Fast Guided Median Filter

Kazu Mishiba, *Member, IEEE*

***Abstract*— Faster computation of a weighted median (WM) filter is impeded by the construction of a weighted histogram for every local window of data. Since the calculated weights vary for each local window, it is difficult, using a sliding window approach, to construct the weighted histogram efficiently. In this paper, we propose a novel WM filter that overcomes the difficulty of histogram construction. Our proposed method achieves real-time processing for higher resolution images and can be applied to multidimensional, multichannel, and high precision data. The weight kernel used in our WM filter is the pointwise guided filter, which is derived from the guided filter. The use of kernels based on the guided filter avoids gradient reversal artifacts and shows a higher denoising performance than the Gaussian kernel based on the color/intensity distance. The core idea of the proposed method is a formulation that allows the use of histogram updates with a sliding window approach to find the weighted median. For high precision data we propose an algorithm based on a linked list that can reduce the memory requirements of storing histograms and the computational cost of updating them. We present implementations of the proposed method that are suitable for both CPU and GPU. Experimental results show that the proposed method indeed realizes faster computation than conventional WM filters and is capable of filtering multidimensional, multichannel, and high precision data. This is an approach which is difficult to achieve with conventional methods.**

***Index Terms*— Weighted median filter, guided filter.**

I. Introduction

SMOOTHING filters are one of the most basic tools in image processing. Their wide range of applications includes image processing for visible light, medical image analysis [1], [2] and seismic data processing [3], [4].

Edge-preserving smoothing is also an important technique for better smoothing. Edge-preserving smoothing filters can be divided into two groups: Filters in the first group calculate their coefficients from the input image. An example is the bilateral filter [5], which calculates a weighted average using filter coefficients derived from a Gaussian distribution of spatial and color/intensity distances. Filters in the second group calculate their coefficients from the guide image. Joint filters are often used when the input image has unreliable edge information, and a guide image is available that has reliable edge information. Examples of the application of joint filters include flash/no-flash denoising [6], image upsampling [7], depth map enhancement [8], and cost volume aggregation in stereo matching [9], [10], [11].

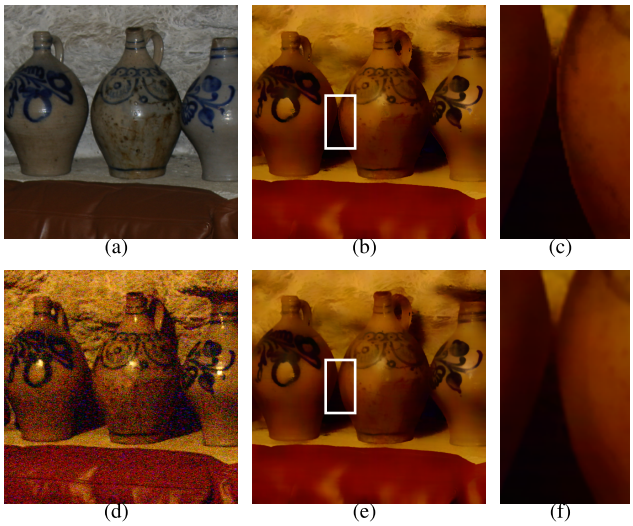

Fig. 1. Flash/no-flash denoising with WM filters. (a) Guide image. (b) Filtered image with the FWM-$\mathcal{O}(r)$ filter, whose weights are based on the Gaussian kernel. (c) Zoomed-in result of (b). (d) Input image. (e) Filtered image with the GPU-$\mathcal{O}(r)$ filter, whose weights are based on the guided filter kernel. (f) Zoomed-in result of (e).

The results of joint filters depend on the filter kernel that is used as well as the guide image. Methods that use the Gaussian kernel based on color/intensity distance, like the bilateral filter, have known problems with gradient reversal artifacts on the edges [12]. On the other hand, the guided filter kernel [12] allows for fast operation while avoiding gradient reversal artifacts.

Joint filtering can be applied to weighted median (WM) filters as well as weighted average filters. WM filters have edge-preserving properties and robustness against outliers. Due to these properties, joint WM filters are used in various image processing applications like optical flow estimation [13], disparity refinement [14], [15], and rain removal [16]. The use of the Gaussian kernel based on color/intensity distance can cause edge reversal artifacts in WM filters. Using a guided filter kernel can avoid this (Fig. 1).

The heavy computational cost of the WM filter is an obstacle to applications that require real-time performance. A fast computation algorithm for WM filters is needed. A possible approach to accelerate the computation is to adopt acceleration approaches used in the unweighted median (UWM) filter. The core idea of acceleration in the UWM filter is the sliding window, which takes advantage of overlapping filter windows by using neighboring pixels to accelerate histogram construction.

Manuscript received 8 June 2022; revised 27 November 2022; accepted 19 December 2022. Date of publication 5 January 2023; date of current version 10 January 2023. This work was supported by JSPS KAKENHI under Grant 20K11887. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Marta Mrak.

The author is with the Department of Electrical and Electronic Engineering, Tottori University, Tottori 680-8550, Japan (e-mail: mishiba@tottori-u.ac.jp).

Digital Object Identifier 10.1109/TIP.2022.3232916

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

As described in [17], it is almost impossible to apply methods that accelerate UWM filters to WM filters. The WM filter requires the construction of a weighted histogram. Since the weights spatially vary for each local window, the sliding window approach cannot be applied directly to construct the weighted histogram. Several acceleration methods have been proposed for WM filters using ideas different from those of UWM filters. Ma et al. [14] proposed a constant time WM filter: An input image is projected onto a 3D space, and then an edge-preserving filter is applied to each 2D slice. The advantage of this method is that any filter kernel can be used, including bilinear and guided filter kernels. Since an edge-preserving filter is applied to all 2D slices, the computational cost is not low enough to realize real-time processing. Zhang et al. [17] proposed an innovative acceleration method for WM filters. They use a joint histogram with two dimensions: one of the dimensions is pixel value and the other is feature. Faster computation is achieved by combining the joint histogram with a median tracking technique and a data structure that provides instant access to the data. This method achieves real-time processing for relatively small grayscale images with small window sizes. Filter kernels that can be used in this method are limited to those calculated from the color/intensity difference between two pixels. This includes the Gaussian kernel based on color/intensity distance. This means that this method cannot use a guided filter kernel, which calculates weights from multiple pixels. A fast WM filter using a guided filter kernel is a challenging task that has not yet been achieved.

In this paper, we propose a fast guided median filter weighted by a pointwise guided filter kernel. Like the guided filter kernel proposed in [12], the pointwise guided filter kernel can prevent gradient reversal artifacts. The core idea of the proposed method is a formulation that allows for the use of histogram updates with a sliding window approach to find the weighted median. Compared to the conventional fast WM filters, the proposed method is not only faster but also applicable to multidimensional, multichannel, and high precision data. Until recently, conventional methods were used to process various types of data. Examples include 8-bit color image processing; multidimensional data such as video and light field images; multichannel data such as multispectral images; and high precision data such as medical images, HDR images, and depth sensor data. The conventional methods have been insufficient for this data due to their heavy computational cost and enormous memory requirements. This has lead to a need to reduce the data size, e.g., by using downsampling. However, this leads to degraded accuracy in execution. Despite the situation, there has been little discussion about using WM filtering for multidimensional, multichannel, or high precision data.

The main contributions of this research are:

1) We propose an accelerated computation algorithm for the WM filter whose kernel is based on the guided filter. Faster computation is achieved by formulating a WM filter to which a sliding window method can be applied.
2) The proposed filter can be applied to multidimensional, multichannel, and high precision data.

The remainder of this paper is organized so that Section II reviews the related work on faster computation for UWM and WM filters. We define the weighted median used in this paper in Section III. The proposed WM filter is presented in Section IV, and its extensions to multidimensional, multichannel, and high precision data are presented in Section V. The implementation details of our method are shown in Section VI. Section VII shows the experimental results over high resolution, multidimensional, multichannel, and high precision data. Finally, we conclude our paper in Section VIII.

II. A Review of Related Work

This section focuses on the study of UWM and WM filters in achieving faster computation in image processing.

A. Median Filter

The median filter, which we also call the UWM filter, replaces the pixel value with the median value in a window centered at each pixel. There are two categories of methods for calculating median values. The first category is histogram-based methods, and the second category is sort-based methods.

