be the number of pixels in an image; N be the number of pixels in a filter window; C be the number of colors; and K be the number of terms used to approximate the BF in Equation (20).

Memory usage in the matrix-form BF Essentially, the matrix-form BF requires MNC nonzero elements and additional side information to represent the coordinates of the sparse matrix. When $M = 1000^2$ and $N = 11^2$ (corresponding to $\sigma_s = 1.5$), at least the following is required at double precision (8 [byte]):

$$\begin{aligned} MNC \times 8 \text{ [byte]} &= 1000^2 \times 11^2 \times 3 \times 8 \\ &\approx 2.7 \text{ GB } (2.7 \times 1024^3 \text{ [byte]}). \end{aligned}$$

The side information is mainly the position information of the nonzero elements, i.e., the indexes (coordinates) of the row and column, which is simply calculated as:

$$MNC \times 4 \text{ [byte]} \times 2 \text{ [row, col]} \approx 2.7 \text{ GB},$$

where 4 [byte] is the size of the integer type and 2 is the number of indexes for the row and column. The total theoretical memory size is

$$MNC \times (8 + 4 \times 2) \text{ [byte]} \approx 5.4 \text{ GB}. \tag{22}$$

Memory usage in the matrix-free BF The matrix-free BF developed based on Sugimoto and Kamata [34] requires $4MCK$ for the parameter images, i.e., $\phi_k(g_\mathbf{p})$, $\psi(g_\mathbf{q})$, and the extended versions in Equation (21), which can be precomputed and shared. As another example, if the parameters are $C = 3$ (RGB images), $\sigma_s = 1.5$, $\sigma_r = 0.1$, $\varepsilon_{\mathrm{BF}} = 0.01$, maximum dynamic range of 1.0, and $K = 6$, the memory usage for the images is

$$\begin{aligned} 4MCK \times 8 \text{ [byte]} &= 4 \times 1000^2 \times 3 \times 6 \times 8 \\ &\approx 550 \text{ MB}. \end{aligned} \tag{23}$$

The difference between the above calculations Equations (22) and (23) is N and $2K$, respectively, where $N = 11^2 = 121$ and $2K = 2 \times 6 = 12$, and precisely, $N \gg 2K$. Therefore, a difference of $\times 10$ appears in this case. When handling larger images, such as, 2K or 4K images that require a larger spatial scale in filtering, this difference becomes critical, and the matrix-form BF cannot handle these images.

The actual memory usage required for an application is thought of becoming larger than the mentioned theoretical one. In addition to the matrix or filter-form BF, the memory usage depends on the number of variables and temporary variables used in the code. In Section 5.2, we show an example of an actual memory usage in an application.

5 Applications to Optimization-based Image Processing

This section presents two applications based on variational approaches with BF and how the BF and BF* (as well as their $O(1)$ algorithms) are used in real-world situations. The second application is a conventional method [2]; however, the equation is more difficult than the first application. Therefore, for the first application, we introduce how to use BF/BF* and the definitions of variables. Then, for the second application, we demonstrate how the proposed methods improve performance.

5.1 Correction After Color Grading

The first example is a post process for color grading [27] that attempts to correct noisy resulting images after color grading by performing BF-based optimization.

5.1.1 Decomposition into Structure and Texture Components

Similar to the original BF [14, 26], a major example of the BF is decomposition of a target image using its guide image into a structure component of the target and texture component of the guide, which corresponds to a low-frequency part, for example, colors and edges, and a high-frequency part. Given a target image $\mathbf{y}$ and guide image $\mathbf{g}$, we can extract the structure component from the target image as $\mathbf{B}_g\mathbf{y}$ and the texture component from the guide image as $\mathbf{g} - \mathbf{B}_g\mathbf{g}$. We can treat this as a linear problem because the BF filter matrix $\mathbf{B}_g$ can be fixed throughout the process. By adding these components appropriately, we can transfer the texture of the guide image onto the structure of the input image. In particular, we obtain the following:

$$\mathbf{x} := \underbrace{\mathbf{B}_g\mathbf{y}}_{\text{structure}} + \lambda_1 \underbrace{\overline{\mathbf{B}}_g\mathbf{g}}_{\text{texture}}, \tag{24}$$

where $\overline{\mathbf{B}}_g = \mathbf{I} - \mathbf{B}_g$ is introduced for simplicity, $\mathbf{I}$ is the identity matrix, and λ_1 is a balancing parameter. However, if component extraction results in insufficient quality, this approach may demonstrate nonnegligible artifacts.

We avoid this drawback using a variational approach. Here, we consider an extension of the above naive approach to an optimization problem with BF; this extension can be formalized as follows:

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}) := \tfrac{1}{2} \underbrace{\|\mathbf{B}_g(\mathbf{x}-\mathbf{y})\|_2^2}_{\text{structure}} + \lambda_2 \tfrac{1}{2} \underbrace{\|\overline{\mathbf{B}}_g(\mathbf{x}-\mathbf{g})\|_2^2}_{\text{texture}} \right\}, \tag{25}$$

where λ_2 is a balancing parameter. In Equation (25), the first and second terms induce a solution that is similar to the structure component of the input image $\mathbf{y}$ and texture component of the guide image $\mathbf{g}$, respectively. This problem can be solved using an iterative optimizer, e.g., the gradient descent method Equation (2) or conjugate gradient method [15, 21]. The gradients required by this computation can be derived by considering the partial derivative of Equation (25) relative to $\mathbf{x}$ as follows:

$$\begin{aligned} \nabla f(\mathbf{x}^{(t)}) &= \mathbf{B}_g^\top \mathbf{B}_g(\mathbf{x}^{(t)} - \mathbf{y}) + \lambda_2 \overline{\mathbf{B}}_g^\top \overline{\mathbf{B}}_g(\mathbf{x}^{(t)} - \mathbf{g}) \\ &= \mathbf{B}_g^\top \mathbf{B}_g \mathbf{x}^{(t)} + \lambda_2 \overline{\mathbf{B}}_g^\top \overline{\mathbf{B}}_g \mathbf{x}^{(t)} \\ &\quad - \{\mathbf{B}_g^\top \mathbf{B}_g \mathbf{y} + \lambda_2 \overline{\mathbf{B}}_g^\top \overline{\mathbf{B}}_g \mathbf{g}\}. \end{aligned} \tag{26}$$

This model seems considerably more sophisticated than the naive model Equation (24); however, it is computationally expensive to naively iterate the computation of Equation (26).

We can reduce the computational burden of the abovementioned approach using the proposed methods. First, the computation of line 3 in Equation (26) can be moved outside the iteration process. It is precomputable because it is independent of the iteration counter t. The matrix product of $\mathbf{B}_g$ or $\mathbf{B}_g^\top$, which indicates BF or BF*, can be replaced by their associated filter operations in practice because this computation is performed from right to left as an image filter or its $O(1)$ algorithm. Moreover, we can efficiently compute $h(\mathbf{x}^{(t)}) := \mathbf{B}_g^\top \mathbf{B}_g \mathbf{x}^{(t)} + \lambda_2 \overline{\mathbf{B}}_g^\top \overline{\mathbf{B}}_g \mathbf{x}^{(t)}$, which is required for iterations in line 2 of Equation (26), as follows:

$$\left\lfloor\begin{aligned} \mathbf{u}^{(t)} &\leftarrow \mathbf{B}_g \mathbf{x}^{(t)}, \\ \mathbf{v}^{(t)} &\leftarrow \lambda_2(\mathbf{x}^{(t)} - \mathbf{u}^{(t)}), \\ h(\mathbf{x}^{(t)}) &= \mathbf{B}_g^\top(\mathbf{u}^{(t)} - \mathbf{v}^{(t)}) + \mathbf{v}^{(t)}, \end{aligned}\right. \tag{27}$$

