Fast Scale-Adaptive Bilateral Texture Smoothing

Sanjay Ghosh, *Student Member, IEEE*, Ruturaj G. Gavaskar, Debasisha Panda, and Kunal N. Chaudhury, *Senior Member, IEEE*

Abstract

In the classical bilateral filter, a range kernel is used together with a spatial kernel for smoothing out fine details, while simultaneously preserving edges. More recently, it has been demonstrated that even coarse textures can be smoothed using joint bilateral filtering. In this work, we demonstrate that superior texture filtering results can be obtained by adapting the spatial kernel at each pixel. To the best of our knowledge, spatial adaptation (of the bilateral filter) has not been explored for texture smoothing. The rationale behind adapting the spatial kernel is that one cannot smooth beyond a certain level using a fixed spatial kernel, no matter how we manipulate the range kernel. In fact, we should simply aggregate more pixels using a sufficiently wide spatial kernel to locally enhance the smoothing. Based on this reasoning, we propose to use the classical bilateral filter for texture smoothing, where we adapt the width of the spatial kernel at each pixel. We describe a simple and efficient gradient-based rule for the latter task. The attractive aspect is that we are able to develop a fast algorithm that can accelerate the computations by an order without visibly compromising the filtering quality. We demonstrate that our method outperforms classical bilateral filtering, joint bilateral filtering and other filtering methods, and is competitive with optimization methods. We also present some applications of texture smoothing using the proposed method.

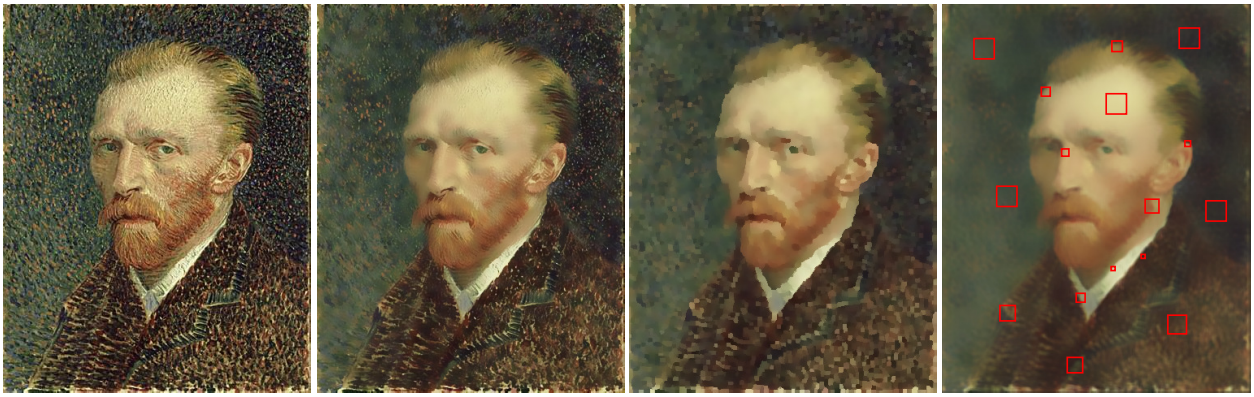
Index Terms

texture, smoothing, bilateral filter, adaptation, fast algorithm.

I. Introduction

Structure-preserving smoothing plays an important role in several image processing applications such as denoising, editing, abstraction, seam carving, compression artifact reduction, tone mapping, detail manipulation, and inverse half-toning [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. The goal of structure-preserving smoothing is to diffuse fine details such as noise and small textures, while preserving salient structures such as edges, corners, etc [1].

S. Ghosh, R. G. Gavaskar, and K. N. Chaudhury are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore, India. D. Panda is with Samsung Electromechanics, Bangalore, India. The work of D. Panda was done as a Masters student at Indian Institute of Science, Bangalore, India. K. N. Chaudhury was partially supported by EMR Grant SB/S3/EECE/281/2016 from DST-SERB, Government of India. Correspondence: kunal@iisc.ac.in.



(a) Input. (b) Bilateral Filtering [2]. (c) Joint Bilateral Filtering [10]. (d) Scale-Adaptive Bilateral Filtering.

Fig. 1. Texture suppression using the classical bilateral filter and its variants. Notice that bilateral filter (b) is unable to remove coarse texture. Joint bilateral filtering (c) performs somewhat better, however, some texture is still visible in the background. The proposed filtering (d) performs better in terms of edge-preservation and texture smoothing. This is because we adapt the width of the spatial kernel at each pixel (some of which are marked with red boxes). In particular, a wider spatial kernel is used in the background in (d) which helps to smooth out coarse texture. In contrast, a fixed spatial kernel is used in (c).

A popular edge-preserving smoother is the bilateral filter [2], [12], [13]. It can smooth fine details, while preserving sharp edges. A range kernel is used along with a spatial kernel for this purpose. As in convolutional filtering, the role of the latter is simply to aggregate neighboring pixels. On the other hand, the range kernel essentially acts as an inhibitor during the aggregation. It reduces blurring of edges by preventing the mixing of pixels from both sides of an edge during the aggregation. In particular, the mixing is controlled using the intensity difference between the pixel of interest and its neighbor; this mechanism is built into the bilateral filter. While the bilateral filter is reasonably good at filtering fine details, it cannot directly be used to smooth coarse textures and preserve edges at the same time (see Figure 1).

One possible way to smooth out coarse textures is to locally adapt the smoothing [9], [10], [14], [15], [16], [17], [18], [19], [20]. The authors in [9], [10] used bilateral filtering for texture smoothing. On the other hand, texture smoothing was performed using total-variation (TV) regularization and its variants in [4], [5], [6], [21]. We now present a detailed review of the literature on texture smoothing.

A. *Filtering based methods*

In several papers, the texture smoothing problem is modeled as one of decomposing the image f as $f = u + v$, where u is the piecewise-smooth (cartoon) component with structures, and v consists of fine details and textures. The idea of using nonlinear filtering was proposed in [14], where a local variant of TV (LTV) was used to decide whether a pixel belongs to a cartoon or textured region. Inspired by [22], the idea of locally adapting the smoothing was investigated in [15]. In particular, the authors used a metric from [23] to distinguish between texture and cartoon, and optimize the filtering at each pixel. A patch-based framework based on a covariance descriptor was

proposed in [8], whereas the rolling guidance filter [16], uses scale-aware Gaussian filtering.

In a recent paper [10], the authors proposed to use joint bilateral filtering, where the input to the range kernel (guide) is used to control the smoothing. The guide image is computed from the local gradients. Joint bilateral filtering was also used in [9], where a smoothed version of the input image is used as a guide. Moreover, it was observed here that the guided filter [28] can be used in place of the joint bilateral filter. The key idea in [9] is to combine the texture-smoothing capability of linear filtering and the edge-preserving capability of bilateral filtering. However, a fixed spatial kernel is used in [9], [10]. Ideally, we should use a small spatial kernel in the vicinity of an edge and a large kernel in a homogeneous region. By using a fixed kernel, one cannot satisfactorily smooth out textures in homogeneous regions while ensuring edge-preservation (see comparison in Figure 1). Moreover, [9], [10] often produces blocking artifacts and residual textures.