The histogram-based methods first construct a histogram of the pixels in the window and then search for their median. For faster processing, faster histogram construction and faster median searches have been studied. To accelerate the construction of histograms, many methods use a sliding window approach that exploits the redundancy of the overlapping regions of the windows of adjacent pixels. The sliding window approach has two categories: The first is called a $\mathcal{O}(r)$ sliding window approach and the latter a $\mathcal{O}(1)$ sliding window approach. This is because the computational complexities with respect to window radius r are $\mathcal{O}(r)$ and $\mathcal{O}(1)$, respectively. Huang et al. [18] proposed a UWM filter that uses the $\mathcal{O}(r)$ sliding window approach to construct a histogram efficiently. The histogram of a horizontal sliding window can be updated efficiently by removing the elements of an excluded column resulting from the sliding window and then adding the elements of a new column included also, from the sliding window. To accelerate the median search, Huang et al. also used a technique referred to as median tracking [17]. Median tracking records changes in the number of pixels below the median value while the histogram is updating and starts the median search for the next window from the median value of the previous window. Huang's method is extendable in the temporal dimension too, so that it can be used as a UWM filter for video [19]. The computational cost increases as the window size increases in the $\mathcal{O}(r)$ sliding window approach. However, the computational cost is independent of the window size in the $\mathcal{O}(1)$ approach [20], [21]. In the $\mathcal{O}(1)$ approach, the histogram of a horizontal sliding window is updated by respectively subtracting and adding the histograms of the column excluded and the column included by the sliding window. To accelerate the median search, a coarse-level histogram that accumulates only the higher order bits of pixels is used to reduce the number of bins to search for and update. SIMD operations are used to accelerate the addition

and subtraction between histograms. The $\mathcal{O}(1)$ sliding window approach is also used in the arc-distance median filter [22].

The sort-based methods first sort the pixel values in the window and then search for their median. Similar to the histogram-based methods, the sorting-based methods use redundancy in the window's overlapping region of adjacent pixels and this accelerates the finding of the median. Chaudhuri [23] used the $\mathcal{O}(r)$ sliding window approach to compute rank orders in a window. This can be used for min, max, or median filters. When updating ordered elements through a sliding window, the elements of the excluded column are removed, and the elements of the windows newly included column are inserted using the principle of mergesort. Sánchez and Rodríguez [24] proposed a UWM filter that combines a sorting algorithm, based on the complementary cumulative distribution function, with the $\mathcal{O}(1)$ sliding window approach. Adams [25] proposed fast UWM filters using separable sorting networks. The key idea for the accelerating median search is to share most of the sorting tasks between adjacent pixels. Sort-based methods work faster than histogram-based methods for small to medium window sizes or high precision data.

B. The Weighted Median Filter

The WM filter replaces the pixel value of each pixel with the weighted median value in a window centered at each pixel. The weights are calculated from an input image itself or from a guide image. Since the weights vary from window to window, it is difficult to accelerate the construction of the weighted histogram using a sliding window approach as is used in UWM filters. Ma et al. [14] proposed the first $\mathcal{O}(1)$ time WM filter. This method projected a 2D input image into a 3D space, where the extended coordinate was a histogram bin of the input image. By applying an $\mathcal{O}(1)$ edge-preserving filter to 2D slices with the same bin, the method achieved $\mathcal{O}(1)$ time computation. Since filtering is repeated for the number of bins, it is difficult to perform real-time processing for high precision images. Zhang et al. [17] used a joint histogram with the $\mathcal{O}(r)$ sliding window approach to accelerate the construction of a weighted histogram. Since the joint histogram had a large number of bins, data traversal was time-consuming. To solve this problem, a data structure that allows fast traversal was introduced. The large memory requirement of the joint histogram made it difficult to apply the $\mathcal{O}(1)$ sliding window approach and to deal with multichannel and high precision data. Zhao et al. [26] applied the bilateral grid data structure [27] to WM filtering. Their method achieved real-time processing on 2D images when the sampling rate of the bilateral grid was high. Since their method was based on the bilateral grid, it had the same problems that the bilateral grid had. For example, using a small window size required fine sampling, which incurred large memory and computational costs. Due to memory limitations, processing high resolution, high-dimensional, multichannel, and high precision data requires coarse sampling, leading to significant degradation of results.

Zhang's method and Zhao's method cannot use the guided filter kernel. Ma's method can use the guided filter kernel, but it is computationally expensive. In this paper, we propose a fast WM filter based on the guided filter kernel.

III. A Weighted Median Formulation

In this section, we define the weighted median formulation used in this paper. Several formulations of weighted medians have been proposed [28], [29], [30]. The weighted median formulation is based on the formulation proposed in [14].

Let $f_x \in \mathcal{F}$ be a pixel value at x in a single channel input image f, where $\mathcal{F} = \{i \in \mathbb{Z} | f_{min} \leq i \leq f_{max}\}$ and $f_{min}, f_{max} \in \mathbb{Z}$ are the minimum and maximum values that f_x can take, respectively. In [14], the weighted median $f_x^\star$ in a local window centered at pixel x is defined as:

$$f_x^\star = \min_{j \in \mathcal{F}} j \quad \text{s.t.} \sum_{i=f_{min}}^{j} H_x(i) \geq \frac{1}{2} \sum_{i=f_{min}}^{f_{max}} H_x(i), \tag{1}$$

where H_x is a local weighted histogram and a value at bin i is calculated by

$$H_x(i) = \sum_{y \in \Omega_x} w_{x,y} f_y^\delta(i), \tag{2}$$

where Ω_x is a set of pixels inside the local window, $w_{x,y}$ is a weight calculated from the affinity between pixels x and y, and

$$f_x^\delta(i) = \begin{cases} 1 & (f_x = i) \\ 0 & (f_x \neq i). \end{cases} \tag{3}$$

In this paper, the weighted median formulation is extended as follows. A weighted cumulative histogram up to bin i can be expressed as

$$H_x^\downarrow(i) = \sum_{j=f_{min}}^{i} H_x(j) = \sum_{y \in \Omega_x} w_{x,y} \sum_{j=f_{min}}^{i} f_y^\delta(j). \tag{4}$$

The weights $w_{x,y}$ are often calculated from the coefficients of a smoothing filter. Since the sum of the coefficients of a smoothing filter is 1, the following holds:

$$H_x^\downarrow(f_{max}) = \sum_{y \in \Omega_x} w_{x,y} \sum_{j=f_{min}}^{f_{max}} f_y^\delta(j) = \sum_{y \in \Omega_x} w_{x,y} = 1. \tag{5}$$

From this, the following weighted median with smoothing filter coefficients can be derived:

$$f_x^\star = \min_{i \in \mathcal{F}} i \quad \text{s.t. } H_x^\downarrow(i) \geq 0.5. \tag{6}$$

To find its solution, we track the change in $H_x^\downarrow(i)$ with i. Suppose all the weights are non-negative. When tracking from $i = f_{min}$ with increasing i, the solution is i where $H_x^\downarrow(i)$ becomes 0.5 or higher for the first time. When tracking from $i = f_{max}$ with decreasing i, the solution is i just before $H_x^\downarrow(i)$ becomes less than 0.5 for the first time. By generalizing this observation, we define the weighted median tracked from a starting bin k as follows:

$$f_x^\star = \begin{cases} \min\limits_{i>k, i \in \mathcal{F}} i \quad \text{s.t. } H_x^\downarrow(i) \geq 0.5 & (H_x^\downarrow(k) < 0.5) \\ 1 + \max\limits_{i \leq k, i \in \mathcal{F}} i \quad \text{s.t. } H_x^\downarrow(i) < 0.5 & (H_x^\downarrow(k) \geq 0.5). \end{cases} \tag{7}$$

This weighted median formulation is used for the proposed method.

When the weights are non-negative, the solutions of (6) and (7) are the same. On the other hand, when the weights contain negative values, the solutions are not necessarily the same. As is seen in the next section, the weights used in the proposed method may contain negative values.

IV. The Fast Guided Median Filter

The construction of a histogram $H_x(i)$ shown in (2) impedes fast computation of WM filters. Section IV-A discusses the problems of the computation of $H_x(i)$ in the WM filter using the guided filter kernel. Section IV-B presents the proposed method to solve this problem. The proposed method efficiently obtains the weighted median using the median tracking technique [17] and the sliding window approach shown in Sec. IV-C.

