The Visual Computer manuscript No.
(will be inserted by the editor)

Fast and Faithful Scale-Aware Image Filters

Shin Yoshizawa · Hideo Yokota

This is a preprint (a post-peer-review, pre-copyedit version) of an article published in The Visual Computer (2021). The final authenticated version is available online at: "https://doi.org/10.1007/s00371-021-02249-5".

Abstract This paper proposes a fast and accurate computational framework for scale-aware image filters. Our framework is based on accurately approximating L^1 Gaussian convolution with respect to a transformed pixel domain representing geodesic distance on a guidance image manifold in order to recover salient edges in a manner faithful to scale-space theory while removing small image structures. Our framework possesses linear computational complexity with high approximation precision. We examined it numerically in terms of speed, accuracy, and quality compared with conventional methods.

Keywords Domain-splitting algorithm · Rolling guidance filter · Domain transform · L^1 Gaussian function

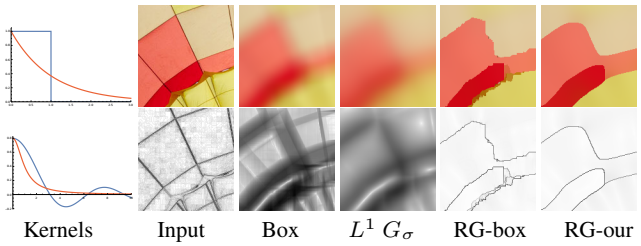

Fig. 1 Top: linear smoothing (center) and scale-aware filtering (right, RG [36]) results with a box and our L^1 Gaussian kernels (top-left, where the bottom-left graphs show their Fourier signals). Bottom images correspond to the magnitudes (square roots) of their gradients.

1 Introduction

Shape and texture structures in images are composed of various scale sizes. Therefore, a multi-scale mathematical framework called scale-space theory [18,33] to represent an image with different scales has been thoroughly studied for many years within pattern recognition and image processing communities. Traditionally, linear convolution of an image with the Gaussian function $G_\sigma(\cdot)$ has been employed to obtain a scale-space representation that smooths out image structures smaller than a given scale parameter $\sigma \in \mathbb{R}_{>0}$[1] as the standard deviation of $G_\sigma(\cdot)$. See the seminal book [21] for the methodology and applications of scale-space theory. The Gaussian convolution also rapidly flattens salient image edges with a size greater than σ, but this is a highly redundant representation of edges over the scale space. Combining smooth image regions with salient edges is therefore more compact and desirable [10].

It is often difficult to control the parameters of popular edge-preserving image filters such as bilateral filters [24] and nonlinear diffusions (especially the number of iterations) to achieve a filtering result with a target scale σ. More recently, *scale-aware filters* [36,19,29,16], which remove image structures smaller than a specific scale while preserving salient edges, have attracted considerable attention because of their stability and simplicity. These scale-aware filters can directly and intuitively specify the target scale σ, and the convergence of their iterative filtering process is very stable compared with conventional edge-aware filters. These filters have therefore been widely used in applications such as edge detection, detail enhancement, and image abstraction. The scale-aware concept has also been extended to 3D meshes [32,35]. Since scale-aware filters usually require time-consuming computations, box-based averaging methods (e.g., [26]) have been employed for their fast approximation [36]. Unfortunately, a box-like averaging kernel that

S. Yoshizawa (the corresponding author: shin@riken.jp) and H. Yokota: Image Processing Research Team, RIKEN, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan

[1] Strictly speaking, it is a high-frequency suppression filter in the Fourier domain, and there is no closed-form relationship between σ and the curvature of a geometric structure in the spatial domain. In contrast, because the fundamental solution of the linear diffusion equation leads to a Gaussian scale-space for some boundary conditions [20], a time variable of the (mean) curvature flow and σ are directly related in continuous cases [23].

includes naive truncation of a Gaussian function often produces undesired artifacts because it leads to a *sinc*-like kernel shape in its Fourier domain. Extension of these methods [4,15,9] may therefore also cause artifacts. See Fig. 1 for an example of artifacts caused by a box filter and its relationship to a scale-aware filter (RG: Rolling Guidance [36]). Although a popular recursive Gaussian filter [6] has been extended to non-uniform pixels [12] and could perhaps be employed for a scale-aware filter, such recursive filters [6, 1,31] have non-trivial numerical issues (e.g., some σ and boundary problems [30,5]) because they optimize their coefficients for fixed parameters. Developing not only a fast but also an accurate approximation of scale-aware filters is thus important because of recent advances in the use of image data for science and engineering as well as images with a high dynamic range (HDR) and high resolution.

In this paper, we propose a novel computational framework for fast and accurate approximation of scale-aware image filters. Our framework is based on adapting domain splitting [34,3] and transformation [11,12] techniques to construct an average-based joint filter that produces a nonlinear color averaging of a given image according to another guidance image. Our scale-aware filters recursively apply the constructed joint filter to recover edge features while removing small image structures. The joint filter consists of a separable implementation of L^1 Gaussian convolutions on a guidance-transformed domain that is equipped with a metric of geodesic distance on an *image manifold* [26] composed of the pixel coordinates and their corresponding guidance image colors. The L^1 Gaussian convolution is approximated very accurately with linear computational complexity by splitting the transformed domain into representative regions where discrete convolutions can be efficiently performed. Figure 2 gives an overview of the framework.

Our framework is faithful to scale-space theory because it implies that the L^1 Gaussian convolution never increases the number of extrema for the one-dimensional continuous case (mentioned as truncated exponential functions by [21] [§6.2.3]). In other words, it does not produce any phantom edges that do not exist in the original image for cases of linear filtering. It is also associated with the absence of ripples in the Fourier domain of the L^1 Gaussian function, as shown in the bottom-left image of Fig. 1. We introduce the implementation of three conventional scale-aware filters [36,19, 29] via our framework in this paper and an assessment of their performances versus the conventional box-based averaging method.

The rest of the paper is organized as follows. We present our scale-aware filters based on joint filters in Section 2. Sections 3 and 4 describe the guidance domain transformation and the domain-splitting Gaussian convolution for the joint filter, respectively. Our numerical experiments are explained in Section 5. We conclude the paper in Section 6.

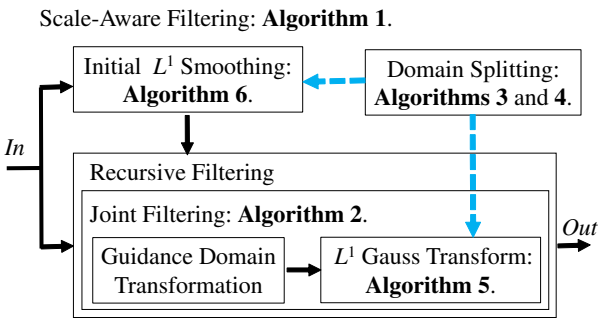

Fig. 2 An overview of our framework. Algorithms are described in the following Sections.

2 Joint-Averaging Scale-Aware Filters

For a given image pixel $\mathbf{x} = \{x_p\}$ on $\mathbb{R}^d$, let $\mathbf{I} = \mathbf{I}(\mathbf{x})$ and $\mathbf{J}^s = \mathbf{J}^s(\mathbf{x})$ on $\mathbb{R}^c$ be the input and the s-th filtered image color of our framework, respectively, where $d, c \in \mathbb{N}$ and $s \in \mathbb{N} \cup \{0\}$. Our scale-aware filter is then recursively defined by

$$\mathbf{J}^{s+1} = \mathbf{F}(\mathbf{x}, \mathbf{f}, \mathbf{J}^s), \quad \mathbf{J}^0 \equiv \frac{\int G_\sigma(\mathbf{x}-\mathbf{y})\mathbf{I}(\mathbf{y})d\mathbf{y}}{\int G_\sigma(\mathbf{x}-\mathbf{y})d\mathbf{y}} \tag{1}$$

where $\sigma \in \mathbb{R}_{>0}$ is a user-specified scale parameter, $G_\sigma(\cdot) = \exp(-\frac{|\cdot|}{\sigma})$ is a L^1 Gaussian function (also known as a Laplace distribution in statistics and probability theory), and $\mathbf{F}$ is a functional of an average-based joint filter $\mathbf{f}$ defined in Eqs. from (2) to (5). Here, $\mathbf{J}^0$ is an initial smoothed image for removing small structures on the input $\mathbf{I}$ by using the normalized Gaussian convolution in Eq. (1). The joint filter $\mathbf{f}$ recovers salient image edges during the above recursive filtering process.