B. Optimization methods

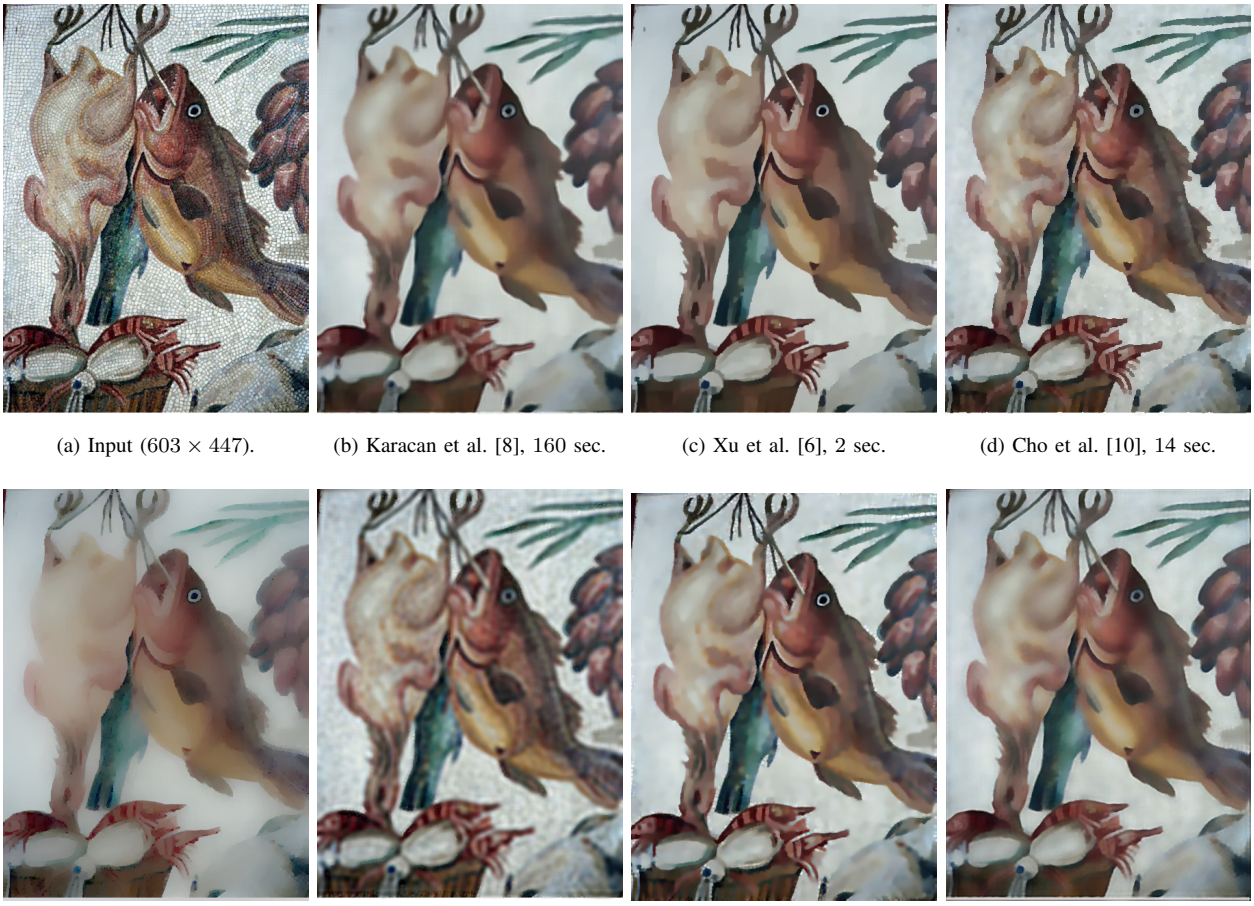
Apart from filtering approaches, several optimization models have been proposed for texture smoothing. In perhaps the earliest model [4], weighted least-squares was used within a multiscale framework for adaptive smoothing. In [29], the local extrema were used to extract information about oscillations, which were then used to distinguish textures from edges. The authors in [26] accelerated this method using post-processing. However, the post-processing often introduces artifacts in the vicinity of edges. In [27] it was demonstrated that one can obtain better results using the extended extrema. It was shown in [30] that TV regularization performs best if we assume no prior information about the texture statistics. TV regularization is defined using the L_1 norm of the image gradients. The use of L_0 norm (number of pixels with non-zero gradient) instead was advocated in [5]. Later, an edge-preserving filter based on L_0-gradient was proposed in [11].

In [31], [32], [33], the texture-structure separation problem was modeled using low-rank regularization. Since textured patches are typically of low rank, the nuclear norm was used in [31] to regularize the texture component. Based on the observation that textures are globally dissimilar but are locally well-patterned, block nuclear norm was used in [32] instead of the standard nuclear norm. Later, it was shown in [33] that by using the "log-determinant" regularizer one can further improve the decomposition.

An optimization model using the so-called relative TV (RTV) was proposed in [6]. This was based on the observation that classical TV cannot distinguish well between structures and textures. To further improve the discrimination between texture and cartoon components, a regularizer was proposed in [21] that enforces incoherence between gradients of the two components. Following the model in [14], compressed sensing techniques were used in [34] for reconstructing the cartoon and texture components. Recently, the authors in [35] proposed to adapt TV minimization to the moving least squares method with non-local weights.

C. Contributions

The idea of adapting the bilateral filter kernel has been used in various applications such as enhancement, denoising [36], [37], and artifact reduction [38], [39]. In particular, the authors in [40] adapted the range kernel



(a) Input (603×447). (b) Karacan et al. [8], 160 sec. (c) Xu et al. [6], 2 sec. (d) Cho et al. [10], 14 sec.

(e) WLS [4], 2.5 sec. (f) Buades et al. [14]. (g) Szolgay and Szirányi [15], 90 sec. (h) Proposed, 2.3 sec.

Fig. 2. Comparison with existing methods. The timings for the respective Matlab implementations on a 3.40 GHz quad-core machine with 32 GB memory are also mentioned. Notice that the texture smoothing and edge-preservation is visibly better for our result compared to [10], [14], [15]. Moreover, unlike the optimization-based methods [4], [6], the shading is better preserved in our result. The closest match is [8], but we are about 30 times faster.

parameter to construct a guidance image for joint bilateral filtering. However, to the best of our knowledge, the idea of adapting the spatial kernel of the bilateral filter for *texture smoothing* has not been explored in the literature. We note that the idea of spatial adaptation was used in [41], but for a different smoothing filter and not the bilateral filter. The novelty of our method is the way the spatial window of the bilateral filter is adapted at each pixel (based on the structure content), its application to texture smoothing, and a fast algorithm for implementing the same. More specifically, our contributions are as follows:

- We show that by locally adjusting the width of the spatial kernel, the classical bilateral filter can be used for edge-preserving texture smoothing. In particular, we demonstrate this has greater potential for texture smoothing than joint bilateral filtering [10] (see Figure 1).
- We study the problem of setting the kernel width (scale) using structural information. We propose a simple

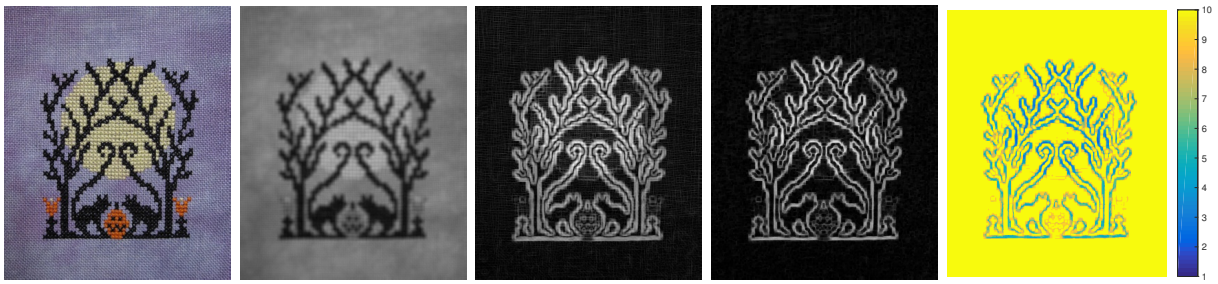
parametric rule for this purpose using three basic operations, namely, convolution, gradient computation, and morphological erosion. In particular, the pixelwise scales are efficiently computed from the input image. The texture smoothing results obtained using this rule are shown to be competitive with state-of-the-art methods.

- We also develop a fast constant-time algorithm for scale-adaptive bilateral filtering. More precisely, the complexity of the algorithm does not depend on the assigned scales. As a result, we can use wide spatial kernels in homogeneous regions to obtain better texture smoothing without compromising the timing. Our algorithm is about an order faster than brute-force filtering. It is also much faster than some of the existing methods. For example, we are about 30 times faster than [8].

In summary, we propose a bilateral filtering based method for texture smoothing that is competitive with state-of-the-art methods in terms of performance and speed (see Figure 2). To demonstrate the effectiveness of our proposal, we present some applications of texture smoothing.

D. Organization

The rest of the paper is organized as follows. The proposed scale-adaptive bilateral filter is described in Section II, along with the rule for setting the width of the spatial kernel at each pixel. A fast algorithm for scale-adaptive bilateral filtering is presented in Section III. The performance of our method is analyzed and compared with existing methods in Section IV. In Section V, we present applications of texture smoothing using our method. We conclude the paper with a summary of the results in Section VI.



(a) Input. (b) Average. (c) Gradient. (d) Erosion. (e) Scale map.

Fig. 3. Computation of the scale map (which drives the adaptive bilateral filtering) from the input image using averaging, gradient operation, and erosion. The windows Θ and Γ are 7×7 and 3×3, and $\omega_0 = 0, \omega_\infty = 10$, and $\lambda = 2$ in (6).

II. Proposed Method

A. Adaptive bilateral filter

As mentioned earlier, the bilateral filter uses a range kernel along with a spatial kernel. In the original proposal [2], both the kernels are Gaussian. In this work, we propose to use a box function for the spatial kernel. This is because we can develop a fast algorithm in this case. We have empirically observed that the results obtained using Gaussian and box kernels have very little visual difference between them. In fact, what is important for texture smoothing is the local scale of the spatial kernel.

In our proposal, we wish to change the scale (width) of the box at each pixel. Stated simply, the aggregation at pixel $i = (i_1, i_2)$ is performed over a window Ω_i centered at i:

$$\Omega_i = \big\{(i_1 + k_1, i_2 + k_2) : -\omega_i \leq k_1, k_2 \leq \omega_i\big\}. \tag{1}$$

where the positive integer ω_i is the scale at pixel i. We will collectively refer to (ω_i) as the *scale map*.

Suppose we have a scale map. The proposed filtering of an input image $f(i)$ is then given by

$$f_{\mathrm{BF}}(i) = \frac{\sum_{j\in\Omega_i} \varphi\big(f(j) - f(i)\big) f(j)}{\sum_{j\in\Omega_i} \varphi\big(f(j) - f(i)\big)}, \tag{2}$$

where φ is a Gaussian range kernel:

$$\varphi(t) = \exp\Big(-\frac{t^2}{2\sigma^2}\Big). \tag{3}$$

This reduces to the standard bilateral filter if the ω_i's are equal. The parameter σ controls the width and, in effect, the inhibitory action of the range kernel [2], [3]. A small σ results in less smoothing and better edge-preservation, while a large σ results in more smoothing but wanted blurring of edges; the optimal setting is somewhere in between. For our proposal, we use a fixed σ at each pixel. For texture filtering, we noticed that the best results are obtained when σ is in the range $[30, 50]$. On the other hand, the amount of pixel aggregation is controlled by ω_i and is proportional to $|\Omega_i| = (2\omega_i + 1)^2$. Henceforth, we will refer to (2) as the *scale-adaptive bilateral filter*.