A. Problems With the Guided Filter Kernel

For the histogram-based WM filter process, the weighted cumulative histograms are updated as follows:

$$H_x^{\downarrow}(i) = H_x^{\downarrow}(i-1) + H_x(i). \tag{8}$$

As this formula shows, fast computation of $H_x(i)$ is the key to achieving faster WM filtering. In the UWM filter, the histogram can be updated efficiently using a sliding approach. This subsection discusses the difficulties of using the same approach with the guided filter kernel.

Let $g_x \in \mathbb{R}$ be a pixel value at pixel x in a single channel guide image g. The guided filter [12] assumes that the filtering output e_x at x can be estimated by a linear transformation of g in a local window Ω_z centered at pixel z:

$$e_x = a_z g_x + b_z, \tag{9}$$

where

$$a_z = \frac{\frac{1}{|\Omega_z|}\sum_{y\in\Omega_z} g_y f_y - \mu(g_z)\mu(f_z)}{v_z}, \tag{10}$$

$$b_z = \mu(f_z) - a_z\mu(g_z), \tag{11}$$

$\mu(\cdot_\star)$ is the mean value of $\cdot$ in $\Omega_\star$, $|\Omega_\star|$ is the number of elements of $\Omega_\star$, and $v_\star = \sigma_\star + \epsilon$ where $\sigma_\star$ is a variance of g in $\Omega_\star$ and ϵ is a regularization parameter. The output value e_x is estimated in multiple windows, which is called multipoint modeling [31]. In the rest of this paper, we call the guided filter proposed in [12] the multipoint guided filter. The final estimate is calculated as the average of multiple estimates.

$$e_x = \mu(a_x)g_x + \mu(b_x). \tag{12}$$

The weight of the multipoint guided filter is

$$w_{x,y} = \frac{1}{|\Omega_x|^2}\sum_{\{z|x,y\in\Omega_z\}}\left(1 + \frac{(g_x - \mu(g_z))(g_y - \mu(g_z))}{v_z}\right). \tag{13}$$

Here it is noted that the weights may contain negative values. The summation of the elements in the local window needed in the calculation of mean values can be computed in $\mathcal{O}(1)$ time using the separable summed area table (SSAT) or the one pass summed area table (OP-SAT) [32].

Using the multipoint guided filter kernel the histogram $H_x(i)$ is,

$$H_x(i) = \mu(a_x^\delta(i))g_x + \mu(b_x^\delta(i)), \tag{14}$$

where

$$a_x^\delta(i) = \frac{\frac{1}{|\Omega_x|}\sum_{y\in\Omega_x} g_y f_y^\delta(i) - \mu(g_x)\mu(f_x^\delta(i))}{v_x}, \tag{15}$$

$$b_x^\delta(i) = \mu(f_x^\delta(i)) - a_x^\delta(i)\mu(g_x). \tag{16}$$

To use a sliding approach, we attempt to rewrite these equations using histograms. Let $F_x(i) = \sum_{y\in\Omega_x} f_y^\delta(i)$ be a histogram of f at pixel x. $a_x^\delta(i)$ and $b_x^\delta(i)$ can be rewritten using $F_x(i)$ as

$$a_x^\delta(i) = \frac{\sum_{y\in\Omega_x} g_y f_y^\delta(i) - \mu(g_x)F_x(i)}{v_x|\Omega_x|}, \tag{17}$$

$$b_x^\delta(i) = \frac{1}{|\Omega_x|}F_x(i) - a_x^\delta(i)\mu(g_x). \tag{18}$$

These equations reveal three problems that arise when the multipoint guided filter kernel is used in WM filtering. First, $\sum_{y\in\Omega_x} g_y f_y^\delta(i)$ in (17) cannot be expressed using a histogram of f. The naive calculation of this term is computationally inefficient. Second, the computation of the mean value of $a_x^\delta(i)$ and $b_x^\delta(i)$ is time consuming. Since $a_x^\delta(i)$ and $b_x^\delta(i)$ depends on i, SSAT and OP-SAT cannot be used to calculate their mean values. Third, since many histograms are required, the calculation is inefficient and requires a lot of memory. For the calculation of the mean values of $a_x^\delta(i)$ and $b_x^\delta(i)$, it is necessary to keep histograms for all pixels in Ω_x.

B. Our Approach

Instead of the multipoint guided filter kernel, the proposed method uses a pointwise guided filter kernel, which uses pointwise modeling [31]. The filtering result is

$$e_x = a_x g_x + b_x = a_x(g_x - \mu(g_x)) + \mu(f_x). \tag{19}$$

This equation can also be formulated as

$$e_x = c_x\sum_{y\in\Omega_x} g_y f_y + d_x\sum_{y\in\Omega_x} f_y, \tag{20}$$

where

$$c_x = \frac{g_x - \mu(g_x)}{v_x|\Omega_x|},\ d_x = \frac{1}{|\Omega_x|} - \mu(g_x)c_x. \tag{21}$$

The weight in the pointwise guided filter can be expressed as

$$w_{x,y} = \frac{1}{|\Omega_x|}\left(1 + \frac{(g_x - \mu(g_x))(g_y - \mu(g_x))}{v_x}\right). \tag{22}$$

Unlike the multipoint guided filter, the weight of the pointwise guided filter is not symmetric, i.e., $w_{x,y} \neq w_{y,x}$.

Using (20), $H_x(i)$ can be expressed as

$$H_x(i) = c_x\sum_{y\in\Omega_x} g_y f_y^\delta(i) + d_x\sum_{y\in\Omega_x} f_y^\delta(i). \tag{23}$$

Algorithm 1 Search Weighted Median

Input: $W = \{F, G, f^{\downarrow}, g^{\downarrow}, k\}, c_x, d_x$
Output: $f_x^{\star}$

```
h↓ ← c_x g↓ + d_x f↓
if h↓ < 0.5 then
    while h↓ < 0.5 do
        k ← k + 1
        f↓ ← f↓ + F(k)
        g↓ ← g↓ + G(k)
        h↓ ← c_x g↓ + d_x f↓
    end while
    f*_x ← k
else
    while h↓ ≥ 0.5 do
        f↓ ← f↓ − F(k)
        g↓ ← g↓ − G(k)
        h↓ ← c_x g↓ + d_x f↓
        k ← k − 1
    end while
    f*_x ← k + 1
end if
```

Here, $g_x f_x^{\delta}(i)$ can be replaced with

$$g_x^{\delta}(i) = \begin{cases} g_x & (f_x = i) \\ 0 & (f_x \neq i). \end{cases} \tag{24}$$

Let $G_x(i) = \sum_{y \in \Omega_x} g_y^{\delta}(i)$ be a histogram of f weighted by g. Then

$$H_x(i) = c_x G_x(i) + d_x F_x(i) \tag{25}$$

which is that histogram $H_x(i)$ can be calculated as a weighted sum of the histograms $G_x(i)$ and $F_x(i)$. Equation (25) is the core result of this research. A fast WM filter using a guided filter kernel can be achieved because all the elements used to compute H_x can be computed fast. G_x and F_x can be obtained efficiently using a sliding window approach. Since the weights c_x and d_x are independent of i, they can be computed efficiently using SSAT or OP-SAT.

C. Efficient Median Search

Let $W = \{F, G, f^{\downarrow}, g^{\downarrow}, k\}$ be the local window at the pixel of interest x, where F and G are the histograms of W, and $f^{\downarrow} = \sum_{j=f_{min}}^{k} F(j)$ and $g^{\downarrow} = \sum_{j=f_{min}}^{k} G(j)$ are values at the tracking bin k of the cumulative histogram of F and G, respectively. Equation (8) can be rewritten as

$$H_x^{\downarrow}(i) = c_x(g^{\downarrow} + G(i)) + d_x(f^{\downarrow} + F(i)). \tag{26}$$

The pseudocode for the median search of the proposed method is shown in Algorithm 1.

For fast computation of the WM filter, the proposed method uses the median tracking technique and the sliding window approach. Here we will discuss the case where the input image and the guide image are both 2D grayscale images of size $M \times N$. Let $W_{s,t}^{(2)}$ be a 2-dimensional square window of size $2r+1$ centered at pixel (s,t) and $W_{s,t}^{(1)}$ be a 1-dimensional column window of length $2r+1$. Here r is the window radius of $W_{s,t}^{(2)}$.