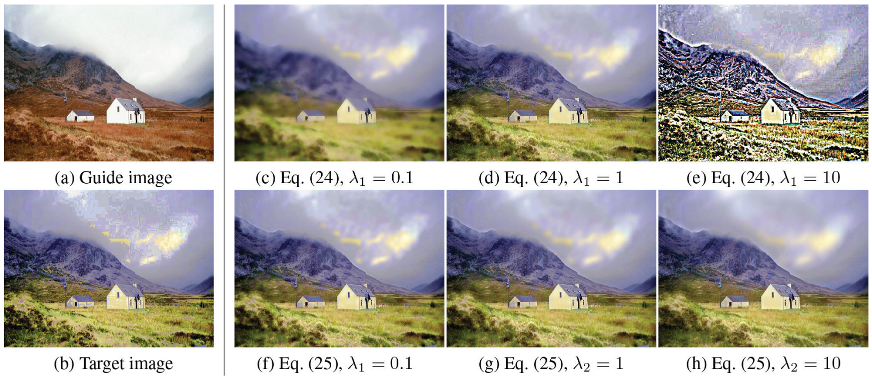
Thus, each iteration performs one BF and one BF*; however, this must be performed three times to process an RGB image.

Moreover, similar to lines 2 and 3 in Algorithm 1, we can reduce the computational time by sharing the generated filter coefficients because we use both BF and BF* in the same order for each term, and we can share the same guide image to compute the filter coefficients.

5.1.2 Results

The existing method based on Equation (24) was proposed in Rabin *et al.* [28]. Note that the previous study used (rather than the BF) Yaroslavsky *et al.* filter [43], which can be considered as a specific case of BF simplified using a box spatial kernel. In these experiments, we attempt to use BF. Here, both the spatial and range kernels were Gaussian kernels, and the scale parameters $\sigma_s = 4$ and $\sigma_r = 0.05$. The tolerance for the $O(1)$ BF and BF* was set to $\varepsilon_{\mathrm{BF}} = 0.1$. We solved Equation (25) using the conjugate gradient method [15, 21]. Moreover, the test environment and implementations were the same as those identified in Section 4. Here, we used an RGB test image (361×481 pixels).

Figure 6 (a) shows the target image before color grading, and Figure 6(b) shows the resulting image after color grading. In this task, we performed BF using (a) as the guide image $\mathbf{g}$ and (b) as the input image $\mathbf{y}$. We successfully obtained the desired colors of the structure component; however, an unexpected texture occurred as artifacts in the sky region. Thus, we aim to reduce this artifact by transferring the texture contained in the original image. Figures 6(c)$\sim$(h) show the resulting images of an existing approach based on the Rabin method (top row) and our optimization Equation (25) obtained with various balancing parameter values ($\lambda_1 = 0.1, 1, 10$ and $\lambda_2 = 0.1, 1, 10$). As can be seen, we successfully reduced the artifact in the sky region. Relative to parameter adjustments, the existing approach Equation (24) was significantly


(a) Guide image (c) Eq. (24), $\lambda_1 = 0.1$ (d) Eq. (24), $\lambda_1 = 1$ (e) Eq. (24), $\lambda_1 = 10$

(b) Target image (f) Eq. (25), $\lambda_1 = 0.1$ (g) Eq. (25), $\lambda_2 = 1$ (h) Eq. (25), $\lambda_2 = 10$

Figure 6: Examples of correcting results of color grading. The images in (a) and (b) are courtesy of Pitié *et al.* [27].

sensitive to the parameter λ_1, which directly affected the sharpness of the entire texture. However, the proposed approach Equation (25) (bottom row) can only affect the texture of the sky region, which is easier for controlling the parameter λ_2.

Here, the number of iterations required to converge to a visually similar result was four, and each iteration required 0.16 s. Thus, we consider that we have overcome the severe problem of the existing methods based on BF optimization [2, 24], which is caused by explicitly generating the BF filter matrix with numerous nonzero elements by replacing it with the proposed $O(1)$ filter operation.