The intuition behind adapting the scale is as follows. Ideally, we wish to use a small ω_i if pixel i is on or near an edge, and a large ω_i if the pixel is in a homogeneous or textured region. The success of our proposal depends critically on the scale map. An important observation, based on exhaustive experiments on scale adaption, is that if the scale is large (so that more pixels are aggregated), then the range kernel can only prevent blurring up to a limited extend, unless σ is taken to be small. However, there is essentially no filtering if σ is very small.

B. Scale adaptation

We now consider the problem of assigning the scale map. The first step is to construct a measure m that can capture local structural information. Ideally, m should be large close to an edge, and should be small in homogeneous and textured regions. Measures to distinguish between texture and structure have in fact been proposed in the literature. Examples include Local TV [14], Relative TV [6], and Modified Relative TV [10]. For our purpose, we propose a simple measure which can be computed efficiently.

First, we blur the input image $f(i)$ using an averaging filter. That is, we consider the blurred image $\tilde{f}(i)$ given by

$$\tilde{f}(i) = |\Theta|^{-1} \sum_{j\in\Theta} f(i - j), \tag{4}$$

where Θ is a square window centered at $(0, 0)$, and $|\Theta|$ denotes the number of pixels in Θ. Let $\nabla\tilde{f}(i)$ be the gradient of (4). The proposed structure measure is defined to be

$$m_i = \min_{j\in\Gamma} \|\nabla\tilde{f}(i - j)\|, \tag{5}$$

where Γ is a square window centered at $(0,0)$. In general, Γ is smaller than Θ. Finally, we rescale (m_i) to $[0,1]$.

In summary, we arrive at (5) from the input in three steps: averaging, gradient computation, and erosion. The averaging operation smooths out fine as well as coarse textures, provided we use a sufficiently large window Θ (i.e., larger than what is typically used in edge detectors). However, we also end up blurring the edges in the process. The gradient step results in thick edges as a result but, importantly, produces a small output in textured and homogeneous regions. The final step is the minimization in (5), where we replace the value of each pixel in the gradient image by the minimum of the values in its Γ-neighborhood. This thins out the edges and removes unwanted "noise" in textured regions. The computation pipeline is explained with an example in Figure 3.

We briefly explain how the final step works to thin out broad edges. Since we apply greater smoothing (to smooth out the texture) prior to the gradient step, the edges obtained are usually wide (less localized). Consider a pixel i located close to the boundary of an edge, on the high-intensity side. The Γ-neighborhood of this pixel will contain few pixels of low intensity, hence the minimum pixel value in this neighborhood (the right side of (5)) is smaller than the original value at i. Since we replace the value of i by this minimum, the boundary of the edge gets pushed further into the high-intensity side, i.e., the boundary gets eroded. In fact, this is simply the well-known morphological operation of erosion, with a flat structuring element [42], [43]. We remark that the size of the Γ window should be smaller than the thickness of the edges, otherwise the edges will disappear completely.

As is well-known, we can compute (4) using a fixed number of operations per pixel regardless of the size of Θ. Moreover there exists an efficient algorithm for the local minimization (erosion) in (5), wherein the number of operations is independent of the size of Γ. We refer the reader to [44, Algorithm 1] where the local maxima are computed using such an algorithm, and note that this algorithm can be easily modified to compute (5). In other words, we can efficiently compute m for any size of Θ and Γ. We also found that the proposed scale map is robust to noise. In this case, the grayscale version of the color image is taken as $f(i)$; we will follow this convention for the images in this paper.

To determine the scale ω_i using m_i, we fix an interval for the former. That is, we assume that each ω_i is in the range $[\omega_0, \omega_\infty]$. We basically need to perform an "inverse" mapping from m_i to ω_i. By design, the structure measure m_i takes small values in smooth regions (or regions with small details and faint edges), whereas it assumes large values near sharp edges/discontinuities. To extract a strong edge, the kernel width ω_i should be relatively small in its vicinity. This ensures less blurring of the edge. On the other hand, ω_i should be large in regions with faint edges and fine textures (which we wish to smooth out). In summary, ω_i should be small if m_i is large and vice versa. Moreover, the dependence should be vary smoothly to ensure regularity in the filtered output. We experimented with various inverse mappings and selected the following:

$$\omega_i = (\omega_\infty - \omega_0)\exp(-\lambda m_i^2) + \omega_0, \tag{6}$$

where $\lambda > 0$ determines the transition rate from ω_0 to ω_∞. Finally, we round each ω_i to the nearest integer. We empirically found that satisfactory texture smoothing is obtained using (6). It follows from (6) that ω_i is small (resp. large) when m_i is large (resp. small). Moreover, $\omega_i \in [\omega_0, \omega_\infty]$ irrespective of the value of m_i. We will shortly

demonstrate that this simple rule produces good texture smoothing. The scale map obtained using (6) is shown in Figure 3(e).

(a) Input (338×450). (b) Brute-force, 9 sec. (c) Fast Algorithm, 1.1 sec. (d) R-channel difference.

Fig. 4. Comparison of the brute-force implementation of adaptive bilateral filtering and the proposed fast algorithm. The parameters used are $\omega_0 = 0, \omega_\infty = 15, \lambda = 4$, and $\sigma = 30$. The tolerance is $\varepsilon = 0.1$. Notice that (b) and (c) look visually similar. The PSNR between (b) and (c) is 79.2 dB. The difference image for the red channel is shown in (d). Notice that the pixelwise error given by (15) is less than 1 on scale of $[0, 255]$.

III. Fast Algorithm

As with the standard bilateral filter, the brute-force computation of (2) can be expensive. The per-pixel complexity of (2) is $O(\omega_i^2)$. Since we intend to make ω_i large in homogeneous and textured regions, the overall computation can be expensive as a result. As for the standard bilateral filter, there exists algorithms that can accelerate the computations using some form of approximation [45], [46], [47], [24]. More specifically, the speedup is obtained by approximating the bilateral filter using fast convolutions. These are referred to as $O(1)$ algorithms in the literature since their complexity does not depend on the scale of the spatial filter.

It turns out that the above fast algorithms can also be used for scale-adaptive bilateral filtering, with the main difference that each convolutions is replaced by a linear averaging. The latter is not as fast as a convolution (we cannot use FFT for example), but can nonetheless be computed at $O(1)$ cost. In particular, given some scale map (ω_i) and an input image $f(i)$, we define the averaging (without the normalization):

$$\bar{f}(i) = \sum_{j \in \Omega_i} f(j), \tag{7}$$

where Ω_i is as in (1). We next demonstrate how the approximation in [24] can be used to express (2) in terms of (7).

In [24], the Gaussian range kernel is approximated using its Fourier expansion. In particular, notice that the domain of the variable t in (3) is $\{-T, \ldots, 0, \ldots, T\}$, where

$$T = \max_i \max_{j \in \Omega_\infty} |f(i-j) - f(i)|, \quad \Omega_\infty = [-\omega_\infty, \omega_\infty]^2.$$

Following [24], we consider the Fourier approximation of (3) over the period $[-T, T]$:

$$\widehat{\varphi}(t) = \sum_{n=-N}^{N} c_n \exp(\iota n \nu t), \tag{8}$$

where $\iota = \sqrt{-1}, \nu = \pi/T$, and N is the order. The latter is used to obtain a tradeoff between approximation fidelity and run time. Details on how N and the coefficients (c_n) are fixed in (8) can be found in [24].

The proposed approximation of (2) is obtained by using (8) as a surrogate for (3). That is, we consider the approximation

$$\hat{f}_{\mathrm{BF}}(i) = \frac{P(i)}{Q(i)}, \tag{9}$$

where

$$P(i) = \sum_{j \in \Omega_i} \widehat{\varphi}\big(f(j) - f(i)\big) f(j), \tag{10}$$

and

$$Q(i) = \sum_{j \in \Omega_i} \widehat{\varphi}\big(f(j) - f(i)\big). \tag{11}$$

Note that we can express (10) in terms of (7):

$$P(i) = \sum_{n=-N}^{N} c_n \exp\big(\iota n \nu f(i)\big) \bar{p}_n(i), \tag{12}$$

where

$$p_n(i) = \exp\big(-\iota n \nu f(i)\big) f(i).$$

Similarly, we can express (11) as

$$Q(i) = \sum_{n=-N}^{N} c_n \exp\big(\iota n \nu f(i)\big) \bar{q}_n(i), \tag{13}$$

where

$$q_n(i) = \exp\big(-\iota n \nu f(i)\big).$$

In summary, we can compute (10) and (11) in terms of (7) and simple pointwise operations on the input image. Furthermore, as is well-known [48], [49], we can recursively compute (7) at $O(1)$ complexity, i.e., using a fixed number of operations for any scale map (ω_i). This is done as follows. We first construct the integral image:

$$F(i) = F(i_1, i_2) = \sum_{k_1=1}^{i_1} \sum_{k_2=1}^{i_2} f(k_1, k_2). \tag{14}$$

The brute-force computation of (14) is expensive. However, note that we can compute (14) using recursion:

$$\begin{aligned} F(i_1 + 1, i_2 + 1) = {} & f(i_1 + 1, i_2 + 1) + F(i_1 + 1, i_2) \\ & + F(i_1, i_2 + 1) - F(i_1, i_2). \end{aligned}$$

This requires three additions per pixel. Having computed (14), we can obtain (7) as follows:

$$\begin{aligned} \bar{f}(i_1, i_2) = {} & F(i_1 + \omega_i, i_2 + \omega_i) - F(i_1 - \omega_i - 1, i_2 + \omega_i) \\ & - F(i_1 + \omega_i, i_2 - \omega_i - 1) + F(i_1 - \omega_i - 1, i_2 - \omega_i - 1). \end{aligned}$$

Again, we require three additions per pixel. The above formula can be easily verified by plugging (14) in (III) and recalling definition (7). In summary, we can compute (7) using just six additions per pixel, and this is true for

(a) Input. (b) Iteration 1. (c) Iteration 2. (d) Iteration 3.