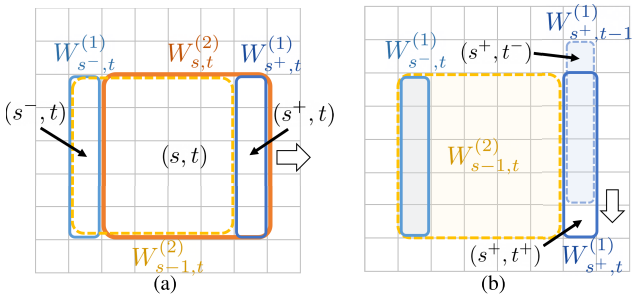

Fig. 2. (a) 2D window and column windows. (b) Column window update in the $\mathcal{O}(1)$ sliding window approach.

The median tracking technique can reduce the computation time for the median search. This technique starts the calculation of $f_{s,t}^{\star}$ in the window $W_{s,t}^{(2)}$ from k which is the ending bin used for the calculation of $f_{s-1,t}^{\star}$ in the previous window $W_{s-1,t}^{(2)}$. Here, k is $f_{s-1,t}^{\star}$ when the median value is searched in ascending order in $W_{s-1,t}^{(2)}$, and $f_{s-1,t}^{\star} - 1$ when the median value is searched in descending order. This method works because the median value does not change much between neighboring pixels and this aids in the search for the median. Since the weights used in the proposed method can be negative values, the search from f_{min} and the search using median tracking may have different solutions, as mentioned in Sec. III. Since it rarely occurs, it makes little visual difference. Less than 0.1 % of the pixels have different solutions in the 30 natural color images used in the experiment in Sec. VII-C.

The sliding window approach allows for efficient calculation of F, G, $f^{\downarrow}$, and $g^{\downarrow}$. When sliding a filter window to the right, window $W_{s,t}^{(2)}$ overlaps much of the region of window $W_{s-1,t}^{(2)}$. Let $s^- = s-r-1$ and $s^+ = s+r$. To obtain the data in window $W_{s,t}^{(2)}$, the sliding window approach adds data in window $W_{s^+,t}^{(1)}$ to data in window $W_{s-1,t}^{(2)}$ and removes data in window $W_{s^-,t}^{(1)}$ from data also in window $W_{s-1,t}^{(2)}$ (see Fig. 2 (a)). The $\mathcal{O}(r)$ and $\mathcal{O}(1)$ sliding window approaches differ in this update process.

The algorithm for the proposed weighted median filter based on the $\mathcal{O}(r)$ sliding window approach is shown in Algorithm 2. The data in $W_{s,t}^{(2)}$ is obtained by adding the pixels in $W_{s^+,t}^{(1)}$ to $W_{s-1,t}^{(2)}$ and removing the pixels in $W_{s^-,t}^{(1)}$ from $W_{s-1,t}^{(2)}$. The algorithm for adding pixels is shown in Algorithm 3. The computational complexity of the histogram update in the $\mathcal{O}(r)$ sliding window approach is proportional to the filter radius. Given an image with a pixel bit depth of n bits, $2(2^n+1)(n+2\log_2(2r+1))+n$ bits are required to store the data in the window. $f^{\downarrow}$ and $g^{\downarrow}$ need $n+2\log_2(2r+1)$ bits each, F and G need $2^n(n+2\log_2(2r+1))$ bits each, and k needs n bits.

The algorithm for the proposed weighted median filter based on the $\mathcal{O}(1)$ sliding window approach is shown in Algorithm 4. The data in $W_{s,t}^{(2)}$ is obtained by merging the data in $W_{s^+,t}^{(1)}$ into $W_{s-1,t}^{(2)}$ and separating the data in $W_{s^-,t}^{(1)}$

Algorithm 2 $\mathcal{O}(r)$ Sliding Window Approach

Input: f, g, c, d of size $M \times N$
Output: $f^\star$
  for $t = 1$ to M **do**
    Initialize $W^{(2)} = \{F^{(2)}, G^{(2)}, f^{(2)\downarrow}, g^{(2)\downarrow}, k^{(2)}\}$
    for $s = 1$ to N **do**
      for $t' = t - r$ to $t + r$ **do**
        Add pixel at (s^+, t') to $W^{(2)}$
        Remove pixel at (s^-, t') from $W^{(2)}$
      end for
      $f^\star_{s,t}$ = Search_weighted_median($W^{(2)}, c_{s,t}, d_{s,t}$)
    end for
  end for

Algorithm 3 Add Pixels to Window

Input: $W = \{F, G, f^\downarrow, g^\downarrow, k\}, f_x, g_x$
  $F(f_x) \leftarrow F(f_x) + 1$
  $G(f_x) \leftarrow G(f_x) + g_x$
  if $f_x \le k$ **then**
    $f^\downarrow \leftarrow f^\downarrow + 1$
    $g^\downarrow \leftarrow g^\downarrow + g_x$
  end if

Algorithm 4 $\mathcal{O}(1)$ Sliding Window Approach

Input: f, g, c, d of size $M \times N$
Output: $f^\star$
  Initialize array of $W^{(1)} = \{F^{(1)}, G^{(1)}, f^{(1)\downarrow}, g^{(1)\downarrow}, k^{(1)}\}$
  for $t = 1$ to M **do**
    Initialize $W^{(2)} = \{F^{(2)}, G^{(2)}, f^{(2)\downarrow}, g^{(2)\downarrow}, k^{(2)}\}$
    for $s = 1$ to N **do**
      Add pixel at (s^+, t^+) to $W^{(1)}[s^+]$
      Remove pixel at (s^+, t^-) from $W^{(1)}[s^+]$
      Merge $W^{(1)}[s^+]$ into $W^{(2)}$
      Separate $W^{(1)}[s^-]$ from $W^{(2)}$
      $f^\star_{s,t}$ = Search_weighted_median($W^{(2)}, c_{s,t}, d_{s,t}$)
    end for
  end for

Algorithm 5 Merge Windows

Input: $W^{(1)} = \{F^{(1)}, G^{(1)}, f^{(1)\downarrow}, g^{(1)\downarrow} k^{(1)}\}, W^{(2)} = \{F^{(2)}, G^{(2)}, f^{(2)\downarrow}, g^{(2)\downarrow}, k^{(2)}\}$
  if $k^{(1)} < k^{(2)}$ **then**
    while $k^{(1)} < k^{(2)}$ **do**
      $k^{(1)} \leftarrow k^{(1)} + 1$
      $f^{(1)\downarrow} \leftarrow f^{(1)\downarrow} + F^{(1)}(k^{(1)})$
      $g^{(1)\downarrow} \leftarrow g^{(1)\downarrow} + G^{(1)}(k^{(1)})$
    end while
  else
    while $k^{(1)} > k^{(2)}$ **do**
      $f^{(1)\downarrow} \leftarrow f^{(1)\downarrow} - F^{(1)}(k^{(1)})$
      $g^{(1)\downarrow} \leftarrow g^{(1)\downarrow} - G^{(1)}(k^{(1)})$
      $k^{(1)} \leftarrow k^{(1)} - 1$
    end while
  end if
  $f^{(2)\downarrow} \leftarrow f^{(2)\downarrow} + f^{(1)\downarrow}$
  $g^{(2)\downarrow} \leftarrow g^{(2)\downarrow} + g^{(1)\downarrow}$
  for $i = f_{min}$ to f_{max} **do**
    $F^{(2)}(i) \leftarrow F^{(2)}(i) + F^{(1)}(i)$
    $G^{(2)}(i) \leftarrow G^{(2)}(i) + G^{(1)}(i)$
  end for