5.2 *Flash/no-flash Photo Integration for Denoising*

The second example is a flash/no-flash image integration method [2], which was referred to as a conventional method in the Introduction. In this method, BF and BF* are used to integrate under-exposed regions.

Here, let $\mathbf{y}$ be a no-flash image, $\mathbf{g}$ be a flash image as the guide image, and $\mathbf{B}_g$ be the BF filter matrix generated using a flash image. The BF and BF* parameters are $\sigma_r = 0.05$ and $\varepsilon_{\mathrm{BF}} = 0.1$, and we varied σ_s from 1 to 4 for comparison. The integration problem for under-exposed regions is expressed as follows:

$$\min_{\mathbf{x}} \ \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda_3\|\overline{\mathbf{B}}_g\mathbf{x}\|_1, \tag{28}$$

where the first term is a data fidelity term used to transfer the color and contrast of the no-flash image to the solution $\mathbf{x}$; the second term is a regularizer comprising the Laplacian using BF $\overline{\mathbf{B}}_g\mathbf{x} := \mathbf{x} - \mathbf{B}_g\mathbf{x}$ and the ℓ_1 norm, i.e., total variation using BF rather than differential filters, to reduce noise on $\mathbf{x}$;

and λ_3 is a balancing parameter (set to $\lambda_3 = 0.1$). To solve this problem, the primal dual splitting [9] convex optimization method was used. Here, the iterative algorithm is derived as follows:

$$\left\lfloor \begin{aligned} \mathbf{x}^{(t+1)} &\leftarrow \mathbf{x}^{(t)} - \tau_1\{(\mathbf{x}^{(t)} - \mathbf{y}) + \overline{\mathbf{B}}_g^\top \mathbf{z}^{(t)}\}, \\ \mathbf{v}^{(t)} &\leftarrow \mathbf{z}^{(t)} + \tau_2\overline{\mathbf{B}}(2\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}), \\ \mathbf{z}^{(t+1)} &\leftarrow \mathbf{v}^{(t)} \frac{\lambda_3}{\max(\sqrt{(\mathbf{v}_R^{(t)})^2 + (\mathbf{v}_G^{(t)})^2 + (\mathbf{v}_B^{(t)})^2}, \lambda_3)}, \end{aligned} \right. \tag{29}$$

where $\mathbf{v}$ and $\mathbf{z}$ are temporal variables; the third row is element-wise computation of each vector; $\mathbf{v}_R$, $\mathbf{v}_G$, and $\mathbf{v}_B$ are the R, G, and B color components, respectively; and τ_1 and τ_2 are parameters that control the convergence ($\tau_1 = 0.1$ and $\tau_2 = 0.8$).

In the original method, the matrix form is used to calculate the BFs: $\overline{\mathbf{B}}_g^\top \mathbf{z}^{(t)}$ and $\overline{\mathbf{B}}(2\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$. However, our method uses the following matrix-free BF and BF*. In this case, the number of iterations for Equation (29) was set 300 to measure the sufficiently long computational time.

5.2.1 Results

First, we demonstrate qualitative results in Figure 7, where (from left to right) the images are (a) the flash image used as the guide, (b) the no-flash image, (c) the result from the matrix form approach (conventional method), (d) the result of the matrix-free approach (proposed method), and (e) the difference between (c) and (d) was magnified 10× times. The original image size was 427×640, and we show a partial region of size 300×270 to exhibit the details. The nearly same results were obtained from the matrix form (c) and matrix-free form (d), and their perceptual difference (e) is minimal even if the error is emphasized. The PSNR between (c) and (d) in the whole 427×640 region exceeds 60 dB; thus, the numerical difference is sufficiently small.