Fig. 5. Iterative smoothing. The parameters are $\omega_0 = 0, \omega_\infty = 15, \lambda = 4$, and $\sigma = 40$. The output of an iteration is used as an input for the next iteration. The original scale map is used in each iteration. The background texture is almost completely smoothed out after three iterations. However, there is some loss of shading compared to the result after the first iteration.

(a) Input. (b) Iteration 1. (c) Iteration 2. (d) Iteration 3.

Fig. 6. Same as in Figure 5. Notice that the strong texture within the red box is smoothed out only after three iterations.

any scale map. Since (12) and (13) are based on (7), this establishes our claim that we can compute (9) at $O(1)$ complexity.

Finally, we comment on the filtering error introduced by the proposed approximation. By filtering error, we mean the difference between the outputs of the brute-force implementation and the fast algorithm. In particular, we define the error to be

$$\|f_{\mathrm{BF}} - \hat{f}_{\mathrm{BF}}\|_\infty = \max_i \ |f_{\mathrm{BF}}(i) - \hat{f}_{\mathrm{BF}}(i)|, \tag{15}$$

which captures the largest offset in pixel intensity incurred by the approximation.

Since (10) and (11) are computed exactly, the error comes from the use of the surrogate range kernel (9). Therefore, we consider the error

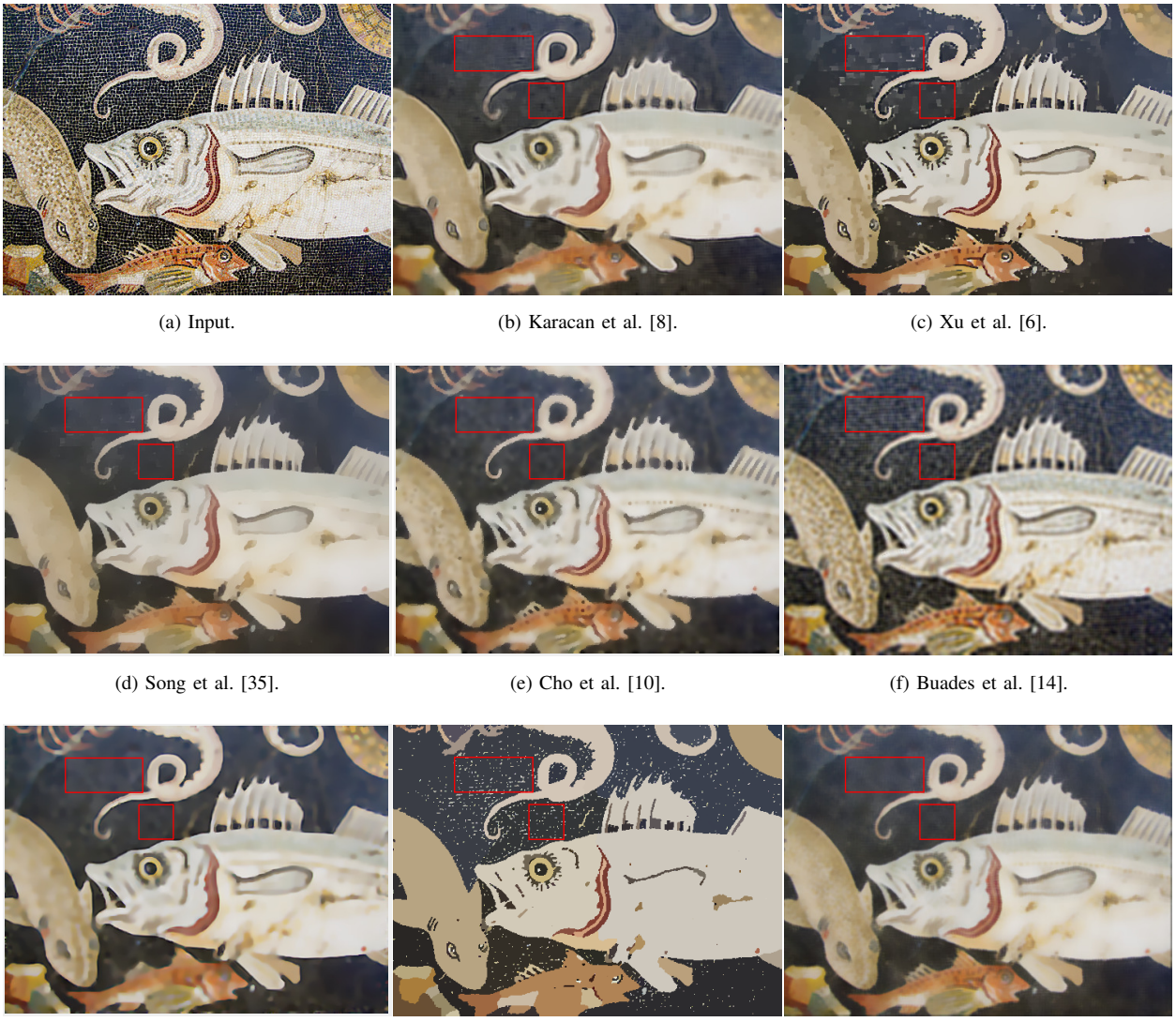
$$\|\varphi - \widehat{\varphi}\|_\infty = \max\Big\{|\varphi(t) - \hat{\varphi}(t)| : t \in \{0, \ldots, T\}\Big\}. \tag{16}$$

Using the analysis from [24], we can bound (15) by (16). This is of practical importance, since we can control the order N in (9) to ensure that (16) is within some tolerance $\varepsilon > 0$. This can be done using the procedure in [24]. In this case, we are guaranteed that

$$\|f_{\mathrm{BF}} - \hat{f}_{\mathrm{BF}}\|_\infty \leq \frac{2M\varepsilon}{c - \varepsilon}, \tag{17}$$

where $c = 1/(2\omega_\infty + 1)^2$ and M is the dynamic range. The derivation of (17) is provided in the Appendix.

The above bound is far from being tight. Moreover, it holds only if $\varepsilon < c$. However, the bound is of theoretical interest. In particular, it follows from (17) that the filtering error can be made arbitrarily small by making ε (resp.



(a) Input. (b) Karacan et al. [8]. (c) Xu et al. [6].

(d) Song et al. [35]. (e) Cho et al. [10]. (f) Buades et al. [14].

(g) Szolgay and Szirányi [15]. (h) Ono [11]. (i) Proposed.

Fig. 7. Comparison of texture smoothing methods. The results in (b)-(h)were generated using the code provided by the authors, with recommended parameter settings. The parameters for our method are $\omega_0 = 0, \omega_\infty = 15, \lambda = 4$, and $\sigma = 30$.

N) sufficiently small (resp. large). On the other hand, the acceleration achieved using the latter is significant (see Figure 4).

IV. Performance Analysis

A. Parameter settings

The parameters for the proposed method are $\Theta, \Gamma, \lambda, \omega_0, \omega_\infty$ and σ. We empirically found that reasonable sizes of Γ and Θ are 3×3 and 7×7. However, we also noticed that the filtering output is not very sensitive to these parameters. The recommended values of λ are in the range $[0.1, 10]$. We have used $\omega_0 = 0$ for all the experiments.