from $W^{(2)}_{s-1,t}$. To achieve this update, column windows must be stored and updated. The number of column windows to be stored is equal to the image width N. Before the merging process, the column window $W^{(1)}_{s^+,t}$ is obtained by sliding $W^{(1)}_{s^+,t-1}$ vertically one pixel to add and remove pixels (see Fig. 2 (b)). The algorithm for merging two windows is shown in Algorithm 5. $F^{(2)}$ and $G^{(2)}$ are updated by adding the values of each bin of $F^{(1)}$ and $G^{(1)}$, respectively. Since $k^{(1)}$ and $k^{(2)}$ can be different, first $k^{(1)}$ is updated to be equal to $k^{(2)}$, then $f^{(1)\downarrow}$ and $g^{(1)\downarrow}$ are added to $f^{(2)\downarrow}$ and $g^{(2)\downarrow}$, respectively. The updated $W^{(1)}_{s^+,t}$ is used again to update $W^{(2)}_{s-1,t+1}$ in the next row. Since $k^{(2)}$ in $W^{(2)}_{s-1,t}$ and $W^{(2)}_{s-1,t+1}$ are expected to be close, $W^{(1)}_{s^+,t}$ updated with $k^{(2)}$ in $W^{(2)}_{s-1,t}$ is expected to have a small update when merged with $W^{(2)}_{s-1,t+1}$. The process of separating $W^{(1)}_{s^-,t}$ from $W^{(2)}_{s-1,t}$ is similar to the opposite process of merging but is partially different. Since $k^{(1)}$ in $W^{(1)}_{s^-,t}$ has been updated to a value suitable for merging in the next row, the separation process adjusts $f^{(2)\downarrow}$ and $g^{(2)\downarrow}$ in $W^{(2)}_{s-1,t}$ without updating $k^{(1)}$ in $W^{(1)}_{s^-,t}$, e.g., $f^{(2)\downarrow} \leftarrow f^{(2)\downarrow} + F^{(1)}(k^{(1)})$ instead of $f^{(1)\downarrow} \leftarrow f^{(1)\downarrow} + F^{(1)}(k^{(1)})$. The computational complexity of the histogram update in the $\mathcal{O}(1)$ sliding window approach is proportional to the bit depth of the image. Since column windows are used in addition to the main window $W^{(2)}$, $(2(n+2\log_2(2r+1))(2^n+1)+n)(M+1)$ bits are required to store the data of all the windows. While the $\mathcal{O}(1)$ sliding window approach has an advantage over the $\mathcal{O}(r)$ sliding window approach in terms of computational complexity due to the window size, it needs additional memory and computation for the column windows.

V. Extensions to the WM Filter

The proposed method can be extended to multidimensional, multichannel, and higher precision data.

A. The Multidimensional Extension

The proposed method can easily be applied to multidimensional images because SSAT and OP-SAT used in the calculation of c_x and d_x and the sliding window approach used in histogram updating can be straightforwardly extended to multidimensional images.

B. Using Multichannel Guide Images

Like various filters, a multichannel input image is processed for each channel. Since c_x and d_x are commonly used for all channels of an input image, they need to be calculated only once.

In the case when the guide image is multichannel, the pointwise guided filter can be extended in the same way as the multipoint guided filter. Let $\mathbf{g}_x \in \mathbb{R}^m$ be a vector whose

entries are the pixel value of each channel at pixel x in the guide image $\mathbf{g}$, where m denotes the number of channels of the guide image. The output of the pointwise guided filter with a multichannel guide image is expressed as

$$e_x = \mathbf{c}_x^\top \sum_{y \in \Omega_x} \mathbf{g}_y f_y + d_x \sum_{y \in \Omega_x} f_y, \tag{27}$$

where $\cdot^\top$ denotes the transpose of the matrix,

$$\mathbf{c}_x = V_x^{-1} \frac{\mathbf{g}_x - \mu(\mathbf{g}_x)}{|\Omega_x|}, \quad d_x = \frac{1}{|\Omega_x|} - \mathbf{c}_x^\top \mu(\mathbf{g}_x), \tag{28}$$

$V_x = \Sigma_x + \epsilon I$, $\Sigma_x \in \mathbb{R}^{m \times m}$ is a covariance matrix of $\mathbf{g}$ in Ω_x, I is the identity matrix, and $\mu(\mathbf{g}_x) \in \mathbb{R}^m$ is a vector whose entries are the mean values of each channel of $\mathbf{g}$ in Ω_x. Using (27), the histogram H_x is derived in the same way as for the single channel as follows:

$$H_x(i) = \mathbf{c}_x^\top \mathbf{G}_x(i) + d_x F_x(i), \tag{29}$$

where $\mathbf{G}_x$ is a weighted histogram of f and $\mathbf{G}_x(i) \in \mathbb{R}^m$ is a vector with values for each channel bin.

We analyze the behavior of the regularization parameter ϵ, which controls the smoothing effect, when using multichannel guide images. Consider the case where all channels of $\mathbf{g}$ are the same. That is, $\mathbf{g}_x = g_x \mathbf{1}_m$, where $\mathbf{1}_m \in \mathbb{R}^m$ is a vector, whose entries are all 1. Here, the same output is wanted as when the single channel g is used as the guide image. Substituting $\mathbf{g}_x = g_x \mathbf{1}_m$ into (27), we obtain

$$e_x = \alpha_x (g_x - \mu(g_x)) + \mu(f_x), \tag{30}$$

where

$$\alpha_x = \mathbf{1}_m^\top \tilde{V}_x^{-1} \mathbf{1}_m \left(\frac{1}{|\Omega_x|} \sum_{y \in \Omega_x} g_y f_y - \mu(g_x)\mu(f_x) \right), \tag{31}$$

and $\tilde{V}_x = \sigma_x \mathbf{1}_m \mathbf{1}_m^\top + \epsilon I$. Using the Sherman-Morrison formula, we obtain

$$\mathbf{1}_m^\top \tilde{V}_x^{-1} \mathbf{1}_m = \left(\sigma_x + \frac{\epsilon}{m} \right)^{-1}. \tag{32}$$

As can be seen by comparing with (19), ϵ becomes relatively small as the number of channels increases. To obtain the same level of smoothing effect when the guide image has multiple channels compared to when the guide image has a single channel, ϵ should be multiplied by the number of channels. The same is true for the multipoint guided filter kernel.

C. High Precision Extensions

Theoretically, the proposed method described so far can be applied to high precision data. However, high precision data causes an explosive increase in the size of the histogram, which leads to practical problems. One is the requirement for a large amount of memory to store histograms and another is the increased computational cost of updating the histograms. In a UWM filter, using the ordinal transform solves these problems because the transform can reduce the number of bins in the histogram [33]. On the other hand, the proposed method cannot use the ordinal transform because (25) is not invariant to the transform.

This problem is solved by having the histogram in a linked list of high precision data in ascending order. Both $\mathcal{O}(r)$ and $\mathcal{O}(1)$ sliding window methods can use this solution. This paper discusses the $\mathcal{O}(1)$ sliding window approach to sorted linked lists. Adding elements to the linked list is like the merge process of a mergesort, although slightly different. Elements with the same bin are not inserted into the list but are added to the existing elements. Please for a moment assume that the linked list of the histogram of a column window has already been sorted during processing in the previous row. Then, adding one pixel to the histogram of the column window merges one element into the linked list. Adding the histogram of the column window to the histogram of the main window merges two sorted lists. When removing elements of the histogram of the column window from the histogram of the main window, bins whose value becomes zero are removed from the list. The computation time for updating the linked list is proportional to the length of the linked list. As the window size increases, the computation time tends to increase because the possibility of having values at various bins increases.

VI. The Implementation Details

In this section, we describe CPU and GPU implementations of the proposed method. In general, a CPU is composed of fewer cores and more cache memory than a GPU. Due to this difference in composition, the suitable sliding window approach differs between CPU and GPU, which will be discussed in Sec. VI-A. In the implementation of the proposed method, the $\mathcal{O}(r)$ sliding window approach is used for the GPU implementation as shown in Sec. VI-B, and the $\mathcal{O}(1)$ sliding window approach is used for the CPU implementation as shown in Sec. VI-C. We also describe the implementation for high precision data in Sec. VI-D.

A. Sliding Window Approach for CPU and GPU

The $\mathcal{O}(1)$ sliding window approach is superior to the $\mathcal{O}(r)$ sliding window approach in computational time. However, while this is true for single-threaded environments, it is not necessarily true for multi-threaded environments. In the $\mathcal{O}(1)$ sliding window approach, each row cannot be computed independently because the data of the column window updated in the previous row is needed. In a parallel implementation of the $\mathcal{O}(1)$ sliding window approach, an input image is divided into multiple blocks, and these blocks are processed in parallel. A small number of divisions is ineffective in terms of parallelism while a high number of divisions causes a lot of overhead, which is mainly in the construction of the histograms of column windows. For fast computation, it is necessary to store the data of the column windows in the cache memory, which needs a large amount of cache memory. The $\mathcal{O}(1)$ sliding window approach is therefore suitable for CPUs but not for GPUs. [34] shows that the CPU implementation outperforms the GPU implementation in terms of computational speed in the UWM filter implementation using the $\mathcal{O}(1)$ sliding window approach. Unlike the $\mathcal{O}(1)$ sliding window approach, the $\mathcal{O}(r)$ sliding window approach does not have the overhead of parallelization because each

row is computed independently. Also, it does not require a large cache for execution because it does not need the data of the column windows. Therefore, the $\mathcal{O}(r)$ sliding window approach is suitable for processors with many cores and a small cache, such as GPUs.

