Abstract—This paper presents an essential algorithm for optimization-based image processing using the bilateral filter (BLF), called constant-time transposed BLF ($O(1)$ TBLF). Some iterative solvers for optimization problems require a pair of filters defined as multiplying a filter matrix or its transpose to vectorized images. Since the BLF can be described as a matrix form, its paired filter also exists, called a TBLF in this paper. BLF-based optimization achieves high smoothing performance; whereas, it requires much high computational complexity due to iterating both BLF and TBLF many times. Hence, this paper designs an $O(1)$ TBLF algorithm to accelerate the iterative process. Experiments show that our $O(1)$ TBLF runs in low complexity regardless of its filter window size and works effectively for flash/no-flash image integration via BLF-based optimization.

I. INTRODUCTION

Edge-preserving smoothing has played a fundamental role on image processing, computer vision, and computer graphics recent years. In particular, the bilateral filter (BLF) [1]–[3], which determines filter coefficients from two laterals: pixel position and pixel intensity, has flourished in various applications because of its clear concept and smoothing efficiency. The BLF has been widely extended in the literature to enhance the smoothing efficiency or to reduce the computational complexity. The cross BLF [4] (XBLF), which is identical to the joint BLF [5], is a natural extension of the BLF. In filtering a noisy target image, the BLF diminishes the smoothing efficiency due to determining its filter coefficients from the noisy target; by contrast, the XBLF overcomes this problem by determining them from a guide image captured under a different photographic condition instead. Another algorithmic extension is the fast BLF [6], [7] including the constant-time BLF ($O(1)$ BLF) [8]–[11]. Both original BLF and XBLF often require unacceptable computational complexity in filtering high-resolution or high-dimensional images because the cost-per-pixel depends on the filter window size. On the other hand, the $O(1)$ BLF runs in $O(1)$ time per pixel, easily generalized to the XBLF. We discuss this BLF family from a different viewpoint. For simplicity, both BLF and XBLF are collectively-referred to as BLF in this paper.

Our motivation comes from optimization-based image processing. This handles image restoration tasks such as deblurring through formalizing the tasks as optimization problems and solving the formalized problems by iterative solvers. The procedures of iterative solvers are generally described as matrix equations that consist of several linear transforms to vectorized images. These transforms can also be interpreted as a series of image filters. Thus, we basically have two aspects to this operation: as a matrix form or as a filter form. Since the BLF is a time-varying filter, its filter matrix has different elements in different rows. Using this filter matrix expression, we can easily formalize optimization problems based on the BLF. Importantly, the BLF-based optimization has been showing outstanding performance in several tasks recent years [12], [13].

A severe problem for the BLF-based optimization is its tremendous computational complexity. This stems from the following situation. Consider a filter matrix $B$ that describes a filter to a vectorized image. In a recent convex optimization as mentioned in [14], some iterative solvers require to apply $B$ and $B^\top$ several times for each iteration. Since the BLF can be represented as $B$, its paired filter $B^\top$ also exists. We call it the transposed BLF (TBLF). Thus, iteratively solving BLF-based optimization problems requires a pair of the BLF and the TBLF. However, straightforwardly generating and multiplying $B^\top$ highly cost for iterative solvers. Moreover, unlike the $O(1)$ BLF, the computational complexity is not $O(1)$ per pixel. We should perform the TBLF without explicitly-generating $B^\top$ in low computational complexity regardless of the filter window size.

This paper presents essential algorithms for accelerating BLF-based optimization. We first derive the filter form of the TBLF to perform it without explicitly-generating $B^\top$. We then propose an $O(1)$ TBLF to solve BLF-based optimization problems in acceptable computational complexity. As an application example, we discuss a BLF-based optimization problem for flash/no-flash image integration [4], [5]. It is worth noting that our approach is applicable to any optimization tasks with a similar mathematical model to this example.

II. OPTIMIZATION BASED ON BILATERAL FILTERING

This section clarifies our motivation through an example of image processing applications under our scenario.