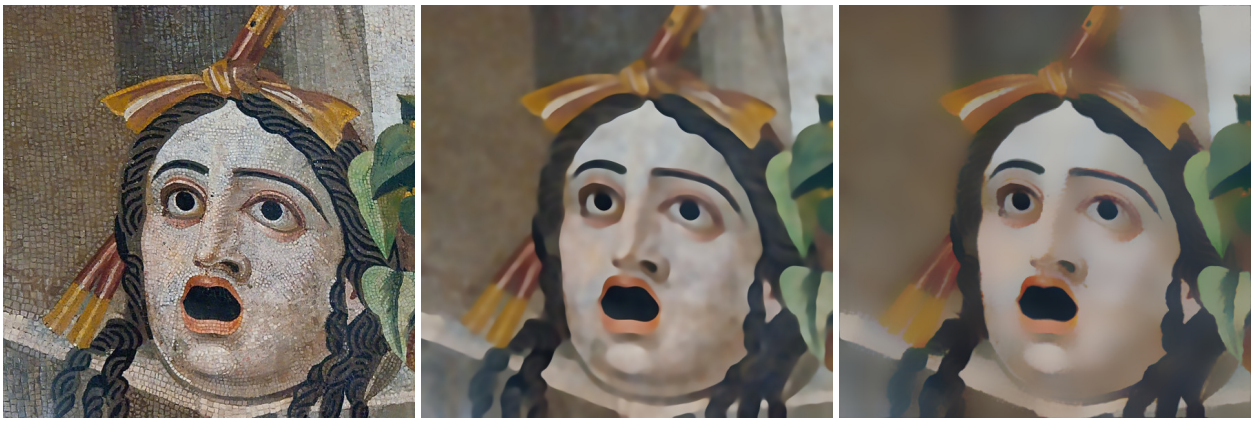
This ensures that ω_i is near-zero if the structural measure m_i is close to 1; no aggregation is performed in this case. We found that the values of Θ and ω_∞ depend strongly on the coarseness of the texture. For most of our experiments, ω_∞ is in the range $[10, 20]$. It is difficult to automatically adjust the parameters for real applications [15]. In fact, the user might wish to tune the parameters (around the recommended values) to explore different tradeoffs between texture smoothing and preservation of shading and edges. We used $\varepsilon = 0.01$ (tolerance parameter) in most of our experiments. We note that by relaxing ε, it is possible to reduce the timing without significantly impacting the visual quality.

B. Iterative mode

As in [10], we tried iterating the proposed filtering. In each iteration, we used the output of the previous iteration as the input to the adaptive bilateral filter. However, we used the same scale map (computed from the original image) for all iterations. Since the strong edges inevitably get weaker after each filtering, we gradually decrease σ to induce less blurring in structured regions. In particular, starting with some initial σ_0, the value of σ is set to be $\gamma^n \sigma_0$ at iteration $n \geq 1$, where $\gamma < 1$ determines the rate of decrease. For our experiments, we have used $\gamma = 0.7$. A couple of results are shown in Figures 5 and 6. In Figure 5, notice that the texture smoothing improves as the iteration progresses; however, the shading is also gradually lost. On the other hand, as seen in Figure 6, the strong texture pattern in the background (marked with a red box) is satisfactorily smoothed out only after three iterations. Notice that, by decreasing σ at each iteration, we are able to preserve the sharpness of the strong edges. In summary, the decision to iterate depends on the strength of the texture that we wish to remove and is difficult to automate. The user might wish to look at the results of the first few iterations and pick one of them. We note that the idea of iterative filtering for depth recovery [50] is related to our use of multiple iterations. Similar to depth recovery, a stable smoothing performance can be obtained for our method by iterating the filter, where the number of iterations depends on the texture strength. However, unlike [50], our framework is purely based on filtering and we do not work with an optimization model.

C. Visual comparisons

We now compare the proposed method with some state-of-the-art methods. One such comparison was already reported in Figure 2. Further comparisons are provided in Figures 7, 9 and 10. We used the original code provided by the authors for the comparisons. From the timings reported in Figure 2, we conclude that the proposed filtering is faster than all methods except [6]. The filtering based method [8] removes textures satisfactorily, but prominent structures are blurred at places. It appears that the latter is essentially due to the limitations of region covariance which is used as a structure measure in [8]. The global optimization framework in [6] exhibits strong edge-preservation. However, the texture smoothing is far from satisfactory. In fact, blocking artifacts occur in the background in Figure 7(c), and there are staircasing artifacts near dominant edges. Fine details are not fully smoothed out in the highlighted homogeneous regions. The performance of [4] is sensitive to the choice of smoothing parameters. As seen in Figure 2(e), strong smoothing results in blurring and contrast degradation. Joint

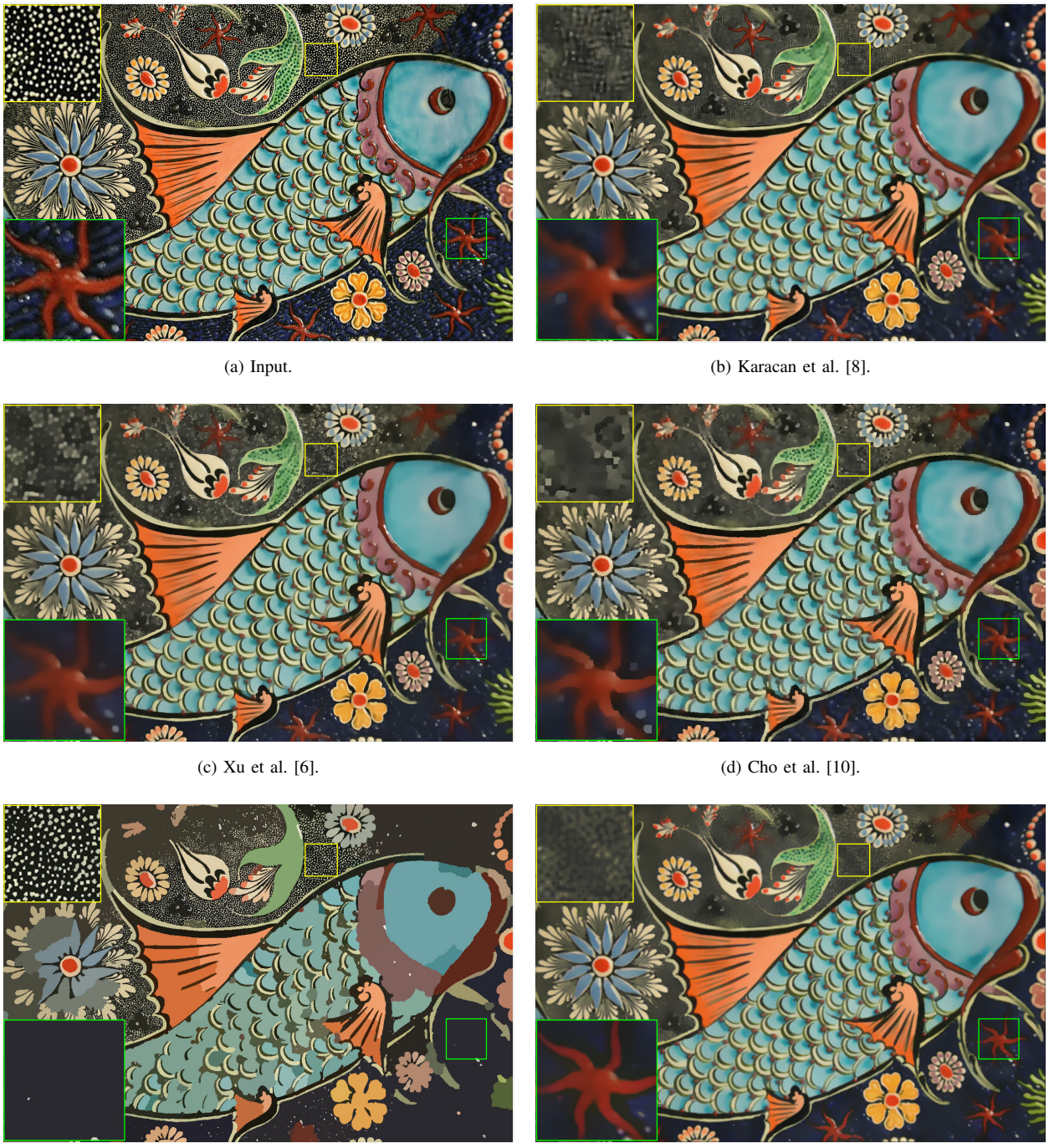


(a) Input. (b) Rolling guidance. (c) Proposed.

Fig. 8. Comparison of the proposed method with rolling guidance filtering [16]. Notice that textures are not smoothed out satisfactorily by [16]. Our method performs better smoothing throughout the image.

bilateral filtering [10] suffers from blocking, and fails to satisfactorily smooth out textures. Surprisingly, local TV [14] seems to perform quite poorly. Adaptive smoothing [15] outperforms [14] in terms of texture suppression, but introduces heavy blurring. The recent method [11] produces oversmoothing of details. Moreover, the surface shading is often obscured in [11], and the strong texture patterns are not smoothed out satisfactorily. In terms of visual quality, we see that our results are generally comparable and sometimes better than existing methods. In particular, the edge preservation is better for our method in the highlighted regions in Figures 9 and 10 (marked with boxes). Moreover, our method preserves subtle shadings and does not suffer from staircasing or blocking. Based on exhaustive experiments, we found that the performance of our and existing methods ultimately depends on the nature of the texture pattern in the image. In fact, there is a tradeoff between edge preservation and smoothing that applies to all methods. Finally, we present a comparison solely with rolling guidance filter [16] in Figure 8. For this example, we notice that the rolling guidance filter performs poor texture smoothing compared to our method. This is clear from the texture on the face and background.

We note that the existing methods [6], [9], [10], [11], [14] rely purely on visual assessment for judging the texture filtering quality; they do not use any ground-truth for quantitative comparisons. Nevertheless, for completeness, we perform an experiment using a synthetic textured image (used in [15]), for which the ground-truth for the structured component (without texture) is available. The results are shown in Figure 11. The PSNR and SSIM [51] values between the ground-truth and the filtered output are provided in the caption. For comparison, we also report the result obtained using [6], and the corresponding PSNR and SSIM values. Our method is found to perform better in terms of both metrics.