The computational times and PSNR for reference are listed in Table 1. In addition to the 427×640 images, 1152×1728 and 2624×3936 images taken with a Canon EOS20D camera. Here, the computational times for the matrix and matrix-free forms are shown (faster times are shown in bold). These results indicate that the advantage of the matrix form method is limited to when $\sigma_s < 2.0$, and it cannot handle large images and large kernels, i.e., high-resolution images, because of memory shortage ("out of memory" in the figure). However, the matrix-free BF* can process images in constant time regardless of the filter size.

The required memory size can be estimated by Equations (22) and (23). First, "out of mem." in Table 1 indicates that the actual memory usage

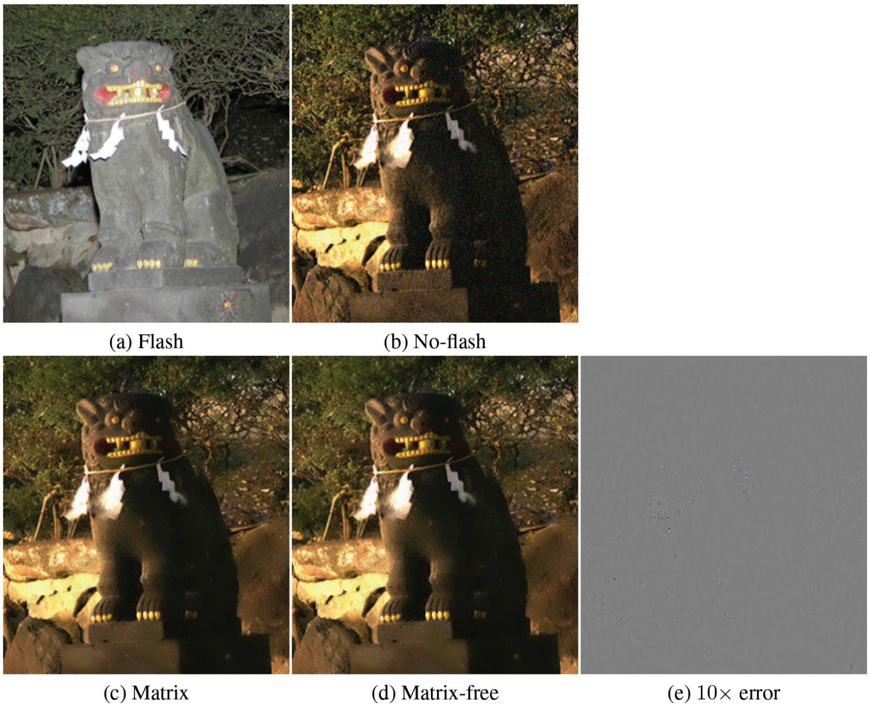


Figure 7: Examples of flash/no-flash image integration. Images (a) and (b) are courtesy of Baba *et al.* [2]. The PSNR of error (e) between matrix form (c) and matrix-free form (d) is 58 dB.

exceeded 16 GB that is mounted on the PC used for all the experiments through the paper. When using the 2624×3936 image and 13×13 filter, the theoretical memory usage for the matrix-form BF/BF* is estimated as follows using Equation (22):

$$\begin{aligned} MNC \times 16 \text{ [byte]} &= (2624 \times 3936) \times 13^2 \times 3 \times 16 \\ &\approx 78 \text{ GB}. \end{aligned}$$

On the other hand, the theoretical memory usage for the matrix-free BF/BF* is estimated as follows using Equation (23):

$$\begin{aligned} 4MCK \times 8 \text{ [byte]} &= 4 \times (2624 \times 3936) \times 3 \times 6 \times 8 \\ &\approx 5.5 \text{ GB}. \end{aligned}$$

Therefore, large-size images and large-size filter critically affect the computation using BF/BF*.

Table 1: Average computational time of BF and BF* in flash/no-flash image integration, and PSNR of resulting images between matrix form and matrix-free $O(1)$ methods for reference.