A. Bilateral Filter

Consider smoothing a grayscale image by the BLF in a cross/joint fashion. Let $x, y, z \in \mathbb{R}^N$ be vectorized images that are a target image, a reference image, and a smoothed image where $N$ indicates the number of pixels. We use a notation [5]

\[ (f \ast g)(x) = \sum_{u} f(u)g(x - u) \text{ and the correlation filtering } (f \odot g)(x) = \sum_{u} f(u)g(x + u), \]

The matrix forms $Bx$ and $B^\top x$ correspond to the convolution filtering $(f \ast g)(x)$ and the correlation filtering $(f \odot g)(x)$, respectively.
to denote the element at \(i\)-th row for a vector and use \([i, j]\) to denote the element at \(i\)-th row of \(j\)-th column for a matrix. Let \(\Omega \subset \mathbb{Z}^2\) be an image domain, \(i\) the index of a pixel, and \(p_i \in \Omega\) a pixel position, and \(j \in \mathbb{N}\) a neighboring pixel centered at the pixel \(i\). The BLF has the form

\[
B_y[i, j] = \frac{g_y(p_j - p_i) g_r(y[j] - y[i])}{\sum_{k \in \mathbb{N}} g_y(p_k - p_i) g_r(y[k] - y[i])},
\]

(1)

where \(g_y(\cdot)\) is a spatial kernel, \(g_r(\cdot)\) is a range kernel. Note that the subscript \(y\) of \(B_y\) indicates that the filter weight is determined from the image \(y\). This filtering process is designed in [5]:

where \(\llVert \cdot \rrVert_2\) denotes \(\ell_2\)-norm. The kernel size \(|\mathcal{N}_i|\), i.e., the window size of the filter, depends on \(\sigma_s\).

B. Sample Model for Optimization

As a sample application under our scenario, we focus on flash/no-flash image integration [4], [5]. This task aims to generate a noiseless no-flash images by composing the base structure component of a no-flash image and the texture detail component of a flash image. Generally, the BLF has been mainly used for smoothing and decomposition of image components. In fact, the XBLF was first proposed for this task. We describe the formulations for this application to clarify our target problem. Note that our proposed algorithm is applicable to any optimization tasks with similar formulations.

In flash/no-flash image integration, \(x\) is a no-flash image that has preferred color information but contains much noise, and \(y\) is a flash image that has unpreferred color information but contains less noise where each color component is processed independently. In order to enhance smoothing performance, the following filtering process is designed in [5]:

\[
z = B_y x + (I - B_y) y,
\]

(3)

where \(I \in \mathbb{R}^{N \times N}\) is an identity matrix. In (3), the first term represents structure components such as color or contour and the second term represents texture components obtained by subtracting the source image \(y\) from its smoothed image \(B_y y\).

C. Motivation

The aforementioned approach can be improved by extending it to an convex optimization problem. Specifically

\[
\arg \min_x \|B_y (z - x)\|_2^2 + \lambda \| (I - B_y) (z - y)\|_2^2,
\]

(4)

where \(\lambda\) is an influence rate for balancing both terms and the second term plays a role analogous to total variation (TV) using the BLF. The general model (4) as an optimization problem has some advantages over the original model (3): it guarantees existence and uniqueness of the optimal solution and provides options such as adjustment of influence rate and flexibility of norm selection. Its solution can be obtained as follows. Derivating (4) w.r.t. \(z\) and assuming it to be zero, we obtain

\[
\begin{aligned}
\left\{ B_y^T B_y + \lambda (I - B_y)^T (I - B_y) \right\} z \\
= B_y^T B_y x + \lambda (I - B_y)^T (I - B_y) y.
\end{aligned}
\]

(5)

This equation can be interpreted as a standard form \(A x = b\). Since \(A\) is a symmetric matrix, we can efficiently solve the problem by an iterative solver such as the conjugate gradient method [15], [16]. For each iteration, the following operation of matrix \(A\) and a conjugate vector \(u\) are required:

\[
A u := \left\{ B_y^T B_y + \lambda (I - B_y)^T (I - B_y) \right\} u.
\]

(6)

Obviously, the computational complexity of this iterative operation is dominated by the two \(B_y\) and the two \(B_y^T\). Although our optimization approach will provide high smoothing performance, it requires tremendous computational complexity due to multiplying \(B_y\) and \(B_y^T\) many times. Hence, we accelerate the operation of \(B_y^T\) to solve the optimization problem in acceptable complexity. Importantly, BLF-based optimization approaches with similar formalizations [12], [13] have also faced to the same difficulty.