Let $\mathbf{f} = \mathbf{f}(\mathbf{x}, \mathbf{g}, \mathbf{h}) : \mathbb{R}^{d+2c} \to \mathbb{R}^c$ be a joint filter of integrand $\mathbf{h} = \mathbf{h}(\mathbf{x})$ and guidance $\mathbf{g} = \mathbf{g}(\mathbf{x})$ on $\mathbb{R}^c$, respectively. The $\mathbf{f}$ is then given by a normalized convolution:

$$\mathbf{f}(\mathbf{x}, \mathbf{g}, \mathbf{h}) = \frac{\int W(\sigma, \phi, \mathbf{x}, \mathbf{y}, \mathbf{g})\mathbf{h}(\mathbf{y})d\mathbf{y}}{\int W(\sigma, \phi, \mathbf{x}, \mathbf{y}, \mathbf{g})d\mathbf{y}}, \tag{2}$$

$$W(\cdot) = \prod_{p=1}^{d} G_\sigma(T_{p,\lambda}(x_p, \mathbf{x}, \mathbf{g}) - T_{p,\lambda}(y_p, \mathbf{y}, \mathbf{g})), \tag{3}$$

$$\lambda = \sqrt{\sigma/(\sigma_s \phi)} \tag{4}$$

where $\mathbf{y} = \{y_p\} \in \mathbb{R}^d$, $\phi \in \mathbb{R}_{>0}$ is a user-specified edge-awareness parameter, σ_s is the standard deviation of the integrand $\mathbf{h}$, and $T_{p,\lambda}(\cdot) : \mathbb{R}^{d+c+1} \to \mathbb{R}_{>0}$ denotes a domain transformation with respect to the p-th coordinate basis (described in Eq. (7) of Section 3). The transformation $T_{p,\lambda}(\cdot)$ measures a geodesic distance on the image manifold $(x_p, \lambda\,\mathbf{g}) \in \mathbb{R}^{c+1}$ in order to obtain an edge-recovering effect according to the guidance $\mathbf{g}$. Note that the Gaussian convolution $G_\sigma(\cdot)$ in Eq. (3) uses only the scale parameter σ for its variance, but both ϕ and σ characterize the transformation $T_{p,\lambda}(\cdot)$ by λ.

Algorithm 1: Scale-Aware Image Filtering

Input : Pixels $\{\mathbf{x}\}$ and their colors $\{\mathbf{I}(\mathbf{x})\}$, scale σ and edge ϕ parameters, and iteration number s.
Output: Filtered colors $\{\mathbf{J}^{s+1}\}$.

Set σ_s equal to the standard deviation of, for example, the grayscale of $\{\mathbf{I}(\mathbf{x})\}$;
$\lambda \leftarrow \sqrt{\sigma/(\sigma_s \phi)}$; // Equation (4)
$\mathbf{J}^0 \leftarrow \text{GaussianFilter}(\{\mathbf{x}\}, \{\mathbf{I}(\mathbf{x})\}, \sigma)$; // Algorithm 6.
for $i \leftarrow 0$ **to** s **do**
  $\mathbf{J}^{i+1}(\mathbf{x}) \leftarrow \mathbf{F}(\mathbf{x}, \mathbf{f}, \mathbf{J}^i)$ with σ and λ for all $\mathbf{x}$;
  // Equation (5) with Algorithm 2.
end for
return $\{\mathbf{J}^{s+1}\}$;

Filter Models: We have implemented the following three conventional filters in our framework: RG [36], SiR (Smooth and iteratively Restore [19]), and AG (Alternating Guided [29]). The RG literature [36] first proposed a scale-aware filtering framework based on a recursive process involving average-based joint filters such as joint/cross bilateral [25, 8], guided [17], and domain transformation [11] filters. The RG filter smooths the curvature of large-scale edges, whereas the SiR filter preserves curvature by exchanging the use of the integrand and guidance images in the joint filter. The AG filter also improves two SiR drawbacks: reducing intensity and restoring small structures around large-scale edges. Figures 3 and 4 demonstrate the different effects of these filters on the same parameters via our framework, and their stable convergence rates are shown in Fig. 5. Four iterations ($s = 4$) have been recommended for fast results [36], and 20 iterations ($s = 20$) are enough for high-quality results.

RG, SiR, and AG filters in our framework are given by $\mathbf{F} = \mathbf{F}(\mathbf{x}, \mathbf{f}, \mathbf{h}) \in \mathbb{R}^c$ in Eq. (1) such that

$$\mathbf{F}(\mathbf{x}, \mathbf{f}, \mathbf{h}) \equiv \begin{cases} \mathbf{f}(\mathbf{x}, \mathbf{h}, \mathbf{I}) : & \text{RG [36]}, \\ \mathbf{f}(\mathbf{x}, \mathbf{I}, \mathbf{h}) : & \text{SiR [19]}, \\ \mathbf{M}_{\mathbf{x}}(\mathbf{f}(\mathbf{x}, \mathbf{I}, \mathbf{f}(\mathbf{x}, \mathbf{h}, \mathbf{I}))) : & \text{AG [29]} \end{cases} \quad (5)$$

where $\mathbf{M}_{\mathbf{x}} = \mathbf{M}_{\mathbf{x}}(\cdot) \in \mathbb{R}^c$ is a vector median filter [2] (only one-link neighbor pixels, i.e., a 3×3 pixel window for a 2D image). Note that the integrand and guidance ($\mathbf{h}$ and $\mathbf{I}$) are exchanged in $\mathbf{F}$ of the RG and SiR filters on the right-hand side of Eq. (5) and that the AG filter applies the RG and SiR filters alternatively. In contrast to conventional filters [36, 19, 29], our framework is restricted to use of the domain transformation for the joint filter $\mathbf{f}$, as in Eq. (3), in order to apply the fast and accurate Gaussian convolutions described in Section 4. Algorithm 1 shows the pseudocode of our scale-aware filters.

3 Guidance Domain Transformation

The domain transform technique [11, 12] provides edge-aware image smoothing efficiently, and it can be utilized for scale-aware image filters, as demonstrated in the RG filter [36]. The basic idea of this technique is to apply fast linear convolutions on the transformed domain representing the magnitude of the image edge as a metric of the domain (length) by using geodesic distance on an image manifold instead of performing expensive nonlinear convolutions, as illustrated in Fig. 6. We describe here the guidance domain transformation $T_{p,\lambda}(\cdot)$ in Eq. (3) adapted to our framework in order to efficiently perform Gaussian convolutions of $\mathbf{f}$ in Eqs. (2) and (3).

For a given pixel $\mathbf{x} = \{x_p\}$ on $\mathbb{R}^d$, the same pixel as in Section 2, consider its one-dimensional local coordinate

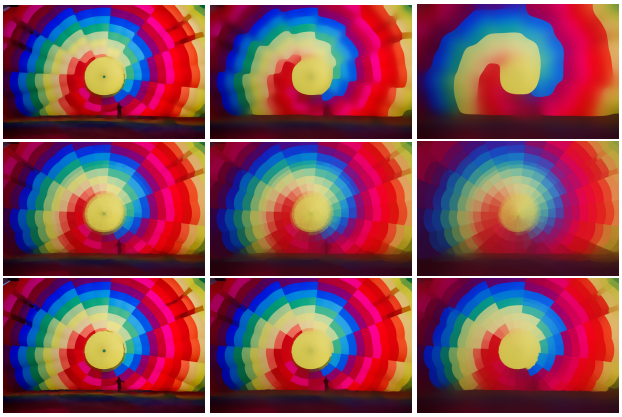

Fig. 3 RG (top), SiR (middle), and AG (bottom) results of filtering the Balloon image [27] via our framework, where $\phi = 1.5$ and $\sigma \in \{16, 32, 64\}$ (left to right).

Fig. 4 Zoomed images of the rightmost panels of Fig. 3: $\sigma = 64$.

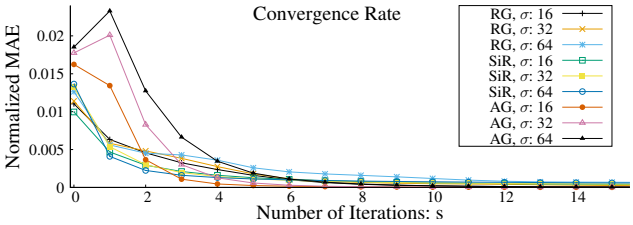

Fig. 5 Convergence rates of our scale-aware filters demonstrated in Fig. 3. Horizontal and vertical axes are the number of iterations s and the normalized MAE (Mean Absolute Error divided by the product of c and $\max(\mathbf{I})$) between $\mathbf{J}^{s+1}$ and $\mathbf{J}^s$.