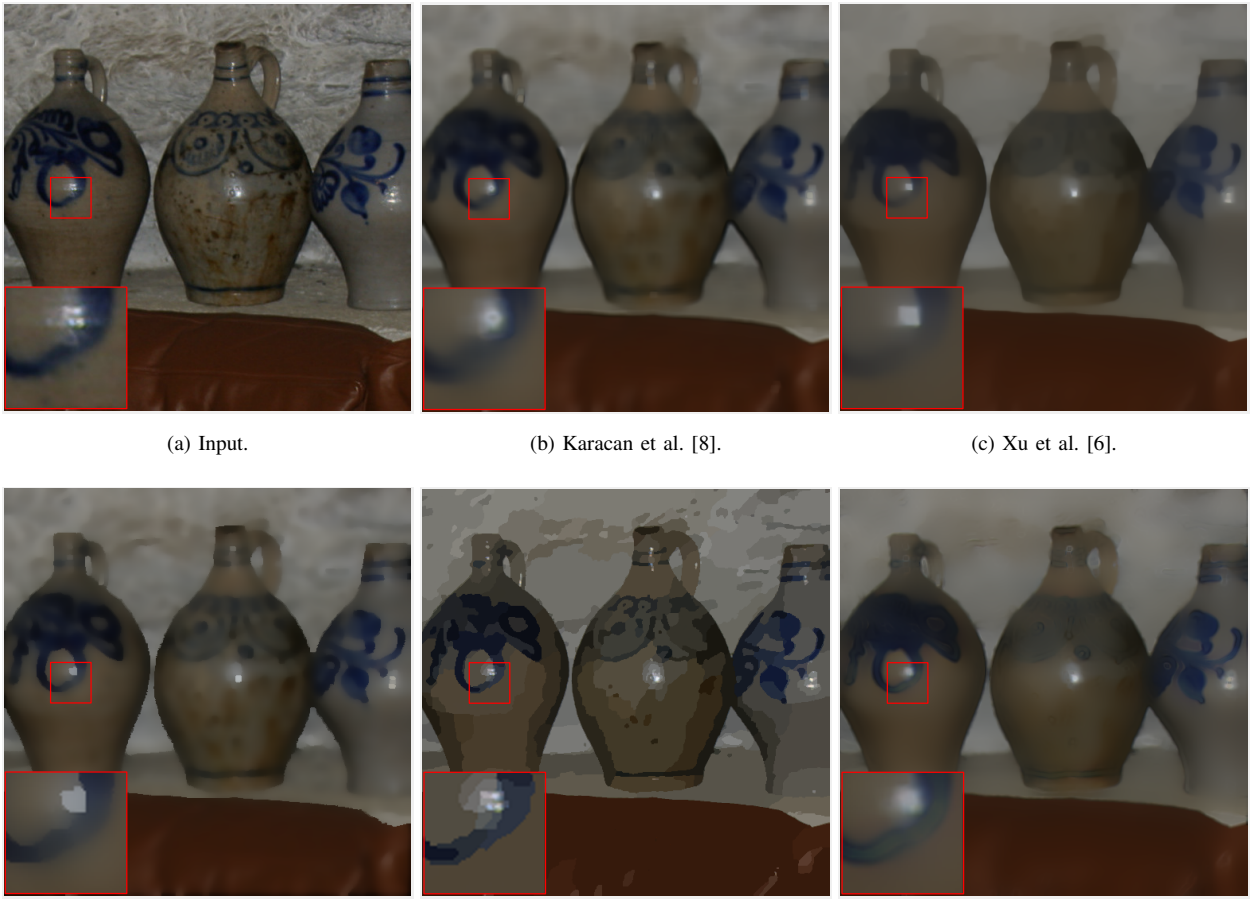


(a) Input. (b) Karacan et al. [8].

(c) Xu et al. [6]. (d) Cho et al. [10].

(e) Ono [11]. (f) Proposed.

Fig. 9. Comparison of texture smoothing methods. The parameters for our method are $\omega_0 = 0, \omega_\infty = 15, \lambda = 4$, and $\sigma = 30$. Notice that our method preserves edges better compared to other methods (green boxes), while also smoothing out textures (yellow boxes).



(a) Input. (b) Karacan et al. [8]. (c) Xu et al. [6].

(d) Cho et al. [10]. (e) Ono [11]. (f) Proposed.

Fig. 10. Comparison of texture smoothing methods. The parameters of our method are as in Fig. 9. Our method preserves edges better than [8] and [6], and smooths textures better than [10] and [11]. Though the edge preservation is better for [11], it fails to smooth out the fine texture in the background.

(a) Input (textured image). (b) Ground truth (structure). (c) [6], 29.76 dB, 94.59. (d) Proposed, 31.77 dB, 97.99.

Fig. 11. Texture smoothing on a synthetic image [15] for which the ground-truth structure (without texture) is available. The structures recovered using [6] and the proposed method are shown in (c) and (d). PSNR and SSIM [51] values (with respect to the ground truth) are mentioned in the caption.

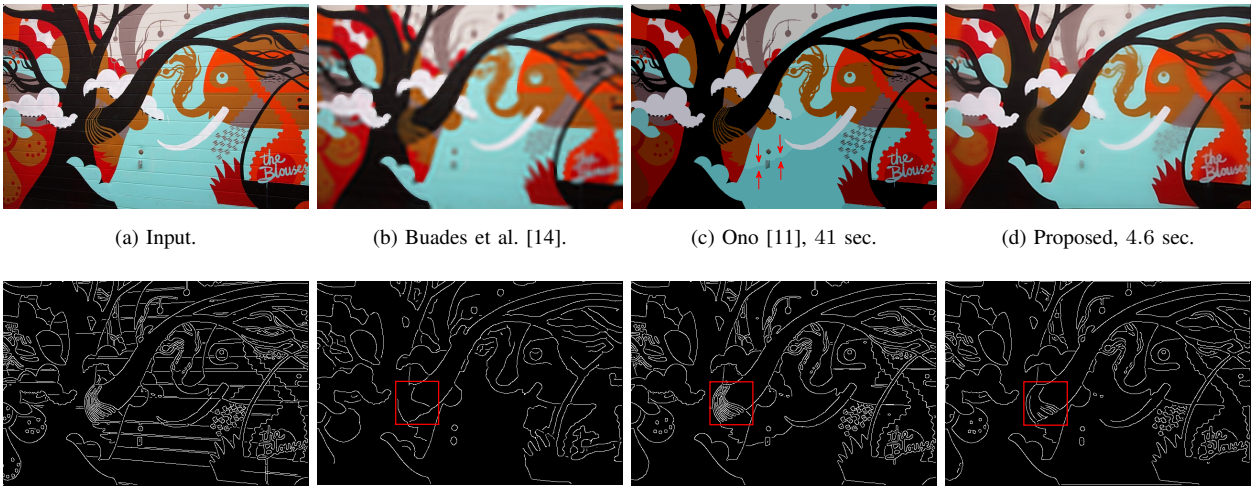
(a) Input. (b) Ono, (0.615, 0.169). (c) Prop., (0.098, 0.338). (d) Zoom from (a) (e) Zoom from (b). (f) Zoom from (c).

Fig. 12. Reduction of blocking artifacts from JPEG compression. The NJQA [52], RMB [53] values are mentioned in the caption. For NJQA, a smaller value indicates higher quality. On the other hand, higher RMB indicates better quality.

V. Applications

A. Compression artifact removal

We use the proposed algorithm for reducing compression artifacts from JPEG images. An images compressed at low-bit rate using standard JPEG usually exhibits blocking artifacts around boundaries [5], [10], [11]. An example is provided in Figure 12, where we also compare our result with [11]. Notice that the proposed filtering suppresses the blocking and restores the sharpness of the boundaries (see zoomed sections). Further, we used two no-reference quality metrics, NJQA [52] and RMB [53], to evaluate the quality of the enhanced images. For NJQA, a smaller value indicates better quality, whereas for RMB, a higher value indicates better quality. We see that our method performs better in terms of both metrics.



(a) Input. (b) Buades et al. [14]. (c) Ono [11], 41 sec. (d) Proposed, 4.6 sec.

(e) Edges, (652, 440, 299). (f) Edges, (161, 119, 111). (g) Edges, (399, 313, 220). (h) Edges, (104, 103, 103).

Fig. 13. Texture smoothing (top row) followed by Canny edge detection (bottom row). The input (a) contains textures, which produces many false edges in (e) if we directly apply the edge detector. To suppress the false edges, we filter out the texture using various methods; the resulting images are shown in (b)-(d). The results of edge detection are shown in (f)-(h). In particular, notice the false edges obtained using [11] (marked with red arrows in (c) and with a box in (g)). The edge count for each color channel is mentioned in the caption.

(a) Input. (b) Ono [11], 35.06.

(c) Xu et al. [6], 31.86. (d) Proposed, 29.64.