Image size	σ_s	Filter size	Computational time [sec/iteration]				PSNR [dB]
			BF		BF*		
			Matrix form	Filter form	Matrix form	Filter form	
427×640	1.0	7×7	**0.035**	0.141	**0.024**	0.174	62.4
	2.0	13×13	**0.113**	0.142	**0.070**	0.191	63.3
	3.0	19×19	0.234	**0.144**	**0.146**	0.213	60.2
	4.0	25×25	0.402	**0.144**	0.255	**0.231**	60.2
1152×1728	1.0	7×7	**0.281**	1.106	**0.185**	1.243	64.1
	2.0	13×13	14.455	**1.087**	5.827	**1.292**	65.8
	3.0	19×19	out of mem.	**1.089**	out of mem.	**1.334**	–
	4.0	25×25	out of mem.	**1.139**	out of mem.	**1.406**	–
2624×3936	1.0	7×7	99.462	**5.562**	28.008	**6.284**	68.7
	2.0	13×13	out of mem.	5.569	out of mem.	6.400	–
	3.0	19×19	out of mem.	5.588	out of mem.	6.520	–
	4.0	25×25	out of mem.	5.592	out of mem.	6.626	–

6 Conclusion

This study has proposed the BF* and demonstrated matrix-free and $O(1)$ algorithms as powerful tools for optimization-based image processing with BF. The proposed method can be extended to cross filtering and $O(1)$ filtering as an essential optimization tool. It can be applied to image processing based on optimization with BF without sacrificing the advantages of BF-based techniques.

Using the proposed approach, various conventional tasks that can traditionally be handled by BF will be extended as an optimization problem with a certain normalization method and appropriate constraints. This approach will outperform the conventional approach because of the added normalization and constraints. Many other cross-filtering methods, including the weighted least squares-based filter [16] and guided image filter [19], have been proposed. BF outperforms such existing methods in terms of computational complexity if high-resolution images are filtered or a large filter window is required. Thus, we consider that the BF will continue to be used in the future and that the proposed method will serve as an effective optimization tool.

Biographies

Keiichiro Shirai received the B.E., M.E., and D.Eng. degrees from Keio University, Yokohama, Japan, in 2001, 2003, and 2006. He has been with Shinshu University, Japan, since 2006 and is currently an Associate Professor of Electrical and Computer Engineering. His research interests include signal processing, image processing, and computer vision.

Kenjiro Sugimoto currently serves as an imaging engineer at Japan R&D Center, Xiaomi, since 2021. His research mainly focuses on signal/image processing, computer vision and machine learning, specializing in particular on accelerating imaging algorithms. He holds engineering degrees from Kurume National College of Technology (B.E. in 2007) and Waseda University (M.E. in 2009 and Ph.D. in 2015). Before working at the current position, he had been a research fellow of the Japan Society for the Promotion of Science (JSPS) during 2010–2012, a visiting research scientist with Durham University in 2015, and a research associate and an assistant professor with Graduate School of Information, Production and Systems, Waseda Univeristy (Japan) during 2014–2021. He is a member of IEEE and IEICE.

Sei-ichiro Kamata received the M.S. degree in computer science from Kyushu University, Fukuoka, Japan, in 1985, and the Doctor of Computer Science, Kyushu Institute of Technology, Kitakyushu, Japan, in 1995. From 1985 to 1988, he was with NEC, Ltd., Kawasaki, Japan. In 1988, he joined the faculty at Kyushu Institute of Technology. From 1996 to 2001, he has been an Associate Professor in the Department of Intelligent System, Graduate School of Information Science and Electrical Engineering, Kyushu University. Since 2003, he has been a professor in the Graduate School of Information, Production and Systems, Waseda University. In 1990 and 1994, he was a Visiting Researcher at the University of Maine, Orono. His research interests include image processing, pattern recognition, image compression, and space-filling curve application. He is a member of IEEE, IEICE and ITE in Japan.

References

[1] V. Aurich and J. Weule, "Non-linear Gaussian Filters Performing Edge Preserving Diffusion," in *Mustererkennung DAGM-Symposium*, 1995, 538–45.