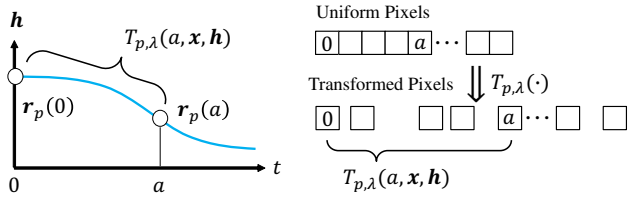

Fig. 6 Uniform pixels are transformed with respect to the geodesic distance of the image manifold to perform convolutions efficiently.

system $\mathcal{S}$ parallel to the p-th coordinate basis. The following straight line $\mathbf{u}_p = \mathbf{u}_p(t, \mathbf{x}) \in \mathbb{R}^d$ passing through $\mathbf{x}$ then reparameterizes $\mathcal{S}$ by $t \in \mathbb{R}$:

$$\mathbf{u}_p = \mathbf{u}_p(t, \mathbf{x}) \equiv \{ \begin{cases} x_l & \text{if } l \neq p, \\ t & \text{Otherwise} \end{cases} \} \tag{6}$$

where the origin of $\mathcal{S}$ is located on $\mathbf{u}_p(0, \mathbf{x})$.

For a given image color $\mathbf{h}(\mathbf{x}) = \{h_q(\mathbf{x})\}$ on $\mathbb{R}^c$ at $\mathbf{x}$, the (λ-scaled) locus of image color $\mathbf{h}(\mathbf{u}_p)$ on the joint space of the p-th pixel coordinate and its color form a hyper curve $\mathcal{C}_p$, i.e. a one-dimensional image manifold,

$$\mathcal{C}_p : \mathbf{r}_p(t) = (t, \lambda\, \mathbf{h}(\mathbf{u}_p)) \in \mathbb{R}^{c+1}$$

where λ, defined in Eq. (4), controls the ratio of the metrics between pixel and color spaces. Note that $\{x_l\} : l \neq p$ of $\mathcal{C}_p$ characterizes a $(d-1)$-parameter family of hyper curves corresponding to each l-th pixel coordinate basis. The convolutions in Eq. (2) are thus performed separately in terms of varying the coordinate element p (see Eq. (3)).

The geodesic distance between two points on $\mathcal{C}_p$ indicates the amount and magnitude of the image edges within its corresponding range of t, because the arc length of $\mathcal{C}_p$ increases when its corresponding color $\mathbf{h}(\mathbf{u}_p)$ changes rapidly with respect to t. The arc length of $\mathcal{C}_p$ within the interval $t \in [0, a]$ is given by integrating the magnitude of its tangent vector [28]: $\int_0^a ||\frac{\partial \mathbf{r}_p(t)}{\partial t}||\, dt = \int_0^a ||(1, \lambda \frac{\partial \mathbf{h}(\mathbf{u}_p)}{\partial t})||\, dt$. Simple computations then yield our guidance domain transformation $T_{p,\lambda}(\cdot)$ (employed in Eq. (3)), as follows:

$$T_{p,\lambda}(a, \mathbf{x}, \mathbf{h}) = \int_0^a \sqrt{1 + \lambda^2 \sum_{q=1}^{c} |\frac{\partial h_q(\mathbf{u}_p)}{\partial t}|^2}\, dt. \tag{7}$$

The above transformation (7) is performed on the image manifold $(\mathbf{x}, \lambda\, \mathbf{g})$ as $T_{p,\lambda}(x_p, \mathbf{x}, \mathbf{g})$ in order to incorporate edge information provided by the guidance $\mathbf{g}$ into the joint filter $\mathbf{f}$ in Eqs. (2) and (3).

Discrete Implementation: As recommended in [11], the first-order partial derivative of $h_q(\mathbf{u}_p)$ with respect to t in Eq. (7) is approximated by using the forward difference scheme:

$$\frac{\partial h_q(\mathbf{u}_p)}{\partial t} \approx h_q(\mathbf{u}_p(t+1, \mathbf{x})) - h_q(\mathbf{u}_p(t, \mathbf{x})).$$

The joint filter $\mathbf{f}$ is also implemented iteratively with the scale variable

$$\sigma_i = \sigma \sqrt{3} (2^{N_{ite} - i}) / \sqrt{4^{N_{ite}} - 1} \tag{8}$$

where N_{ite} is the number of iterations ($N_{ite} = 3$ in our numerical experiments). The pseudocode in Algorithm 2 illustrates a separable implementation of our joint filter $\mathbf{f}$ based on the above domain transformations, where $n_p \in \mathbb{N}$ is the number of pixels in the p-th coordinate basis (i.e., the image size of the p-th dimension). Its one-dimensional L^1 Gaussian convolution on the non-uniform pixel coordinates (transformed domain) is also accurately performed by using our domain-splitting technique described in Section 4.

Algorithm 2: Joint Filtering via Domain Transform

Input : Pixels $\{\mathbf{x} \in \mathbb{R}^d\}$ with side sizes $\{n_p\} \in \mathbb{N}^d$, their integrand $\{\mathbf{h}(\mathbf{x})\}$ and guidance $\{\mathbf{g}(\mathbf{x})\}$ colors on $\mathbb{R}^c$, and scale σ and metric λ parameters.
Output: Joint filtered colors $\{\mathbf{f}(\mathbf{x})\}$ as $\mathbf{f}(\mathbf{x}, \mathbf{g}, \mathbf{h})$.

```
// (c+1)-th channel for the denominator of Eq. (2).
X(x) ← {h(x), 1} for all x;
/* Transformation (7) can be computed here: a
   tradeoff between speed and memory usage.      */
for i ← 1 to N_ite do
    Update σ_i by Eq. (8);
    for p ← 1 to d do
        for any 1D pixel array of X parallel to its p-th basis do
            t[·] ← DomainTransform(g, n_p, c, λ);
            X[·] ← GaussTransform(t, X, c+1, n_p, σ_i);
                // Algorithm 5.
        end for
    end for
end for
for q ← 1 to c do
    f_q(x) ← X_q(x)/X_{c+1}(x) for all x;
end for
return {f(x) := {f_q(x)}};
/* Domain transformation by Eq. (7):             */
Function DomainTransform({g[·] = {g_q[·]}}, n, c, λ):
t[1] ← 0;
for j ← 1 to n − 1 do
    Y ← Σ_{q=1}^{c} | g_q[j+1] − g_q[j] |^2 ;
    t[j+1] ← t[j] + sqrt(1 + λ^2 Y) ;
end for
return t[·];
```

4 Domain-splitting Gaussian Convolution

Using conventional fast methods such as the well-studied FFTs [7] and recursive filters [6, 1, 31] to perform Gaussian convolutions on a transformed domain is not trivial, because these methods have usually been designed for use with only uniform-image pixels. See [13] for an excellent review of fast Gaussian convolution methods. In contrast, a domain-splitting technique [34, 3] approximates a discrete analogue of a Gaussian convolution (known as a *Gauss transform* [14]) very accurately (without ringing artifacts) and quickly, even for a non-uniformly sampled variable of the Gaussian function. The key feature of this technique is decomposition of the integral domain of Gaussian convolution into subdomains centered at each representative point on the domain by using the L^1 norm for a metric instead of the popular L^2 norm. We explain here the domain-splitting technique adapted to our framework in order to accurately compute the Gaussian convolutions of the initial smoothed image $\mathbf{J}^0$

in Eq. (1) and the joint filter (defined in Eqs. (2) and (3)) on the transformed domain $T_{p,\lambda}(x_p, \mathbf{x}, \mathbf{g})$.

For a given pixel $\mathbf{x} = \{x_p\}$ on $\mathbb{R}^d$, consider a set of n points $\mathcal{P} : \{t_i\}$ on $\mathbb{R}$ representing the transformed pixels of $\mathbf{x}$ with respect to its p-th coordinate basis and corresponding color $\mathbf{h} = \{h_q(\mathbf{x})\} \in \mathbb{R}^c$ such that

$$t_i \equiv T_{p,\lambda}(i, \mathbf{x}, \mathbf{h}), \quad t_1 \le t_2 \le ... \le t_n,$$
$$h(t_i) \equiv h_q(\mathbf{u}_p(i, \mathbf{x})), \quad i = \{1, 2, ..., n\}$$

where $h(t_i) \in \mathbb{R}$ is a q-th channel color at $\mathbf{t}_i$, $n \in \mathbb{N}$ is the number of pixels, and the transformation $T_{p,\lambda}(\cdot)$ with its straight line $\mathbf{u}_p$ are defined in Eqs. (7) and (6), respectively. The L^1 Gauss transform $f(\cdot) \in \mathbb{R}$ with $h(\cdot)$ at $t_j \in \mathcal{P}$ is given by

$$f(t_j) = \sum_{i=1}^{n} G_\sigma(t_j - t_i) h(t_i) \tag{9}$$

where naively computing $f(t_j)$ for all $j = \{1, 2, ..., n\}$ requires quadratic computational complexity $O(n^2)$.