III. \(O(1)\) TRANSPOSED BILATERAL FILTERING

This section describes the filter form and the matrix form of the BLF and the TBLF. The important points are how to perform the TBLF without generating the \(B^T\) and how to accelerate it to \(O(1)\) complexity per pixel.

A. Filter Form of Transposed Cross Bilateral Filter

We clarify differences between the BLF (\(z = B_y x\)) and the TBLF (\(z = B_y^T x\)). Consider decomposing \(B_y\) into a numerator part and a denominator part by newly introducing filter matrix \(G_y \in \mathbb{R}^{N \times N}\),

\[
G_y[i, j] = g_y(p_j - p_i) g_r(y[j] - y[i]), \quad w_y = G_y 1,
\]

where \(1 \in \mathbb{R}^N\) is a column vector of all ones. The BLF is expanded to

\[
B_y x = \{G_y \odot (w_y 1^T)\} x = (G_y x) \odot w_y,
\]

(7)

where \(\odot\) denotes an element-wise division operator for a matrix or vector, and we used \(\{A \odot (bc^T)\} x = \{A(x \odot c)\} \odot b\) and \(b, c \in \mathbb{R}^N, A \in \mathbb{R}^{N \times N}\). Similarly, the TBLF is

\[
B_y^T x = \{G_y \odot (w_y 1^T)\}^T x = \{G_y^T \odot (w_y 1^T)^T\} x
\]

\[
= \{G_y \odot (1 w_y^T)\} x = G_y (x \odot w_y),
\]

(8)

where \(G_y = G_y^T\). Note that \(w_y 1^T\) in (7) and \(1 w_y^T\) in (8) represent matrices aligning \(w_y\) in each column and \(w_y^T\) in each row, respectively. An important difference between the matrix forms of the BLF and the TBLF is the target of dividing \(G_y x\) by \(w_y\). The BLF divides the output image of \(G_y x\); by contrast, the TBLF divides the input image of \(G_y x\) in advance.
Filter forms, which we can operate without explicitly-generating filter matrices, are more efficient than matrix forms in terms of computational and space complexity. The filter form of the TBLF is derived from (8) by using intermediate image \(s \in \mathbb{R}^N\) as
\[
s[i] = \sum_{j \in \mathcal{N}_i} g_r(p_j - p_i) g_s(y[j] - y[i]) =: x[i],
\]
\[
z[i] = \sum_{j \in \mathcal{N}_i} g_s(p_j - p_i) g_r(y[j] - y[i]) s[j].
\]
Although this operation is identical to \(z = B^T_y x\), we can operate it without explicitly-generating entire \(B^T_y\). Thereby, a procedure of the TBLF is summarized as follows:

**Algorithm 1 Transposed Bilateral Filter**

1. **function** TBLF(x, y) \(\triangleright x\): Target, \(y\): Reference
2. \(s \leftarrow x \odot (G_y 1)\) \(\triangleright (9)\)
3. \(z \leftarrow G_y s\) \(\triangleright (10)\)
4. return \(z\)

### B. Constant-time Algorithm

The computational complexity of the aforementioned naive algorithm depends on the filter window size, i.e., the spatial scale \(\sigma_s\), in order to achieve faster optimization, we should improve the naive TBLF to the constant-time one. An important observation is that both (9) and (10) can execute in \(O(1)\) complexity per pixel. As [10], [11] reported, each of the numerator and the denominator in (1) can be executed in \(O(1)\). Obviously, (9) and (10) have the same forms as the numerator and denominator and the division in (9) obviously runs in \(O(1)\). Due to space limitation, we have explained only a summary of the procedure of our \(O(1)\) TBLF (see also [11] for the detail).