B. GPU Implementation With $\mathcal{O}(r)$ Sliding Window Approach

In this paper, the NVIDIA's CUDA platform is used to implement the proposed method on GPUs.

In the $\mathcal{O}(r)$ sliding window approach on GPU, c_x and d_x are first calculated using SSAT, and then the weighted median is calculated by sliding a window vertically for each image column. The reason for sliding vertically instead of horizontally is to achieve coalesced memory access for efficient pixel loading.

High parallelism and efficient memory access are required to achieve high-speed processing. Since the data of the sliding window are accessed frequently, they should be stored in on-chip memory, such as the L1 cache or shared memory. One viable way to achieve high parallelism is to allocate one thread for each image column. This approach, however, achieves low parallelism. Since each thread stores the data of the sliding window, running a small number of threads consumes most of the on-chip memory in one thread block. Therefore, the number of threads running per thread block is small. Furthermore, the window update is inefficient because it is not parallelized.

In the proposed implementation, one thread block is allocated for each image column. Each thread block has the same number of threads as the width of a local window. When a local window slides, each thread loads a pixel in parallel and updates F, G, $f^{\downarrow}$ and $g^{\downarrow}$ in parallel using atomic operations. The median tracking process is performed by one of the threads in a thread block. The pseudocode for the GPU kernel is shown in Algorithm 6. Except when using a small filter radius, this implementation uses more threads than the implementation of allocating one thread for each image column. In practical applications, this implementation is effective because a large filter radius is often used.

C. CPU Implementation With $\mathcal{O}(1)$ Sliding Window Approach

Unlike the $\mathcal{O}(r)$ sliding window on GPUs, we compute the weighted median while computing c_x and d_x using OP-SAT, because both computations are one-pass algorithms. For multichannel input, c_x and d_x are saved in the first channel processing and loaded in the subsequent channels processing without recalculating c_x and d_x.

For high-speed computation, we use SIMD operations and multi-threaded processing. Since the histogram update is simply the addition and subtraction of elements between the same bins of histograms, it can be accelerated by using SIMD operations. In our implementation, we use Intel AVX2 instructions, which have 256-bit vector integer instructions. We assign 32 bits to each bin of a histogram, so we can update 8 bins in parallel with a single instruction.

Algorithm 6 GPU Kernel Implementation

Input: f, g, c, d of size $M \times N$
Output: $f^{\star}$
  if threadIdx == r **then**
    Initialize $W = \{F, G, f^{\downarrow}, g^{\downarrow}, k\}$
  end if
  s = blockIdx + threadIdx - r
  __syncthreads()
  for $t = 1$ to M **do**
    atomicAdd($F(f_{s,t^+}), 1$)
    atomicAdd($G(f_{s,t^+}), g_{s,t^+}$)
    atomicSub($F(f_{s,t^-}), 1$)
    atomicSub($G(f_{s,t^-}), g_{s,t^-}$)
    if $f_{s,t^+} \leq k$ **then**
      atomicAdd($f^{\downarrow}, 1$)
      atomicAdd($g^{\downarrow}, g_{s,t^+}$)
    end if
    if $f_{s,t^-} \leq k$ **then**
      atomicSub($f^{\downarrow}, 1$)
      atomicSub($g^{\downarrow}, g_{s,t^-}$)
    end if
    __syncthreads()
    if threadIdx == r **then**
      $f^{\star}_{s,t}$ = Search_weighted_median($W, c_{s,t}, d_{s,t}$)
    end if
    __syncthreads()
  end for

A strategy for multi-threaded processing is to divide an image into vertical, horizontal, or both partitions and assign each thread to process each partial image. Note that this strategy incurs the overhead of initializing the histograms of a 2D window and column windows for each partial image. In our implementation, we simply divide an image into T vertical partitions, where T is the number of threads used. The optimal division method and number depend on the image size, window size, CPU cache size, etc. Consideration of a more appropriate division is beyond the scope of this paper.

D. $\mathcal{O}(1)$ Sliding Window on CPU for High Precision Data

With high precision data, the $\mathcal{O}(1)$ sliding window approach is used on CPU with the linked list-based histogram as described in Sec. V-C. This implementation searches for a median from f_{min} instead of using median tracking because it takes time to find specific elements from a linked list. SIMD operations are not used because they cannot update the histogram for the linked list effectively. However, multi-threaded processing is used to process each partial image in parallel.

VII. Experimentation

A. Experimental Setup

The specification of the computer conducting the experiment was a Core i7-8700 @ 3.20 GHz CPU with 6 cores (12 threads) and 64 GB of RAM, and an NVIDIA GeForce GTX 1060 @ 1.51 GHz graphics card with 1152 CUDA cores

and 3 GB of RAM. We used the three types of the proposed methods shown in Sec. VI. We name each implementation as follows: GPU-$\mathcal{O}(r)$ (Sec. VI-B), CPU-$\mathcal{O}(1)$ (Sec. VI-C), and List-$\mathcal{O}(1)$ (Sec. VI-D). The proposed method was compared with the following conventional methods: The UWM filter, the constant time WM filter (CT-$\mathcal{O}(1)$) [14] and a hundred times or faster WM filter (FWM-$\mathcal{O}(r)$) [17]. All methods were implemented in the C++ programming language. The OpenMP application programming interface was used for parallel computation of the CPU implementation. The CUDA toolkit was used for the GPU implementation. CT-$\mathcal{O}(1)$ was coded using CUDA by the author. It employs the multipoint guided filter [12] for calculating weights. The code of FWM-$\mathcal{O}(r)$ is provided by the author and is slightly modified for parallelization using OpenMP. The default parameters that were set in the code of FWM-$\mathcal{O}(r)$ were used. I.e., the weights were calculated as

$$w_{x,y} = \exp\left(-\frac{(g_x - g_y)^2}{2\sigma^2}\right), \tag{33}$$

where σ is a parameter.

B. Noise Reduction Performance

In this experiment, noise reduction performance is verified. One scenario in which WM filters are used is to remove noise contained in a disparity image using a color image as a guide image. We performed denoising of a disparity image estimated from a 4D light field image. We used 16 scenes in the *Additional* category of the dataset [35]. Each scene consisted of a 4D color light field image with a spatial resolution of 512×512 and an angular resolution (the number of viewpoints) of 9×9. First, an 8-bit disparity image of the central viewpoint was estimated using the initial estimation process in the disparity estimation method [36]. Next, we denoised the initial estimation result using the GPU-$\mathcal{O}(r)$ filter with the color image of the central viewpoint of the light field image as a guide image. Finally, the mean squared errors (MSEs) were obtained using the ground truth of the disparity image provided in the dataset. The filters and their parameters used in the experiment were as follows: The UWM filter with $r_s = 14$, the FWM-$\mathcal{O}(r)$ filter with $r_s = 16$ and $\sigma = 25.5$, the CT-$\mathcal{O}(1)$ filter with $r_s = 24$ and $\epsilon = 0.51^2$, and the GPU-$\mathcal{O}(r)$ filter with $r_s = 18$ and $\epsilon = 2.55^2$. Here r_s was the spatial radius of a filter window. For a fair comparison, parameters were adjusted for the best results in terms of MSEs.

The MSEs are shown in Table I. The WM filters showed higher denoising performance compared to the UWM filter. The CT-$\mathcal{O}(1)$ and GPU-$\mathcal{O}(r)$ filters, which used kernels based on the guided filter, showed higher denoising performance than the FWM-$\mathcal{O}(r)$ filter, which used the Gaussian kernel based on color/intensity distance. Fig. 3 shows how the CT-$\mathcal{O}(1)$ filter and the GPU-$\mathcal{O}(r)$ filter helped recover the structural details. Compared to GPU-$\mathcal{O}(r)$, the CT-$\mathcal{O}(1)$ filter showed higher performance. This may be due to the CT-$\mathcal{O}(1)$ filter using the kernel based on multipoint modeling, while the GPU-$\mathcal{O}(r)$ filter used the kernel based on pointwise modeling. Multipoint modeling has better estimation performance than pointwise modeling because of its redundant estimation [31].