Basic Decomposition Concept: For a fixed point t_1, splitting the domain of the L^1 norm distance between t_i and t_j with respect to their order yields

$$|t_i - t_j| = \begin{cases} |t_i - t_1| - |t_j - t_1| & \text{if} \quad t_1 \le t_j \le t_i, \\ -|t_i - t_1| + |t_j - t_1| & \text{if} \quad t_1 \le t_i < t_j. \end{cases}$$

This process is illustrated conceptually in Fig. 7. Substituting the above equation into an L^1 Gaussian $G_\sigma(t_i - t_j)$ leads to

$$G_\sigma(t_i - t_j) = \begin{cases} \dfrac{G_\sigma(t_i - t_1)}{G_\sigma(t_j - t_1)} & \text{if} \quad t_1 \le t_j \le t_i, \\ \dfrac{G_\sigma(t_j - t_1)}{G_\sigma(t_i - t_1)} & \text{if} \quad t_1 \le t_i < t_j \end{cases}$$

where the dependency of the two indices i and j within the Gaussian is resolved such that a two-variable function can be replaced by a combination of two one-variable functions. The L^1 Gauss transform $f(\cdot)$ consequently becomes

$$f(t_j) = h(t_j) + G_\sigma(t_j - t_1)\xi(j-1) + \frac{\eta(j+1)}{G_\sigma(t_j - t_1)}, \tag{10}$$

$$\xi(j) = \sum_{i=1}^{j} \frac{h(t_i)}{G_\sigma(t_i - t_1)}, \quad \eta(j) = \sum_{i=j}^{n} G_\sigma(t_i - t_1) h(t_i)$$

where $\xi(0) \equiv 0 \equiv \eta(n+1)$. Note that Eq. (10) depends on only the index j, and $\xi(\cdot)$ and $\eta(\cdot)$ can be pre-computed (with linear computational time) for all $j = \{1, 2, ..., n\}$ at once before calculating $f(t_j)$. Linear computational complexity $O(3n)$ is hence required to calculate $f(t_j)$ for all indices j by the decomposition (10). However, some numerical instability makes it difficult to compute Eq. (10) accurately. For instance, $\frac{1}{G_\sigma(t_i - t_1)}$ may cause an overflow for very large values of $|t_i - t_1|$.

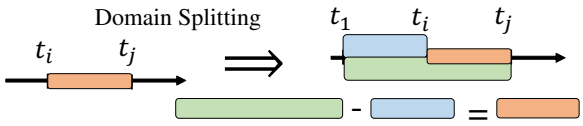

Fig. 7 A basic domain decomposition concept that enables measurement of the distance between t_i and t_j by subtraction of two distances via an anchor coordinate t_1.

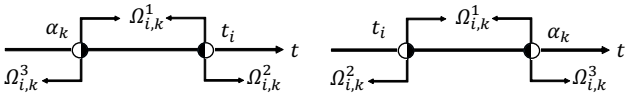

Fig. 8 Division of the domains $\Omega^1_{i,k}$, $\Omega^2_{i,k}$, and $\Omega^3_{i,k}$ via the poles $\{\alpha_k\}$.

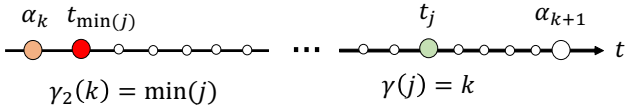

Fig. 9 Indexing functions $\gamma(\cdot)$ and $\gamma_2(\cdot)$.

Accurate Approximation: To avoid the above-mentioned numerical problem, consider a set of m representative poles $\{\alpha_k\}$ on $\mathbb{R}$ instead of using the fixed point t_1. Assuming that $\alpha_1 < \alpha_2 < ... < \alpha_m$, the domain splitting of $|t_i - t_j|$ around the pole α_k is given by

$$|t_i - t_j| = \begin{cases} |t_i - \alpha_k| - |t_j - \alpha_k| & \text{if} \quad t_j \in \Omega^1_{i,k}, \\ -|t_i - \alpha_k| + |t_j - \alpha_k| & \text{if} \quad t_j \in \Omega^2_{i,k}, \\ |t_i - \alpha_k| + |t_j - \alpha_k| & \text{if} \quad t_j \in \Omega^3_{i,k} \end{cases}$$

where the domains $\Omega^1_{i,k}$, $\Omega^2_{i,k}$, and $\Omega^3_{i,k}$ are defined by

$$\begin{aligned} \Omega^1_{i,k} &= \{z \in \mathcal{P} : \alpha_k \le z \le t_i, \text{ or } t_i \le z \le \alpha_k\}, \\ \Omega^2_{i,k} &= \{z \in \mathcal{P} : \alpha_k \le t_i < z, \text{ or } z < t_i \le \alpha_k\}, \\ \Omega^3_{i,k} &= \{z \in \mathcal{P} : z < \alpha_k \le t_i, \text{ or } t_i \le \alpha_k < z\}, \end{aligned}$$

as shown in Fig. 8. This domain splitting with the poles $\{\alpha_k\}$ thus makes the Gauss transform $f(\cdot)$ in Eq. (9) become

$$f(t_j) = h(t_j) + C_j + D_j + E_j, \tag{11}$$

$$\begin{aligned} C_j = \{G_{(\sigma,j,\gamma(j))} \sum_{i=\gamma_2(\gamma(j))}^{j-1} \frac{h(t_i)}{G_{(\sigma,i,\gamma(j))}}\} + \\ + \{\frac{1}{G_{(\sigma,j,\gamma(j))}} \sum_{i=j+1}^{\gamma_2(\gamma(j)+1)-1} G_{(\sigma,i,\gamma(j))} h(t_i)\}, \end{aligned} \tag{12}$$

$$D_j = \sum_{k=1}^{\gamma(j)-1} G_{(\sigma,j,k)} A_k, \quad E_j = \sum_{k=\gamma(j)+1}^{m} G_{(\sigma,j,k)} B_k,$$

$$A_k = \sum_{i=\gamma_2(k)}^{\gamma_2(k+1)-1} \frac{h(t_i)}{G_{(\sigma,i,k)}}, \quad B_k = \sum_{i=\gamma_2(k)}^{\gamma_2(k+1)-1} G_{(\sigma,i,k)} h(t_i) \tag{13}$$

where $G_{(\sigma,l,k)} \equiv G_\sigma(t_l - \alpha_k)$, $\gamma(j) = k$, and $\gamma_2(k) = \min(j)$ such that $\alpha_k \le t_j < \alpha_{k+1}$, as illustrated in Fig. 9.

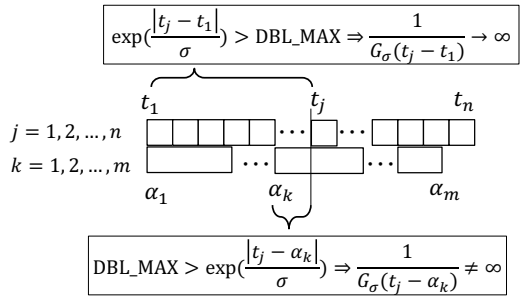

Fig. 10 Stable domain splitting by $\{\alpha_k\}$, where the indices j and k are chosen such that $|\alpha_{k+1}-\alpha_k| \geq |t_j - \alpha_{\gamma(j)}| \geq |t_{\gamma_2(\cdot)} - \alpha_{\gamma(\cdot)}|$ in Eqs. (12) to (13). It implies $\delta > \exp(\frac{|t_j-\alpha_k|}{\sigma})$ in this figure.

For a given upper bound of computational precision δ, the inequality $\exp(\frac{|\alpha_{k+1}-\alpha_k|}{\sigma}) < \delta$ should be held to avoid the above-mentioned numerical instability. Satisfying this condition leads to a stable choice of distances between poles by using the relationship $|\alpha_{k+1} - \alpha_k| = \varphi\sigma\log(\delta)$, where $\varphi \in (0,1)$ is a parameter. In our framework, we use $\varphi = 0.5$ and a double floating-point precision format for δ (i.e., 64bit *DBL_MAX* of the C++ programming language). Since the range of $\mathcal{P}$ is defined by $w = t_n - t_1 > 0$, the number of poles and their coordinates are automatically determined by

$$\{\alpha_k\} = t_1 + \{0, 1, ..., m-1\}\frac{w}{m}, \quad m = [\frac{w}{\varphi\sigma\log(\delta)}] \quad (14)$$

where $[\cdot]$ is the ceiling function and $m \ll n$ in general cases. Although t_1 is always equal to zero in our framework, shifting the first pole ($\alpha_1 = t_1$), as in Eq. (14), guarantees that Eq. (11) is numerically stable because of how the poles $\{\alpha_k\}$ are chosen (see Fig. 10).