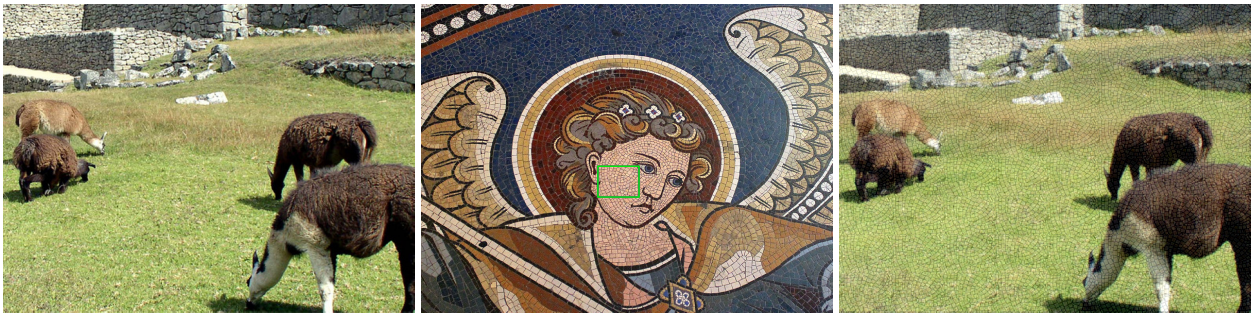
Fig. 14. Detail enhancement using amplification $\theta = 4$. The NIQE [54] metric is mentioned in the caption (smaller NIQE indicates better quality).

B. Edge detection

Edge extraction is a fundamental task in image analysis and high-level inference. Directly applying an edge detector on an image with textures results in many false edges. An obvious solution is to suppress textures (along with noise and other fine details) and then extract the edges. We have used the proposed method for the former step. An example is provided in Figure 13, wherein we have also compared with existing methods [11], [14]. The popular Canny edge detector was used for this example. To quantify the effect of smoothing, we count the number of detected edges with and without preprocessing. We notice that the proposed algorithm is able to detect prominent edges and produces less spurious edges compared to other methods (see the highlighted region).

C. Detail enhancement

The goal of detail enhancement is to improve the visual appearance of an image. Most detail enhancement techniques use edge-preserving filtering [4], [9], [10], [8]. The image is first decomposed into base and detail layers. The former primarily consists of homogeneous regions and sharp edges, while the detail layer captures fine details and textures. In the present context, we compute the base layer using the proposed method. The detail layer is just the difference image $d(i) = f(i) - \hat{f}_{\mathrm{BF}}(i)$. The enhanced image is set to be $f_{\mathrm{e}}(i) = \hat{f}_{\mathrm{BF}}(i) + \theta \cdot d(i)$, where $\theta > 1$ is an amplification factor. A representative result obtained using our method is shown in Figure



(a) Input. (b) Source of texture. (c) After texture transfer.

Fig. 15. The texture from the highlighted region in (b) is transferred to (a) using the proposed method. Best viewed on a computer screen.

14. For completeness we have also shown the results obtained using [11] and [6]. To quantitatively evaluate the performance, we use the no-reference quality metric NIQE [54]. A small NIQE value indicates better image quality. The image obtained using our method has the smallest NIQE value.

D. Texture transfer

Texture transfer refers to copying textures from one image (source) to another image (target). The details of how texture transfer is performed can be found in [9], and we have adopted this in the present case. We have simply used our algorithm to smooth the textures in the source and target images. In brief, a texture-smoothed version of the source is subtracted from the source to obtain its texture component. A small portion of this (green box in Figure 15(b)) is then repeated to cover the size of the target. The final output is added to a smoothed version of the target Figure 15(a) to get the result in Figure 15(c).

VI. Conclusion

We proposed a novel algorithm for edge-preserving texture smoothing using adaptive bilateral filtering. Our contribution was twofold. First, we introduced a simple rule for adapting the width of the spatial kernel using local gradient information. The proposed modification retains the simplicity of the classical bilateral filter. Moreover, we demonstrated that adaptive bilateral filtering can be used for texture smoothing, and the results are often superior to that obtained using state-of-the-art methods in terms of visual quality. Second, we proposed a fast approximation of adaptive bilateral filtering that can cut down the timing of brute-force computation by almost an order. The speedup stems from the fact that the cost of the fast algorithm does not scale with size of spatial kernel. An attractive aspect of the fast algorithm is that it can be tuned to achieve different trade-offs between run time and filtering accuracy. Finally, we demonstrated the usefulness of our method in the context of various image processing applications.

VII. Appendix

In this section, we give the derivation of (17). We can write (2) as $f_{\mathrm{BF}}(i) = P_1(i)/Q_1(i)$, where

$$P_1(i) = \sum_{j\in\Omega_i} \varphi\big(f(j)-f(i)\big)f(i-j),$$

and

$$Q_1(i) = \sum_{j\in\Omega_i} \varphi\big(f(j)-f(i)\big).$$

From (9), we can write $f_{\mathrm{BF}}(i) - \hat{f}_{\mathrm{BF}}(i)$ as

$$\frac{1}{Q(i)}\Big(f_{\mathrm{BF}}(i)\big(Q(i)-Q_1(i)\big) + P_1(i) - P(i)\Big). \tag{18}$$

To obtain an upper bound on (18), we will upper bound its numerator and lower bound its denominator. Since the kernel (3) is nonnegative,

$$Q_1(i) = \varphi(0) + \text{positive terms} \geq 1.$$

On the other hand, since $\|\varphi - \widehat{\varphi}\|_\infty \leq \varepsilon$, we have

$$|Q(i) - Q_1(i)| \leq \varepsilon|\Omega_i|. \tag{19}$$

Therefore, from the inverse triangle inequality, we have

$$|Q(i)| \geq Q_1(i) - |Q(i)-Q_1(i)| \geq 1 - \varepsilon|\Omega_i|. \tag{20}$$

Note that the averaging in (2) can be expressed as a convex combination of pixel intensities. Thus, if M is the dynamic range of f, then $\|f_{\mathrm{BF}}\|_\infty \leq M$. As a result, we obtain

$$\|P_1 - P\|_\infty \leq \sum_{j\in\Omega_i} M\|\varphi - \widehat{\varphi}\|_\infty \leq M\varepsilon|\Omega_i|. \tag{21}$$

The numerator of (18) can be bound as

$$\begin{aligned}
&|f_{\mathrm{BF}}(i)\big(Q(i)-Q_1(i)\big) + P_1(i) - P(i)| \\
&\leq \|f_{\mathrm{BF}}\|_\infty\|Q - Q_1\|_\infty + \|P_1 - P\|_\infty \\
&\leq 2M\varepsilon|\Omega_i|,
\end{aligned}$$

where we have used (19) and (21). Combining this with (20), namely the lower bound on the denominator of (18), we obtain

$$\|f_{\mathrm{BF}} - \hat{f}_{\mathrm{BF}}\|_\infty \leq \frac{2M\varepsilon|\Omega_i|}{1-\varepsilon|\Omega_i|} = \frac{2M\varepsilon}{|\Omega_i|^{-1}-\varepsilon}.$$

Since $|\Omega_i| = (2\omega_i+1)^2 \leq (2\omega_\infty+1)^2$, we finally arrive at (17):

$$\|f_{\mathrm{BF}} - \hat{f}_{\mathrm{BF}}\|_\infty \leq \frac{2M\varepsilon}{c-\varepsilon},$$

where $c = 1/(2\omega_\infty+1)^2$.

VIII. Acknowledgements

We thank the authors of [4], [6], [8], [11], [35] for making their codes and images publicly available. We are grateful to Hojin Cho for sharing the images from [10] and to Prof. Tamas Sziranyi for the dataset used in Figure 11. We also thank the editor and the reviewers for their comments and suggestions that have helped improve the manuscript.

References

[1] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 12, no. 7, pp. 629-639, 1990.

[2] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *Proc. IEEE International Conference on Computer Vision*, pp. 839-846, 1998.

[3] M. Zhang and B. K. Gunturk, "Multiresolution bilateral filtering for image denoising," *IEEE Transactions on Image Processing,* vol. 17, no. 12, pp. 2324-2333, 2008.

[4] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM Transactions on Graphics,* vol. 27, no. 3, pp. 67:1-67:10, 2008.

[5] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via L_0 gradient minimization," *ACM Transactions on Graphics,* vol. 30, no. 6, pp. 174:1-174:11, 2011.

[6] L. Xu, C. Lu, Y. Xu, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Transactions on Graphics,* vol. 31, no. 6, pp. 139:1-139:10, 2012.

[7] B. Gu, W. Li, M. Zhu, and M. Wang, "Local edge-preserving multiscale decomposition for high dynamic range image tone mapping," *IEEE Transactions on Image Processing*, vol. 22, no. 1, pp. 70-79, 2013.

[8] A. Karacan, E. Erdem, and A. Erdem, "Structure preserving image smoothing via region covariances," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 176:1-176:11, 2013.

[9] Z. Su, X. Luo, Z. Deng, Y. Liang, and Z. Ji, "Edge-preserving texture suppression filter based on joint filtering schemes," *IEEE Transactions on Multimedia*, vol. 15, pp. 535-548, 2013.

[10] H. Cho, H. Lee, H. Kang, and S. Lee, "Bilateral texture filtering," *ACM Transactions on Graphics,* vol. 33, no. 4, pp. 128:1-128:8, 2014.

[11] S. Ono, "L_0 gradient projection," *IEEE Transactions on Image Processing,* vol. 26, no. 4, pp. 1554-1564, 2017.