TABLE I
AVERAGE MSE ($\times100$) ON NOISE REDUCTION FOR DISPARITY IMAGE

UWM	FWM-$\mathcal{O}(r)$	CT-$\mathcal{O}(1)$	GPU-$\mathcal{O}(r)$
11.3	8.62	7.21	7.83

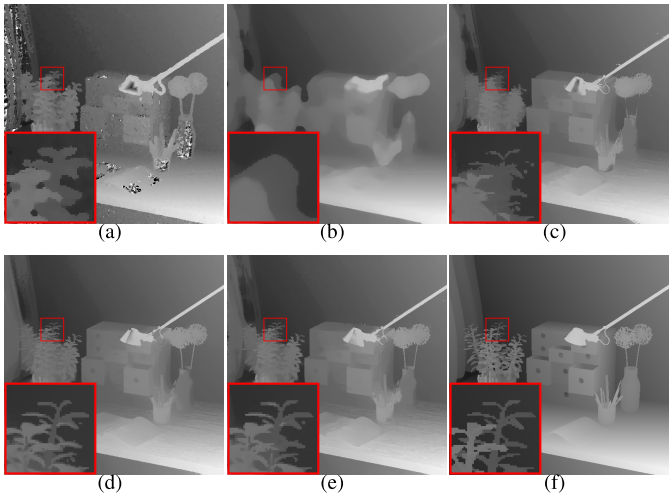

Fig. 3. Zoomed up of denoising results of estimated disparity in dataset *table*. (a) Estimated disparity in initial estimation process of [36]. (b) UWM filter. (c) FWM-$\mathcal{O}(r)$ filter. (d) CT-$\mathcal{O}(1)$ filter. (e) GPU-$\mathcal{O}(r)$ filter. (f) Ground truth.

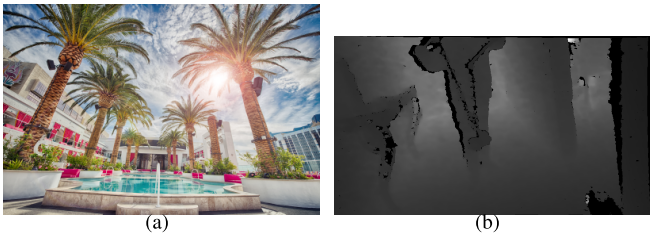

Fig. 4. Examples of test images. (a) Color image. (b) Depth image.

C. Computation Time of Natural Images

In this experiment, the computation time of each method was evaluated for grayscale and color images (8 bits/channel) with different image sizes and window sizes. The dataset [37] was used. It contained high resolution natural color images of assorted sizes. Thirty images of size 2040×1356 were selected, resized, and converted to grayscale for the experiment. Figure 4 (a) shows one of the test images.

The computation times for three different image sizes (640×480, 1024×768, and 1920×1080) with window size 15 are shown in Table II. The CPU-$\mathcal{O}(1)$ filter and GPU-$\mathcal{O}(r)$ filter were clearly faster than the conventional methods. List-$\mathcal{O}(1)$ was suitable for high precision data and not for 8-bit natural images, but it was competitive with the FWM-$\mathcal{O}(r)$ filter and faster than the CT-$\mathcal{O}(1)$ filter. For color images, the computation times of all methods increased significantly because the guide images were also color images. When the FWM-$\mathcal{O}(r)$ filter uses a color guide image, the image requires quantization into 256 different values, which increases the computation time as overhead. The significant increase in computation time of the CT-$\mathcal{O}(1)$ filter and our proposed methods, both of which are the guided filter-based methods,

TABLE II
AVERAGE COMPUTATION TIMES (IN MILLISECONDS) WITH RESPECT TO DIFFERENT IMAGE SIZES FOR GRAYSCALE AND COLOR IMAGES

	640×480		1024×768		1920×1080	
	Gray	Color	Gray	Color	Gray	Color
FWM-$\mathcal{O}(r)$ [17]	31.4	188.8	66.6	331.5	146.0	618.1
CT-$\mathcal{O}(1)$ [14]	216.8	1285.8	465.8	2772.1	1190.2	7176.2
CPU-$\mathcal{O}(1)$	12.5	90.2	30.0	193.6	68.2	409.0
List-$\mathcal{O}(1)$	51.4	185.6	93.1	357.9	223.5	817.2
GPU-$\mathcal{O}(r)$	3.1	21.5	5.8	40.9	13.7	100.5

was due to the inverse matrix operations shown in (28). Given these reasons, despite the difficulty of efficient computation for large color images, the GPU-$\mathcal{O}(r)$ filter achieved near real-time processing speed. Of particular emphasis is the fact that both the CT-$\mathcal{O}(1)$ and GPU-$\mathcal{O}(r)$ filters use kernels based on the guided filter with high noise reduction performance, but the GPU-$\mathcal{O}(r)$ filter is more than 50 times faster than the CT-$\mathcal{O}(1)$ filter.

The computation times for images of size 1920×1080 with different window sizes are shown in Fig. 5. The computation times of the CT-$\mathcal{O}(1)$ and CPU-$\mathcal{O}(1)$ filters were nearly constant for increasing the window size. The slight increase in computation time with increasing window size in the CPU-$\mathcal{O}(1)$ filter is due to the increase in overhead caused by parallelization. The computation time of the List-$\mathcal{O}(1)$ filter increased as the window size increased. This is because the length of the linked list tends to become longer as the window size increases, as mentioned in Sec. V-C. The computation time of the FWM-$\mathcal{O}(r)$ and GPU-$\mathcal{O}(r)$ filters increased as the window size increased. Despite $\mathcal{O}(r)$ time algorithm, the GPU-$\mathcal{O}(r)$ filter was faster than other $\mathcal{O}(1)$ time algorithms.

D. Application to High Precision Data

In this experiment, we verify the behavior of our proposed methods for high precision data. We used 16-bit depth images in the dataset [38]. Thirty images of size 1280×720 were selected. To measure the computation time for different bits, we created 8-, 10-, 12-, and 14-bit data by quantizing the original data. Figure 4 (b) shows one of the test images. The window size was set to 15.

The computation times with different bit depths are shown in Fig. 6. The GPU-$\mathcal{O}(r)$ filter was not able to manage 14-bit or higher data because the histograms could not be stored on the shared memory of the GPU. The CPU-$\mathcal{O}(1)$ filter was able to manage 16-bit data. Its computation time for 16-bit data was about 600 times longer than one in the List-$\mathcal{O}(1)$ filter. Most of the computation time was spent on adding and subtracting histograms with 2^{16} bins. The List-$\mathcal{O}(1)$ filter operated at near real-time speed for 16-bit data. Unlike the results for 8-bit grayscale images in Sec. VII-C, the computation time of the List-$\mathcal{O}(1)$ filter was shorter than that of the CPU-$\mathcal{O}(1)$ filter for the 8-bit data in this experiment. Because the histogram distribution of the depth data used in the experiment was sparse, the linked list of the depth data was shorter than that of natural images. The List-$\mathcal{O}(1)$ filter runs faster the sparser the distribution of the histogram is, while the CPU-$\mathcal{O}(1)$ filter does not depend on the distribution of the histogram for its computation speed.

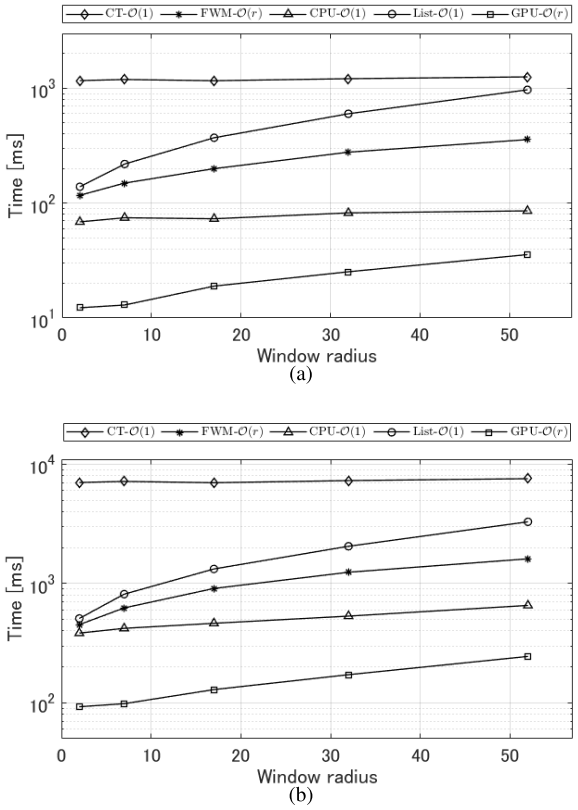

Fig. 5. Average computation times with respect to different window radii for (a) grayscale and (b) color images.