If $|\alpha_k - t_j| > \sigma\log(\delta)$, then $G_\sigma(\alpha_k - t_j)$ becomes numerically zero where α_k is located far from t_j. Therefore, D_j and E_j are approximated by

$$D_j \approx G_{(\sigma,j,\gamma(j)-1)}A_{\gamma(j)-1}, \quad E_j \approx G_{(\sigma,j,\gamma(j)+1)}B_{\gamma(j)+1} \quad (15)$$

where $D_j \approx 0$ if $\gamma(j) < 2$, and $E_j \approx 0$ if $\gamma(j) > n-1$. The approximation error at t_j is given by

$$\sum_{k=1}^{\gamma(j)-2} G_\sigma(t_j - \alpha_k)A_k + \sum_{k=\gamma(j)+2}^{m} G_\sigma(t_j - \alpha_k)B_k.$$

The above approximation of $f(t_j)$ thus possesses accuracy equivalent to truncation of a sum within a $3\varphi\sigma\log(\delta)$ region. Such a truncation is very accurate[2] and equivalent to naive truncation with a radius of about 525σ, where $\varphi = 0.5$ and $\log(\text{DBL\_MAX}) \approx 700$. Also, this approximation algorithm requires only $O(4n + 2m + \max(n,m))$ operations ($O(1/[\sigma])$ with respect to σ).

[2] In statistics and its application to the empirical sciences, a 3σ radius is often used and considered sufficient to approximate normally distributed data (i.e., an L^2 Gaussian function). However, a Gaussian convolution with naive truncation within a 3σ radius produces some undesired ringing artifacts around sharp image edges [34].

Algorithm 3: Initialization of Domain Splitting

Input : n points $t[\cdot]$ on $\mathbb{R}_{\geq 0}$, and scale σ.
Output: The number of poles m, index set $\gamma[\cdot]$, and coefficients $C_1[\cdot]$, $C_2[\cdot]$, $D_f[\cdot]$, and $E_f[\cdot]$.

Function DSInit($t[\cdot], n, \sigma$):
/* Computing $\gamma(\cdot)$, m, and α_k by Eq. (14). */
$\varphi \leftarrow 0.5$; $\delta \leftarrow$ DBL_MAX; // Precision: $\varphi \in (0,1)$.
$w \leftarrow t[n] - t[1]$; // Range of $\mathcal{P}$.
$m \leftarrow \text{ceil}(w/(\varphi\sigma\log(\delta)))$; // Number of poles.
$\alpha[1] \leftarrow t[1]$; $k \leftarrow 1$; $\alpha_{dist} \leftarrow w/m$;
for $j \leftarrow 1$ **to** $m-1$ **do**
  $\alpha[j+1] \leftarrow \alpha[j] + \alpha_{dist}$;
end for
for $j \leftarrow 1$ **to** n **do**
  if $k < m$ **then**
    while $t[j] > \alpha[k+1]$ **do**
      if $k < m$ **then** $k \leftarrow k+1$;
      else Break;
    end while
  end if
  $\gamma[j] \leftarrow k$;
end for
/* Coefficients of C_j, D_j, and E_j in Eqs. (12), (15): $C_1[\cdot]$, $C_2[\cdot]$, $D_f[\cdot]$, and $E_f[\cdot]$. */
for $j \leftarrow 1$ **to** n **do**
  $D_f[j] \leftarrow E_f[j] \leftarrow 0$;
  $C_1[j] \leftarrow \exp(-(t[j] - \alpha[\gamma[j]])/\sigma)$; $C_2[j] \leftarrow 1/C_1[j]$;
  if $\gamma[j] > 1$ **then** $D_f[j] \leftarrow \exp(-\frac{t[j]-\alpha[\gamma[j]-1]}{\sigma})$;
  if $\gamma[j] < m$ **then** $E_f[j] \leftarrow \exp(-\frac{\alpha[\gamma[j]+1]-t[j]}{\sigma})$;
end for
return $\{m, \gamma[\cdot], C_1[\cdot], C_2[\cdot], D_f[\cdot], E_f[\cdot]\}$;

Algorithm 4: Integration of Domain Splitting

Input : n colors $h[\cdot]$, numbers of poles m, index set $\gamma[\cdot]$ and coefficients: $C_1[\cdot]$, $C_2[\cdot]$, $D_f[\cdot]$, and $E_f[\cdot]$.
Output: Gauss transform $f[\cdot]$ of $h[\cdot]$.

Function DSSum($t[\cdot], n, m, \gamma[\cdot], C_1[\cdot], C_2[\cdot], D_f[\cdot], E_f[\cdot]$):
/* Calculating A_k and B_k in Eq. (13), which are also used to obtain C_j. */
for $k \leftarrow 1$ **to** m **do**
  $A[k] \leftarrow B[k] \leftarrow 0$;
end for
for $j \leftarrow 1$ **to** n **do**
  $A[\gamma[j]] \leftarrow A[\gamma[j]] + C_2[j]h[j]$; $S_2[j] \leftarrow A[\gamma[j]]$;
end for
for $j \leftarrow n$ **to** 1 **do**
  $B[\gamma[j]] \leftarrow B[\gamma[j]] + C_1[j]h[j]$; $S_1[j] \leftarrow B[\gamma[j]]$;
end for
/* Integrating D_j, E_j, and C_j in Eq. (11). */
$f[1] \leftarrow h[1] + S_1[2]C_2[1]$;
if $\gamma[1] < m$ **then** $f[1] \leftarrow f[1] + E_f[1]B[2]$;
for $j \leftarrow 2$ **to** $n-1$ **do**
  $X \leftarrow Y \leftarrow 0$; $Z \leftarrow h[j]$;
  if $\gamma[j] > 1$ **then** $X \leftarrow D_f[j]A[\gamma[j]-1]$;
  if $\gamma[j] < m$ **then** $Y \leftarrow E_f[j]B[\gamma[j]+1]$;
  if $\gamma[j] = \gamma[j+1]$ **then** $Z \leftarrow Z + S_1[j+1]C_2[j]$;
  if $\gamma[j] = \gamma[j-1]$ **then** $Z \leftarrow Z + S_2[j-1]C_1[j]$;
  $f[j] \leftarrow X + Y + Z$;
end for
$f[n] \leftarrow h[n] + S_2[n-1]C_1[n]$;
if $\gamma[n] > 0$ **then** $f[n] \leftarrow f[n] + D_f[n]A[\gamma[n]-1]$;
return $f[\cdot]$;

	Complexity	Mult.	Add.	$\exp(\cdot)$
DSInit	$O(n+m+\max(n,m))$	4	3	3
DSSum	$O(3n+m)$	6	6	0

Table 1 Complexities and costs: numbers of multiplications (Mult.), additions (Add.), and exponential functions ($\exp(\cdot)$) per pixel.

Algorithm 5: Guided L^1 Gauss Transform

Input : n points $t[j]$ on $\mathbb{R}_{\geq 0}$ and their c-channel colors $\mathbf{h}[j] = \{h_q[j]\}$, and scale σ.
Output: Transformed $\mathbf{h}[\cdot]$ with non-uniform coordinates $t[\cdot]$.

Function GaussTransform($t[\cdot], \mathbf{h}[\cdot], c, n, \sigma$):
  $Coef \leftarrow$DSInit(t, n, σ); `// Algorithm 3.`
  for $q \leftarrow 1$ **to** c **do**
    $h_q[\cdot] \leftarrow$DSSum($h_q, n, Coef$); `// Algorithm 4.`
  end for
  return $\mathbf{h}[\cdot]$;

Efficient Implementation: The above domain-splitting technique is efficiently implemented in our framework by using the following initialization (*DSInit*) and integration (*DSSum*) procedures described in the pseudocodes of Algorithms 3 and 4, respectively. DSInit($\cdot$) computes the poles $\{\alpha\}$ in Eq. (14) and some of the coefficients C_j, D_j, and E_j in Eqs. (12) and (15). DSSum($\cdot$) then calculates the L^1 Gauss transform $\{f(t_j)\}$ in Eq. (11) with respect to the given colors $\{h(t_j)\}$ by using the coefficients determined by DSInit($\cdot$) : $Coef$. Since $Coef$ depends on only the pixel (transformed) coordinates $\mathcal{P} : \{t_j\}$ and scale σ, it can be reused for different color channels, as in Algorithm 5 (*GaussTransform* in the joint filter: Algorithm 2), as well as for the uniform pixel coordinates, as in Algorithm 6 (*GaussianFilter* in the scale-aware filter: Algorithm 1, i.e., $\mathbf{J}^0$ in Eq. (1)). Note that the terms $\sum_i h(t_i)/G_\sigma(t_i - \alpha_\cdot)$ and $\sum_i G_\sigma(t_i - \alpha_\cdot)h(t_i)$ appear in the equations for A_k, B_k, and also C_j. The results of computing A_k and B_k are therefore reused to obtain C_j in DSSum($\cdot$) (see Algorithm 4). We also employed a fast library [22] for the exponential function $\exp(\cdot)$ in DSInit($\cdot$). Ignoring substitutions and indexing, the complexities and costs of DSInit($\cdot$) and DSSum($\cdot$) are listed in Table 1.