[2] T. Baba, R. Matsuoka, S. Ono, K. Shirai, and M. Okuda, "Flash/No-flash Image Integration Using Convex Optimization," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, 1194–8.

[3] J. T. Barron, A. Adams, Y. Shih, and C. Hernández, "Fast Bilateral-space Stereo for Synthetic Defocus," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, 4466–74.

[4] J. T. Barron and B. Poole, "The Fast Bilateral Solver," in *Proc. Euro. Conf. Comput. Vis. (ECCV)*, 2016, 617–32.

[5] D. Charalampidis, "Recursive Implementation of the Gaussian Filter Using Truncated Cosine Functions," *IEEE Transactions on Signal Processing*, 64(14), 2016, 3554–65.

[6] K. N. Chaudhury, "Acceleration of the Shiftable $O(1)$ Algorithm for Bilateral Filtering and Nonlocal Means," *IEEE Transactions on Image Processing*, 22(4), 2013, 1291–300.

[7] K. N. Chaudhury, "Constant-time Filtering Using Shiftable Kernels," *IEEE Signal Processing Letters*, 18(11), 2011, 651–4.

[8] K. N. Chaudhury, D. Sage, and M. Unser, "Fast $O(1)$ Bilateral Filtering Using Trigonometric Range Kernels," *IEEE Transactions on Image Processing*, 20(12), 2011, 3376–82.

[9] L. Condat, "A Primal-dual Splitting Method for Convex Optimization Involving Lipschitzian, Proximable and Linear Composite Terms," *Journal of Optimization Theory and Applications*, 158(2), 2013, 460–79.

[10] G. Deng, "Fast Compressive Bilateral Filter," *Electronics Letters*, 53(3), 2017, 150–2.

[11] G. Deng, J. H. Manton, and S. Wang, "Fast Kernel Smoothing by a Low-rank Approximation of the Kernel Toeplitz Matrix," *Journal of Mathematical Imaging and Vision*, 60(8), 2018, 1181–95.

[12] R. Deriche, "Fast Algorithms for Low-level Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1), 1990, 78–87.

[13] F. Durand and J. Dorsey, "Fast Bilateral Filtering for the Display of High-dynamic-Range Images," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3), 2002, 257–66.

[14] E. Eisemann and F. Durand, "Flash Photography Enhancement via Intrinsic Relighting," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 1(212), 2004, 673–8.

[15] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of Inverse Problems*, Springer, 1996.

[16] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving Decompositions for Multi-scale Tone and Detail Manipulation," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3), 2008, 67:1–67:10.

[17] R. Franzen, "Kodak Lossless True Color Image Suite," http://r0k.us/graphics/kodak/, December 2021.

[18] L. Hang and K. Urahama, "Color Restoration Based on Scene Observation Model," *IEICE Lett. Fundamentals Electronics (in Japanese)*, J98-A(12), 2015, 695–9.

[19] K. He, J. Sun, and X. Tang, "Guided Image Filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6), 2013, 1397–409.

[20] F. Heide, S. Diamond, M. Nieß, J. Ragan-Kelley, W. Heidrich, and G. Wetzstein, "ProxImaL: Efficient Image Optimization Using Proximal Algorithms," *ACM Trans. Graph.*, 35(4), 2016, 84:1–84:15.

[21] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, 49, 1952, 409–36.

[22] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint Bilateral Upsampling," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26(3), 2007, 1–5.

[23] T. Matsuo, N. Fukushima, and Y. Ishibashi, "Weighted Joint Bilateral Filter with Slope Depth Compensation Filter for Depth Map Refinement," in *Proc. Int. Conf. Comput. Vis. Theory Appl. (VISAPP)*, Vol. 2, 2013, 300–9.

[24] R. Morita, K. Shirai, and Y. Tanaka, "Retargeting Pyramid Using Direct Decimation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, 2832–6.