We modify the algorithm of [11] and then apply the same approach to the TBLF. In [11], the BLF is decomposed into a bunch of the constant-time Gaussian filters such as the recursive Gaussian filters [17], [18]. This paper applies two techniques to achieve more efficient performance. First, replacing the recursive Gaussian filtering to the constant-time Gaussian filter proposed in [19] provides a higher approximate accuracy in lower computational complexity. Second, aggregating the cosine terms of approximate Gaussian kernels used in [11] can reduce the number of the Gaussian filters by half without a loss of accuracy. Specifically, in Eq. (16) of [11], the \(m\)-th and \((N - m)\)-th cosine terms (\(m = 0, 1, \ldots, \frac{N}{2}\)) share the same cosine basis. These two modifications provide a 4\(\times\) faster computation than [11].

### IV. EXPERIMENTS AND DISCUSSION

First of all, we evaluate the basic performance of the naive BLF, the naive TBLF, and our \(O(1)\) TBLF. The test image is “lenna” (RGB with 512\(\times\)512 pixels), treated as 64-bits floating point RGB data with a normalized dynamic range, i.e., [0, 1]. We set window size \(M = 2 \lceil 3\sigma_s \rceil + 1\) for the naive BLF and the naive TBLF, and \(\epsilon = 0.5\) and \(K = 1\) for our \(O(1)\) TBLF. All the implementations are written in Matlab with MEX. Our test environment mounts Intel 2.67GHz CPU with 8GB main memory. Figure 1 plots the spatial scale \(\sigma_s\) or the range scale \(\sigma_r\), against the computational time \([s]\). Against \(\sigma_s\), the naive TBLF consumes \(O(M)\) complexity because \(M\) depends on \(\sigma_s\); by contrast, our \(O(1)\) TBLF evidently achieves \(O(1)\) complexity. Against \(\sigma_r\), even though our \(O(1)\) BLF somewhat depends on \(\sigma_r\), the computation is at least 3\(\times\) faster than the naive TBLF over a wide range of \(\sigma_r\). The naive TBLF is nearly 2\(\times\) slower than the naive BLF due to its inefficient pipeline, i.e., (7) requires only one-pass processing but (8) does three-pass. Figure 2 plots \(\sigma_s\) and \(\sigma_r\) versus the approximate accuracy \([\text{dB}]\), revealing that our \(O(1)\) TBLF can produce a sufficient approximate accuracy regardless of \(\sigma_r\) and \(\sigma_s\). Thus, our \(O(1)\) TBLF can replace the naive TBLF without a large loss of accuracy and can provide a faster filtering speed.

Figure 3 lists output images of the competitors and the 20\(\times\) amplified error between the naive TBLF and our \(O(1)\) TBLF for visual assessment. The BLF works as an edge-preserved smoother; on the other hand, the TBLF works as smoothing around edges, e.g., some Halo occurred around edges, but emphasizing the center core of edges or small regions. Our \(O(1)\) TBLF produces some error around edges as compared with the TBLF but the amount of errors is sufficiently acceptable since the error image is 20\(\times\) amplified.

Third, we observe the effectiveness of BLF-based optimization approaches to flash/no-flash image integration. The test images are RGB with 400\(\times\)476 pixels. Figure 4 shows the results of well-known approaches. A key advantage of our approach is to strictly provide one optimum solution and to adjust the strongness/weakness of the texture component
by changing $\lambda$. The computational time of our optimization algorithm is about 9 s [10 iterations $\times$ 0.9 s per iteration]. Our $O(1)$ TBLF achieved this acceptable time of iterative approaches.

V. CONCLUSIONS

This paper presented an $O(1)$ TBLF as an essential algorithm for BLF-based optimization problems. Our experiments validated the efficiency of our TBLF in terms of the computational complexity and the approximate accuracy over a wide range of scale parameters. This performance enables us to make it possible solve BLF-based optimization problems in acceptable computational complexity, as we validated in flash/no-flash image integration. Moreover, the simple model (4) can be extended to more highly-designed models, e.g., by adding terms of vector-valued total variation for color images, and can be solved by more advanced solvers such as the primal-dual splitting [14]. Hence, our approach has a large potential to contribute many image processing applications.

REFERENCES