5 Numerical Experiments

All numerical experiments in this paper were performed on an i9-10980XE CPU (3.0 GHz, 36 core, and no parallelization was used) PC with 128 GB RAM and a 64-bit OS with a GNU C++ 9.3 compiler.

We first examined the accuracy and computational speed of our domain-splitting approximation (Our: uniform and Our NU: non-uniform pixels) described in Section 4 (Algorithms 6 and 5) by numerically comparing their accuracy and computational speed with those of the popular box and recursive filters such as the Box (moving average) [26], EBox (extended box) [15], SII (stacked image integral) [4, 9], Deriche [6], VYV [31], and AM [1] filters. Their implementations and boundary conditions were based on the open

Algorithm 6: Uniform L^1 Gaussian Image Filtering

Input : Pixels $\{\mathbf{x} := \{x_p\} \in \mathbb{R}^d\}$ with side sizes $\{n_p\} \in \mathbb{N}^d$, their colors $\{\mathbf{I}(\mathbf{x}) \in \mathbb{R}^c\}$, and scale σ.
Output: Filtered colors $\{Z(\mathbf{x})\}$.

Function GaussianFilter($\{\mathbf{x}\}, \{\mathbf{I}(\mathbf{x})\}, \sigma$):
  for $p \leftarrow 1$ **to** d **do**
    $X[\cdot] \leftarrow \{1, 2, ..., n_p\}$; `// Pixel coordinates.`
    $Coef[p] \leftarrow$DSInit(X, n_p, σ); `// Algorithm 3.`
    $Y_p[\cdot] \leftarrow$DSSum($\{1\}, n_p, Coef[p]$); `// Algorithm 4.`
  end for
  $Z(\mathbf{x}) \leftarrow \mathbf{I}(\mathbf{x})$ for all $\mathbf{x}$;
  for $p \leftarrow 1$ **to** d **do**
    for *any 1D pixel array of Z parallel to its p-th basis* **do**
      $Z[\cdot] \leftarrow$DSSum($Z, n_p, Coef[p]$); `// Algorithm 4.`
    end for
  end for
  $Z(\mathbf{x}) \leftarrow Z(\mathbf{x}) / \prod_{p=1}^{d} Y_p[x_p]$ for all $\mathbf{x}$;
  return $\{Z(\mathbf{x})\}$;

library of [13] with 10^{-15} tolerance (4th order recursion). Table 2 shows the peak signal-to-noise ratio (PSNR) [24, 34] and the maximum error $E_{\max}$ [34] of one-dimensional Gaussian filtering ($d = 1$ for $\mathbf{J}^0$ in Eq. (1) and convolutions in the joint filter) for relatively small σ (0.05% to 1% of image size $w = n$), where the exact Gauss transform (Eq. (9)) and FIR (Finite Impulse Response) [13] were employed for comparison with their approximations. Our approximations were slightly slower than those with the conventional methods but significantly more accurate (about 10^{10} times).

Figure 11 demonstrates a typical error profile of Table 2 parameter settings, where some ripple-shaped errors, which cause undesired artifacts and phantom edges, are observed for all conventional methods. At first glance, the VYV and Deriche filters look preferable for use in computer graphics applications because the magnitudes of their errors are small, as shown in Table 2 and Fig. 11. However, applying these methods stably and accurately for variable parameters is not trivial. During our experiments, we easily encountered cases of artifacts that could not be ignored or that resulted in complete failure. Figure 11 (top-right) shows a boundary artifact caused by use of the Deriche filter for large σ (10% of the image size, which is not irrelevantly large). This artifact may have been the result of one of the causes discussed in [30, 13, 5]. Moreover, the errors of these methods, including the AM and EBox filters, behave in a nonlinear manner with respect to not only σ but also the image size w (number of pixels n as well) because their formulations (e.g., coefficient optimization) rely on these parameters. Despite the fact that both σ and n were linearly scaled from the case in Fig. 11, the VYV and Deriche filters generated the results with the large errors, as shown in Fig. 12. In contrast to these conventional methods, our theoretical formulations guarantee the high stability and accuracy of our approximations, which were numerically confirmed by our experiments. Since the

Method	PSNR	$E_{\max}$	Time: $n \in 10^{\{4,5,6\}}$		
EBox [15]	55.79	1.03e-02	0.014	1.77	18
SII [4,9]	38.88	5.509e-02	0.092	0.86	9.4
Box [26]	31.34	1.535e-01	0.039	0.32	3.4
Deriche [6]	96.37	1.184e-04	0.165	1.62	18
VYV [31]	74.8	1.294e-03	0.167	1.46	14.9
AM [1]	53.6	1.438e-02	0.495	3.8	38.4
Our [34,3]	291.6	2.842e-14	0.325	3.58	43.2
Our NU [34,3]	298.7	3.12e-14	0.43	3.57	43

Table 2 Timing and accuracy comparisons of $\mathbf{J}^0$ where $d = 1$, $n = 10^4$, $h(\cdot) \in [0,1]$, averaged PSNR of 20 $\sigma \in \{5, 10, ..., 100\}$, and 100 datasets randomly generated for each σ were employed. Time shows averaged computational time (milliseconds) for $n \in 10^{\{4,5,6\}}$.

		RG		SiR		AG	
Name	s	box	our	box	our	box	our
Balloon	4	2.4	4	4.1	4.2	10.2	9.9
	20	11	19	20	19.5	50.2	48
Face	4	0.32	0.57	0.37	0.62	1.07	1.41
	20	1.44	2.7	1.75	2.92	5.34	6.9
Glass	4	3.5	5.35	3.3	5.36	9.5	12.8
	20	15.5	25.4	15.4	25.7	46	63
Snack	4	0.93	1.55	1.4	1.6	3.2	3.9
	20	4.2	7.4	6.6	7.6	15.8	19
Ham	4	0.74	0.97	0.9	1.06	2.4	2.51
	20	3	4.7	4.4	5.01	11.1	12.5

Table 3 Timings (seconds) by box and our framework, where s represents the iteration number in Eq. (1). The image sizes and figure numbers (ϕ and σ are described in the captions) are listed in Fig. 13.

RG-box	RG-our	SiR-box	SiR-our	AG-box	AG-our
5.41/s	3.072/s	3.902/s	3.014/s	1.597/s	1.244/s

Table 4 Speed comparison of box [26] and our scale-aware filters (megapixels per second), where s is the iteration number in Eq. (1).

quality of both $\mathbf{J}^0$ and the convolutions used in the joint filter are important, the approximation described in Section 4 is suitable for scale-aware filters.

We next evaluated the quality and computational speed of our scale-aware filters. The input images are listed in Fig. 13, and the timings are shown in Table 3. Figures 3, 4, and 18 demonstrate our scale-aware filtering results with varying scales (σ). The scale-space and salient edge features are clearly apparent. Our filters did not produce undesired artifacts generated by box-based convolutions (moving average [26], where $\sqrt{3}\sigma$ radius was employed to obtain the visually similar results), as illustrated in Figs. 1, 15, and 16. Figure 14 shows timing comparisons for various numbers of iterations s and images sizes, and it is summarized in Table 4.

Our filters achieved linear computational speeds (slightly slower than the box kernel convolutions[3]), and their convergence properties were similar to those of the RG, SiR, and AG filters, as shown in Fig. 5. See Fig. 17 for the filtering effects of iterations $s \in \{4, 20\}$ recommended by [36].

[3] Note that the timing results in Table 2 was optimized for one-dimensional array. In contrast, the results shown in Fig. 14 and Tables 3 and 4 were based on two-dimensional arrays which are not optimized for random memory access in the separable implementation.

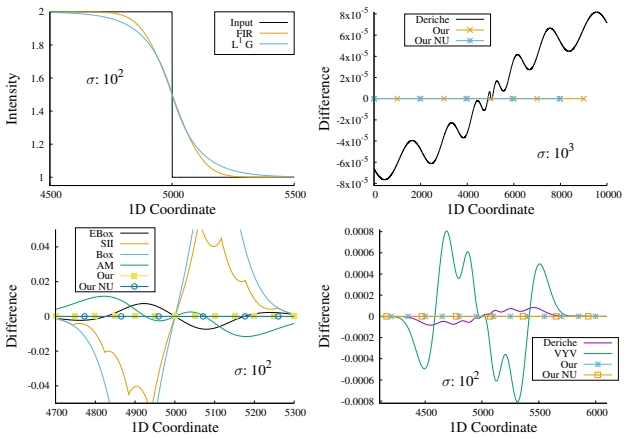

Fig. 11 1-D Gaussian filtering $\mathbf{J}^0$ comparisons for $w = n = 10^4$ pixels with $\sigma \in \{10^2, 10^3\}$, where differences between the approximations and their correct results are shown. Top-left: input and the correct results with $\sigma = 10^2$ based on FIR and exact L^1 Gauss transform. Top-right: A boundary artifact of the Deriche [6] with $\sigma = 10^3$.