[25] S. Paris and F. Durand, "A Fast Approximation of the Bilateral Filter Using a Signal Processing Approach," *International Journal of Computer Vision*, 81(1), 2009, 24–52.

[26] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama, "Digital Photography with Flash and No-flash Image Pairs," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 1(212), 2004, 664–72.

[27] F. Pitié, A. C. Kokaram, and R. Dahyot, "Automated Colour Grading Using Colour Distribution Transfer," *Comput. Vis. Image Understd.*, 107(1-2), 2007, 123–37.

[28] J. Rabin, J. Delon, and Y. Gousseau, "Regularization of Transportation Maps for Color and Contrast Transfer," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2010, 1933–6.

[29] R. Ramanath and W. E. Snyder, "Adaptive demosaicking," *J. Electronic Imag.*, 12(4), 2003, 633–42.

[30] Y. Romano, M. Elad, and P. Milanfar, "The Little Engine That Could: Regularization by Denoising (RED)," *SIAM Journal on Imaging Processing*, 10(4), 2017, 1804–44.

[31] S. M. Smith and J. M. Brady, "SUSAN – A New Approach to Low Level Image Processing," *International Journal of Computer Vision (IJCV)*, 23(1), 1997, 45–78.

[32] K. Sugimoto, T. P. Breckon, and S. Kamata, "Constant-time Bilateral Filter Using Spectral Decomposition," in *IEEE Internat. Conf. Image Process. (ICIP)*, September 2016, 3319–23.

[33] K. Sugimoto, N. Fukushima, and S. Kamata, "200 FPS Constant-Time Bilateral Filter Using SVD and Tiling Strategy," in *IEEE Internat. Conf. Image Process. (ICIP)*, September 2019, 190–4.

[34] K. Sugimoto and S. Kamata, "Compressive Bilateral Filtering," *IEEE Transactions on Image Processing (TIP)*, 24(11), 2015, 3357–69.

[35] K. Sugimoto and S. Kamata, "Efficient Constant-time Gaussian Filtering with Sliding DCT/DST-5 and Dual-domain Error Minimization," *ITE Trans. Media Technol. Appli.*, 3(1), 2015, 12–21.

[36] K. Sugimoto and S. Kamata, "Fast Gaussian Filter with Second-order Shift Property of DCT-5," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2013, 514–8.

[37] K. Sugimoto, K. Shirai, and S. Kamata, "$O(1)$ Transposed Bilateral Filtering for Optimization," in *Proc. APSIPA Conf. Signal Process.* 2014, 1–4.

[38] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE Int. Conf. Comput. VIs. (ICCV)*, 1998, 839–46.

[39] L. J. van Vliet, I. T. Young, and P. W. Verbeek, "Recursive Gaussian Derivative Filters," in *Proc. IAPR Int. Conf. Pattern Recognit. (ICPR)*, Vol. 1, 1998, 509–14.

[40] H. Winnemöller, S. C. Olsen, and B. Gooch, "Real-time Video Abstraction," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3), 2006, 1221–6.

[41] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi, "Bilateral Filtering-based Optical Flow Estimation," *Proc. Euro. Conf. Comput. Vis. (ECCV)*, Springer LNCS 3951, 2006, 211–24.

[42] Q. Yang, K.-H. Tan, and N. Ahuja, "Real-time $O(1)$ Bilateral Filtering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, 557–64.

[43] L. P. Yaroslavsky, *Digital Picture Processing. An Introduction*, Springer-Verlag, Berlin, 1985.

[44] S. Yoshizawa, A. Belyaev, and H. Yokota, "Fast Gauss Bilateral Filtering," *Computer Graphics Forum*, 29(1), 2010, 60–74.

[45] L. Yuan, J. Sun, L. Quan, and H.-Y. Shum, "Progressive Inter-scale and Intra-scale Non-blind Image Deconvolution," *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3), 2008, 74:1–74:10.