[12] V. Aurich and J. Weule, "Non-linear Gaussian filters performing edge preserving diffusion," *Mustererkennung*, pp. 538-545, 1995.

[13] S. M. Smith and J. M. Brady, "SUSAN–A new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45-78, 1997.

[14] A. Buades, T. M. Le, J. M. Morel, and L. A. Vese, "Fast cartoon + texture image filters," *IEEE Transactions on Image Processing,* vol. 19, no. 8, pp. 1978-1986, 2010.

[15] Dániel Szolgay and Tamás Szirányi, "Adaptive image decomposition into cartoon and texture parts optimized by the orthogonality criterion," *IEEE Transactions on Image Processing*, vol. 21, no. 8, pp. 3405-3415, 2012.

[16] Q. Zhang, X. Shen, L. Xu, and J. Jia, "Rolling guidance filter," *Proc. European Conference on Computer Vision,* pp. 815-830, 2014.

[17] P. Xu and W. Wang, "Improved bilateral texture filtering with edge-aware measurement," *IEEE Transactions on Image Processing,* vol. 27, no. 7, pp. 3621-3630, 2018.

[18] T. H. Lin, D. L. Way, Z. C. Shih, W. K. Tai, and C. C. Chang, "An efficient structure-aware bilateral texture filtering for image smoothing," *Proc. Pacific Conference on Computer Graphics and Applications,* pp. 57-66, 2016.

[19] T. H. Lin, D. L. Way, Z. C. Shih, W. K. Tai, and C. C. Chang, "An efficient structure-aware bilateral texture filtering for image smoothing," *Computer Graphics Forum,* vol. 35, no. 7, pp. 57-66, 2016.

[20] Z. Zhang and P. Xu, "An efficient learning-based bilateral texture filter for structure preserving," *Proc. International Workshop on Next Generation Computer Animation Techniques,* pp. 149-158, 2017.

[21] Q. Liu, J. Liu, P. Dong, and D. Liang, "SGTD: Structure gradient and texture decorrelating regularization for image decomposition," *Proc. IEEE International Conference on Computer Vision,* pp. 1081-1088, 2013.

[22] J. F. Aujol and G. Gilboa, "Constrained and SNR-based solutions for TV-Hilbert space image denoising," *Journal of Mathematical Imaging and Vision*, vol. 26, pp. 217-237, 2006.

[23] L. Kovacs and T. Sziranyi, "Focus area extraction by blind deconvolution for defining regions of interest," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 29, no. 6, pp. 1080-1085, 2007.

[24] S. Ghosh and K. N. Chaudhury, "On fast bilateral filtering using Fourier kernels," *IEEE Signal Processing Letters,* vol. 23, no. 5, pp. 570-573, 2016.

[25] K. Sugimoto and S. I. Kamata, "Compressive bilateral filtering," *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3357-3369, 2015.

[26] Y. Zang, H. Huang, and L. Zhang, "Efficient structure-aware image smoothing by local extrema on space-filling curve," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 9, pp. 1253-1265, 2014.

[27] Y. Zang, H. Huang, and L. Zhang, "Guided adaptive image smoothing via directional anisotropic structure measurement," *IEEE Transactions on Visualization and Computer Graphics,* vol. 21, no. 9, pp. 1015-1027, 2015.

[28] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35, no. 6, pp. 1397-1409, 2013.

[29] K. Subr, C. Soler, and F. Durand, "Edge-preserving multiscale image decomposition based on local extrema," *ACM Transactions on Graphics,* vol. 28, no. 5, pp. 147:1-147:9, 2009.

[30] J. F. Aujol, G. Gilboa, T. Chan, and S. Osher, "Structure-texture image decomposition – modeling, algorithms, and parameter selection," *International Journal of Computer Vision,* vol. 67, no. 1, pp. 111-136, 2006.

[31] H. Schaeffer and S. Osher, "A low patch-rank interpretation of texture," *SIAM Journal on Imaging Sciences,* vol. 6, no. 1, pp. 226-262, 2013.

[32] S. Ono, T. Miyata, and I. Yamada, "Cartoon-texture image decomposition using blockwise low-rank texture characterization," *IEEE Transactions on Image Processing*, vol. 23, no. 3, pp. 1128-1142, 2014.

[33] Y-R Fan, T-Z. Huang, T-H. Ma, and X-L. Zhao, "Cartoontexture image decomposition via non-convex low-rank texture regularization," *Journal of the Franklin Institute,* vol. 354, no. 7, pp. 3170-3187, 2017.

[34] T. N. Canh, K. Q. Dinh, and B. Jeon, "Detail-preserving compressive sensing recovery based on cartoon texture image decomposition," *Proc. IEEE International Conference on Image Processing*, pp. 1327-1331, 2014.

[35] J. Song, H. Cho, J. Yoon, and S. M. Yoon, "Structure adaptive total variation minimization-based image decomposition," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 28, no. 9, pp. 2164-2176, 2018.

[36] B. Y. Zhang, J. P. Allebach, "Adaptive bilateral filter for sharpness enhancement and noise removal," *IEEE Transactions on Image Processing,* vol. 17, no. 5, pp. 664-678, 2008.

[37] K. N. Chaudhury, A. Munoz-Barrutia, and M. Unser, "Fast space-variant elliptical filtering using box splines," *IEEE Transactions on Image Processing,* vol. 19, no. 9, pp. 2290-2306, 2010.

[38] M. Zhang and B. K. Gunturk, "Compression artifact reduction with adaptive bilateral filtering," *SPIE Electronic Imaging,* vol. 7257, pp. 72571A, 2009.

[39] S. Mangiat and J. Gibson, "Spatially adaptive filtering for registration artifact removal in HDR video," *Proc. IEEE International Conference on Image Processing,* pp. 1317-1320, 2011.

[40] J. Jeon, H. Lee, H. Kang, and S. Lee, "Scaleaware StructurePreserving Texture Filtering," *Computer Graphics Forum,* vol. 35, no. 7, pp. 77-86, 2016.

[41] C. Kervrann, "An adaptive window approach for image smoothing and structures preserving," *Proc. European Conference on Computer Vision,* pp. 132-144, Berlin, 2004.

[42] S. Chen and R. M. Haralick, "Recursive erosion, dilation, opening, and closing transforms," *IEEE Transactions on Image Processing,* vol. 4, no. 3, pp. 335-345, 1995.

[43] A. Jain, "Fundamentals of Digital Image Processing," *Prentice-Hall,* p 384, 1986,

[44] K. N. Chaudhury, "Acceleration of the shiftable $O(1)$ algorithm for bilateral filtering and nonlocal means," *IEEE Transactions on Image Processing,* vol. 22, no. 4, pp. 1291-1300, 2013.

[45] F. Porikli, "Constant time $O(1)$ bilateral filtering," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.

[46] Q. Yang, K. H. Tan, and N. Ahuja, "Real-time $O(1)$ bilateral filtering," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 557-564, 2009.

[47] K. N. Chaudhury, D. Sage, and M. Unser, "Fast $O(1)$ bilateral filtering using trigonometric range kernels," *IEEE Transactions on Image Processing*, vol. 20, no. 12, pp. 3376-3382, 2011.

[48] F. Crow, "Summed-area tables for texture mapping", *Proc. ACM Siggraph*, vol. 18, no. 3, pp. 207-212, 1984.

[49] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 511-518, 2001.

[50] J. Yang, X. Ye, and P. Frossard, "Global Auto-Regressive Depth Recovery via Iterative Non-Local Filtering," *IEEE Transactions on Broadcasting*, (2018).

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, **13**(4), pp. 600-612 (2004).

[52] S. A. Golestaneh, and D. M. Chandler, "No-reference quality assessment of JPEG images via a quality relevance map," *IEEE Signal Processing Letters,* vol. 21, no. 2, pp. 155-158, 2014.

[53] L. Li, H. Zhu, G. Yang, and J. Qian, "Referenceless measure of blocking artifacts by Tchebichef kernel analysis," *IEEE Signal Processing Letters,* vol. 21, no. 1, pp. 122-125, 2014.

[54] A. Mittal, R. Soundararajan, and A. C. Bovik, "Making a "completely blind" image quality analyzer," *IEEE Signal Processing Letters,* vol. 20, no. 3, pp. 209-212, 2013.

Sanjay Ghosh received the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 2013. From August 2013 to July 2014, he served as a faculty member at the National Institute of Technology, Jamshedpur, India.

Currently, he is working towards the Ph.D. degree in the Department of Electrical Engineering, Indian Institute of Science, Bangalore, India. His research interests are in image processing and computational imaging.

Ruturaj G. Gavaskar is working towards the Ph.D. degree in the Department of Electrical Engineering, Indian Institute of Science, Bangalore. His research interests include image processing and optimization.

Debasisha Panda received the M.E. degree in signal processing from the Indian Institute of Science, Bangalore, India, in 2017. Since 2017 he has been with Samsung Electromechanics as a developer of image processing algorithms. His research activities focus on development of cross-platform image processing solutions for industrial automation, computer vision based solutions and deep learning networks.

Kunal N. Chaudhury is an Assistant Professor in the Department of Electrical Engineering at the Indian Institute of Science, Bangalore. His research areas are image processing, computer vision, and numerical optimization. Dr. Chaudhury is a member of IEEE, SIAM, and a Senior Editor of the SPIE Journal of Electronic Imaging.