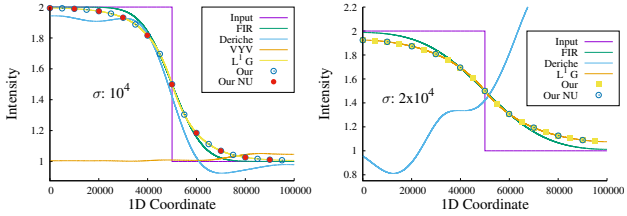

Fig. 12 1-D Gaussian filtering $\mathbf{J}^0$ comparisons with artifacts where $n = 10^5$ and σ: 10^4 (left) and 2×10^4 (right). The PSNRs of the Deriche, VYV, Our, and Our NU in the left (right) image are equal to 24.7 (6.6), 9.2 (10.3), 280 (280), and 278 (280), respectively.

Balloon [27]		Face	Glass	Snack	Ham
w	2121	610	1654	1368	1032
h	1414	752	2181	912	774
Fig.	3	15	16	17	18

Fig. 13 Input images with their numbers of pixels (w: width and h: height) and numbers of figures employed.

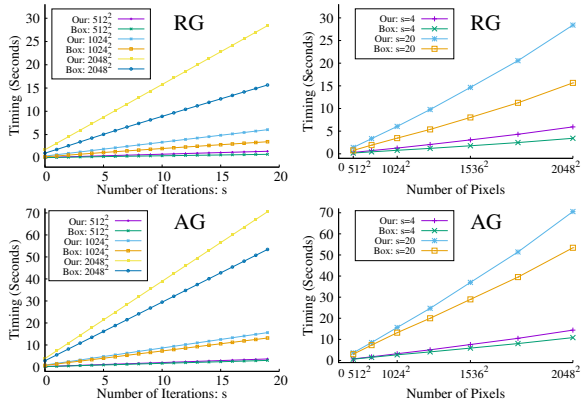

Fig. 14 Timings versus iterations (left: three different sizes) and versus numbers of pixels (right), where scaled Lena images representing an average of 10 iterations for each of 10 different scales ($\sigma \in \{5, 10, 15..., 50\}$) were employed. Note that the SiR graphs are omitted because their profiles are similar to the RG graphs visually.

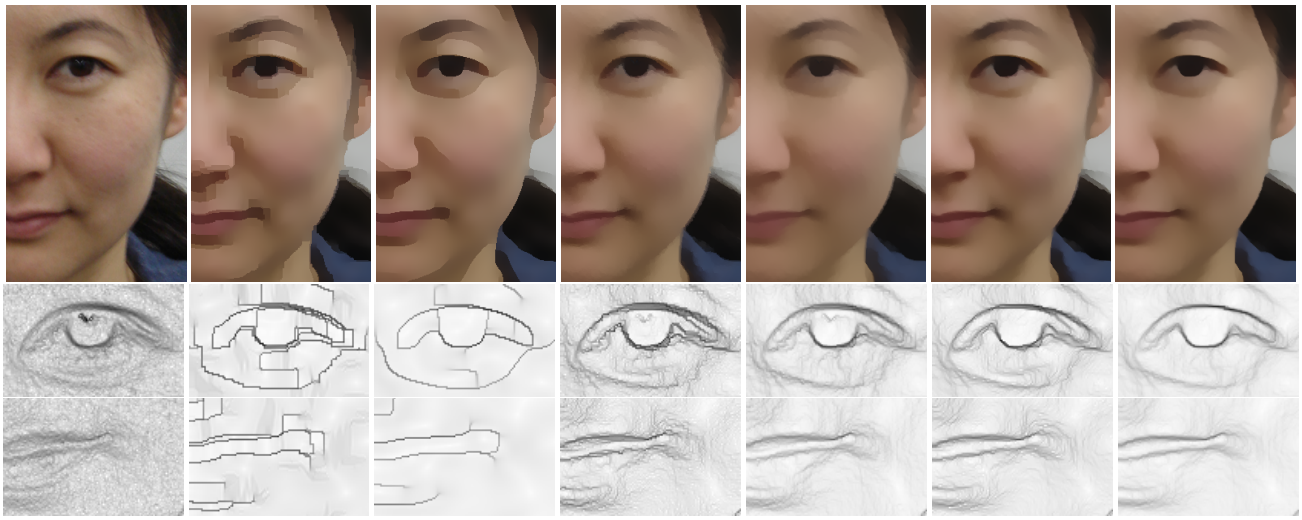

Fig. 15 Quality comparison of image of face for the RG, SiR, and AG filters with the box kernel ($\sqrt{3}\sigma$ radius was used) and our framework (L^1 Gaussian), where $s = 20$, $\phi = 1$, $\sigma = 6$ (RG and AG) and $\phi = 0.5$, $\sigma = 3$ (SiR). Bottom images correspond to the square root of the magnitude of the gradient ($||\nabla \mathbf{J}^{20}||^{1/2}$) of the eye and mouth parts of the upper images, where some artifacts appeared in the box-based results.

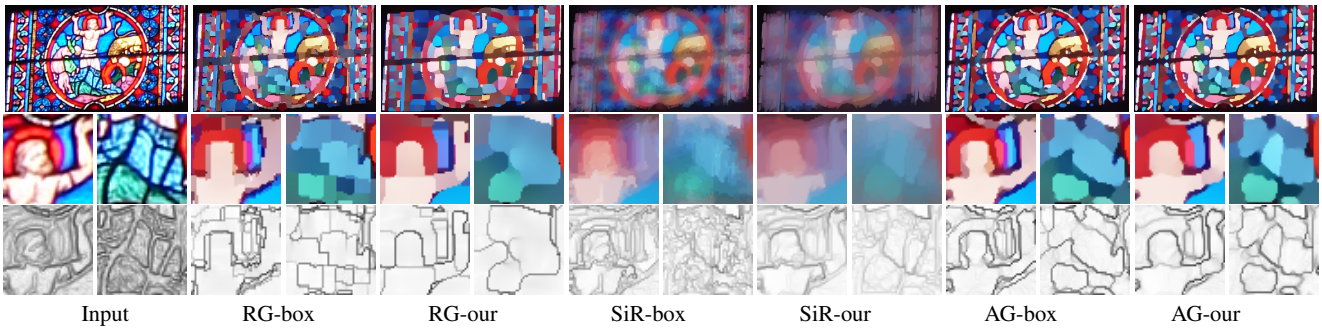

Fig. 16 Quality comparison of image of glass for the RG, SiR, and AG filters with the box kernel ($\sqrt{3}\sigma$ radius was used) and our framework (L^1 Gaussian), where $\phi = 1.5$, $\sigma = 8$, and $s = 20$. Bottom images correspond to the same square root of the magnitude of the gradient ($||\nabla \mathbf{J}^{20}||^{1/2}$) as the bottom images in Fig. 15.

Fig. 17 Iteration effects of image of snack via our framework with $\phi = 1.5$, $\sigma = 16$, and $s \in \{4, 20\}$.

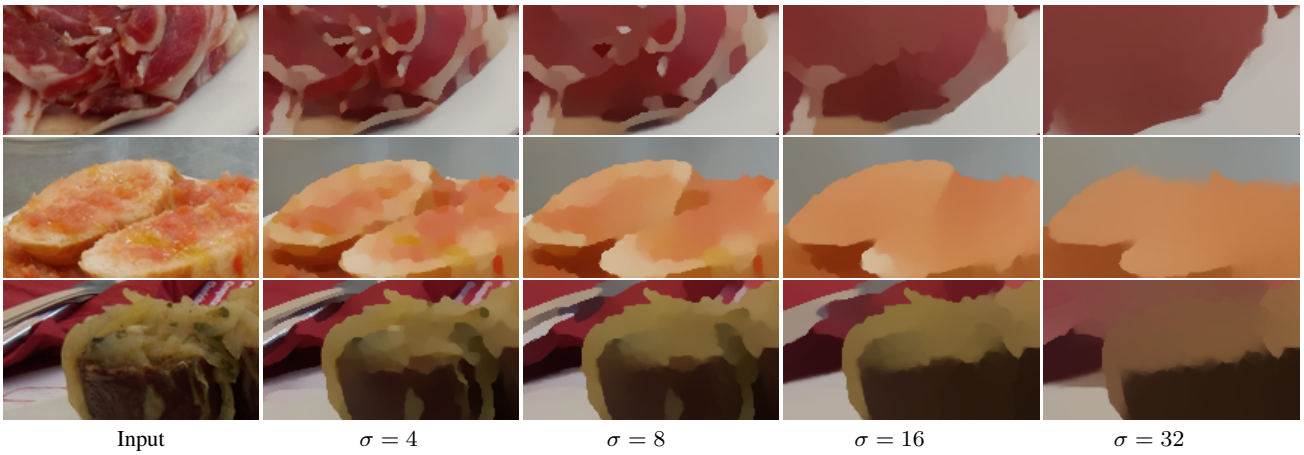

Fig. 18 Various scales of image of ham via our framework with the AG filter where $\phi = 0.5$, $s = 20$, and $\sigma \in \{4, 8, 16, 32\}$.

6 Conclusion

We have proposed a fast and accurate computational framework for scale-aware image filters. Our new framework is based on accurately approximating L^1 Gaussian convolution with respect to a transformed pixel domain representing geodesic distance on a guidance image manifold in order to recover salient edges in a manner faithful to scale-space theory while removing small image structures. Our framework achieved linear computational complexity with high-quality filtering results. We compared our framework numerically in terms of speed, precision, and visual quality with popular conventional methods.

Since our framework is robustly applicable to HDR images with a wide range of scale σ, applications to computational photography, engineering, and science are promising future work. Limitations of our framework are related to domain transformations and scale-aware filters, such as slow convergence of elongated image regions (Fig. 17) and reduced intensity (SiR, Fig. 4). Combining our framework with a guided filter [17] may reduce such artifacts. Future work will also include numerical comparisons with the non-uniform recursive method [12] (an extension of Deriche [6]).

References

1. Alvarez, L., Mazorra, L.: Signal and image restoration using shock filters and anisotropic diffusion. SIAM J. Numer. Anal. **31**(2), 590–605 (1994). DOI 10.1137/0731032
2. Astola, J., Haavisto, P., Neuvo, Y.: Vector median filters. Proc. of the IEEE **78**(4), 678–689 (1990). DOI 10.1109/5.54807
3. Bashkirova, D., Yoshizawa, S., Latypov, R., Yokota, H.: Fast L^1 Gauss transforms for edge-aware image filtering. Proc. ISP of RAS **29**(4), 55–72 (2017). DOI 10.15514/ISPRAS-2017-29(4)-4
4. Bhatia, A., Snyder, W., Bilbro, G.: Stacked integral image. In: IEEE ICRA, pp. 1530–1535 (2010). DOI 10.1109/ROBOT.2010.5509400
5. Cuomo, S., Farina, R., Galletti, A., Marcellino, L.: A k-iterated scheme for the first-order Gaussian recursive filter with boundary conditions. In: FedCSIS, pp. 641–647. IEEE (2015). DOI 10.15439/2015F286
6. Deriche, R.: Recursively implementing the Gaussian and its derivatives. Tech. Rep. RR-1893, INRIA (1993)
7. Duhamel, P., Vetterli, M.: Fast Fourier transforms: A tutorial review and a state of the art. Signal Process. **19**(4), 259–299 (1990). DOI 10.1016/0165-1684(90)90158-U
8. Eisemann, E., Durand, F.: Flash photography enhancement via intrinsic relighting. ACM TOG **23**(3), 673–678 (2004). DOI 10.1145/1015706.1015778
9. Elboher, E., Werman, M.: Efficient and accurate Gaussian image filtering using running sums. In: ISDA, pp. 897–902 (2012). DOI 10.1109/ISDA.2012.6416657
10. Elder, J.: Are edges incomplete? IJCV **34**(2-3), 97–122 (1999). DOI 10.1023/A:1008183703117
11. Gastal, E., Oliveira, M.: Domain transform for edge-aware image and video processing. ACM TOG **30**(4), 69:1–69:12 (2011). DOI 10.1145/2010324.1964964
12. Gastal, E., Oliveira, M.: High-order recursive filtering of non-uniformly sampled signals for image and video processing. CGF **34**(2), 81–93 (2015). DOI 10.1111/cgf.12543
13. Getreuer, P.: A survey of Gaussian convolution algorithms. Image Process. Line **3**, 276–300 (2013). DOI 10.5201/ipol.2013.87
14. Greengard, L., Strain, J.: The fast Gauss transform. SIAM J. Sci. Stat. Comput. **12**(1), 79–94 (1991). DOI 10.1137/0912004
15. Gwosdek, P., Grewenig, S., Bruhn, A., Weickert, J.: Theoretical foundations of Gaussian convolution by extended box filtering. In: SSVM, pp. 447–458 (2011). DOI 10.1007/978-3-642-24785-9_38
16. Ham, B., Cho, M., Ponce, J.: Robust guided image filtering using nonconvex potentials. IEEE TPAMI **40**(1), 291–207 (2018). DOI 10.1109/TPAMI.2017.2669034
17. He, K., Sun, J., Tang, X.: Guided image filtering. IEEE TPAMI **56**(6), 1397 – 1409 (2013). DOI 10.1109/TPAMI.2012.213
18. Iijima, T.: Basic theory of pattern observation. In: Papers of Technical Group on Automata and Automatic Control. IEICE (1959). (Japanese)
19. Kniefacz, P., Kropatsch, W.: Smooth and iteratively restore: A simple and fast edge-preserving smoothing model. ArXiv/CoRR: 1505.06702 (2015)
20. Koenderink, J.: The structure of images. Biol. Cybern. **50**, 363370 (1984). DOI 10.1007/BF00336961
21. Lindeberg, T.: On the axiomatic foundations of linear scale-space. In: Gaussian Scale-Space Theory, pp. 75–97. Springer (1997). DOI 10.1007/978-94-015-8802-7_6
22. Mitsunari, S.: Fast approximate function of exponential function exp and log (2012). Library: github.com/herumi/fmath
23. Osher, S., Sethian, J.: Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton- Jacobi formulations. J. Comput. Phys. **79**(1), 1249 (1988). DOI 10.1016/0021-9991(88)90002-2
24. Paris, S., Durand, F.: A fast approximation of the bilateral filter using a signal processing approach. IJCV **81**(1), 24–52 (2009). DOI 10.1007/s11263-007-0110-8
25. Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., Toyama, K.: Digital photography with flash and no-flash image pairs. ACM TOG **23**(3), 664–672 (2004). DOI 10.1145/1015706.1015777
26. Sochen, N., Kimmel, R., Malladi, R.: A general framework for low level vision. IEEE TIP **7**(3), 310–318 (1998). DOI 10.1109/83.661181
27. Song, B.L.: Cappadocia balloon inflating. In: CC BY-SA 2.0 (2010). URL www.flickr.com/people/blieusong
28. Struik, D.: Lectures on Classical Differential Geometry. Dover Publications (1988). 2nd Edition
29. Toet, A.: Alternating guided image filtering. PeerJ Comput. Sci. **2**, e72 (2016). DOI 10.7717/peerj-cs.72
30. Triggs, B., Sdika, M.: Boundary conditions for young-van vliet recursive filtering. IEEE TSP **54**(6), 2365–2367 (2006). DOI 10.1109/TSP.2006.871980
31. van Vliet, L., Young, I., Verbeek, P.: Recursive Gaussian derivative filters. In: ICPR, vol. 1, pp. 509–514 (1998). DOI 10.1109/ICPR.1998.711192
32. Wang, P.S., Fu, X.M., Liu, Y., Tong, X., Liu, S.L., Guo, B.: Rolling guidance normal filter for geometric processing. ACM TOG **34**(6), 173:1–173:9 (2015). DOI 10.1145/2816795.2818068
33. Weickert, J., Ishikawa, S., Imiya, A.: Linear scale-space has first been proposed in Japan. J. Math. Imaging Vis. **10**, 237–252 (1999). DOI 10.1023/A:1008344623873
34. Yoshizawa, S., Yokota, H.: Fast L^1 Gaussian convolution via domain splitting. In: IEEE ICIP, pp. 2908–2912. IEEE-SP (2014). DOI 10.1109/ICIP.2014.7025588
35. Zhang, J., Deng, B., Hong, Y., Peng, Y., Qin, W., Liu, L.: Static/dynamic filtering for mesh geometry. IEEE TVCG **25**(4), 1774–1787 (2019). DOI 10.1109/TVCG.2018.2816926
36. Zhang, Q., Shen, X., Xu, L., Jia, J.: Rolling guidance filter. In: ECCV, LNCS, pp. 815–830. Springer (2014). DOI 10.1007/978-3-319-10578-9_53