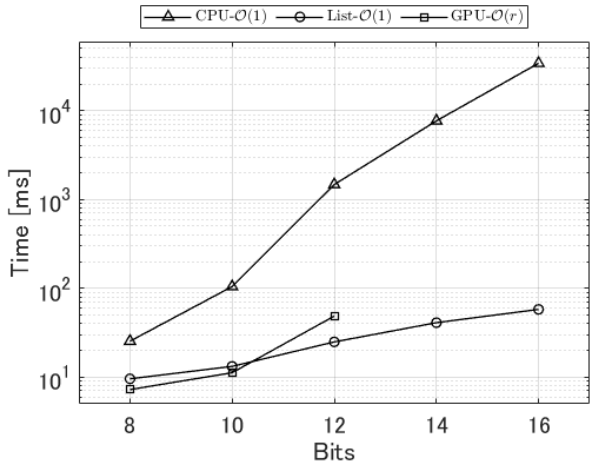

Fig. 6. Average computation times for each precision.

E. Application to Multichannel Images

In this experiment, the behavior of the proposed methods for multichannel data is verified. The University of Pavia dataset [39] was used, which is a multispectral image of size 610×340 with 103 bands. Salt-and-pepper noise with a probability of 0.05 was added to each channel.

TABLE III
AVERAGE PSNRS OF IMAGES FILTERED WITH DIFFERENT CHANNEL RADII r_c ON NOISE REDUCTION FOR MULTISPECTRAL IMAGE

	UWM	FWM-$\mathcal{O}(r)$	CT-$\mathcal{O}(1)$	GPU-$\mathcal{O}(r)$	GPU-$\mathcal{O}(r)$ with multichannel guide			
r_c	-	-	-	-	1	3	5	7
PSNR [dB]	32.36	32.36	31.25	32.36	30.64	33.75	34.04	34.12

TABLE IV
AVERAGE MSES OF IMAGES FILTERED WITH DIFFERENT ANGULAR RADII r_a ON DISPARITY REFINEMENT FOR LIGHT FIELD IMAGE

	UWM	FWM-$\mathcal{O}(r)$	CT-$\mathcal{O}(1)$	GPU-$\mathcal{O}(r)$				
r_a	-	-	-	0	1	2	3	4
MSE ($\times 100$)	12.1	9.43	8.03	8.60	8.25	8.03	7.90	7.84

We performed multispectral image denoising using the GPU-$\mathcal{O}(r)$ filter and varied the number of channels as a guide image. The number of channels to be used as a guide image was determined by the channel radius r_c. Filtering for the i-th channel with r_c used channels from $i - r_c$ to $i + r_c$ except i as a guide image. The number of channels of the guide image to be used was expressed as $2r_c$. The reason why the i-th channel was not used for the guide image was that if the guide image was the same as the input image, it would have been difficult to efficiently remove noise in the input image. As described in Sec. V-B, ϵ was adjusted for the number of channels in the guide image as follows: $\epsilon = 2\, r_c \hat{\epsilon}$. We set the spatial window radius to 1 and $\hat{\epsilon} = 55^2$. For comparison, the following methods were applied to each channel for denoising: The UWM filter with $r_s = 1$, the FWM-$\mathcal{O}(r)$ filter with $r_s = 1$ and $\sigma = 510$, the CT-$\mathcal{O}(1)$ filter with $r_s = 1$ and $\epsilon = 255^2$, and the GPU-$\mathcal{O}(r)$ filter with $r_s = 1$ and $\epsilon = 510^2$. Parameters were adjusted for best results in terms of the average peak signal-to-noise ratios (PSNRs).

The PSNRs of the filtered images are shown in Table III. The FWM-$\mathcal{O}(r)$, CT-$\mathcal{O}(1)$ and GPU-$\mathcal{O}(r)$ filters gave the best results using large σ and ϵ. If the pixel of interest is affected by noise, the weights computed from that pixel result in improper smoothing. While the use of large σ and ϵ reduces the effect of noise in the guide image, their kernels approach the mean filter kernel. As a result, they are almost identical to the UWM filter. As can be seen from (13), the multipoint guided filter kernel used in the CT-$\mathcal{O}(1)$ filter considers overlapping local windows. Therefore, the range of pixels used to calculate the output is $2r_s$. This leads to a lower PSNR for the CT-$\mathcal{O}(1)$ filter with $r_s = 1$ compared to the UWM filter with $r_s = 1$. As shown in Fig. 7, these methods degrade luminance changes that are not noise in the subject. In the GPU-$\mathcal{O}(r)$ filter with a multichannel guide, the use of more channels achieved better performance. Figure 8 shows the input channels of bands 78 to 82. The luminance changes that the subject has commonly appear in multiple channels at the same coordinates, while the luminance changes due to noise appear only in some channels at the same coordinates. The use of multiple channels may lead to appropriate weights based on similarities between channels.

F. Application to Multidimensional Data

In this experiment, the behavior of the proposed methods is verified for multidimensional data. Denoising of a 4D light field image was performed. The same dataset used in the experiments in Sec. VII-B was used. Disparities of all viewpoint estimates were obtained by using the initial estimation process of the disparity estimation method [36]. The disparities of all viewpoints could be regarded as 8-bit 4D data of size $512 \times 512 \times 9 \times 9$. Next, the initial estimation results were denoised using the GPU-$\mathcal{O}(r)$ filter using the 4D color light field image as a guide image. The angular radius r_a of the filter varied from 0 to 4. An $r_a = 0$ meant that the disparity of each viewpoint was filtered independently. I.e., it is equivalent to applying a 2D filter instead of a 4D filter. The spatial window radius was set to 20 and $\hat{\epsilon} = 2.55^2$. For comparison, the following methods were applied to each view for denoising: The UWM filter with $r_s = 14$, the FWM-$\mathcal{O}(r)$ filter with $r_s = 16$ and $\sigma = 25.5$, and the CT-$\mathcal{O}(1)$ filter with $r_s = 24$ and $\epsilon = 2.55^2$. The parameters were adjusted for the best results in terms of MSEs.

The MSEs of filtered images with different angular radii are shown in Table IV. Figure 9 shows the refinement results of the upper left viewpoint. As can be seen by comparing the result of $r_a = 0$ with the other results, using the 4D filter instead of the 2D filter ($r_a = 0$) significantly improved the refinement performance.

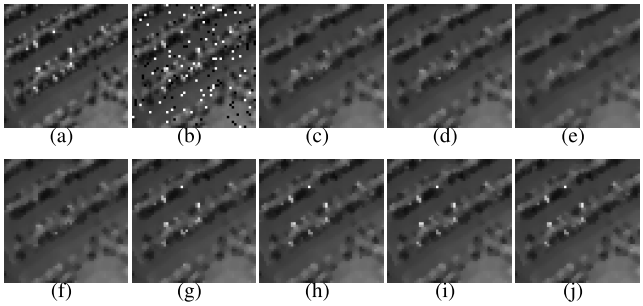

Fig. 7. Denoising results of a multispectral image (band 80). (a) Original image. (b) Noisy image. (c) UWM filter. (d) FWM-$\mathcal{O}(r)$ filter. (e) CT-$\mathcal{O}(1)$ filter. (f) GPU-$\mathcal{O}(r)$ filter with single channel guide. (g), (h), (i) and (j) are GPU-$\mathcal{O}(r)$ filter with $r_c = 1, 3, 5$ and 7, respectively.

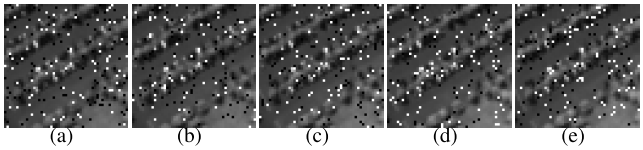

Fig. 8. Noisy input channels. (a) Band 78. (b) Band 79. (c) Band 80. (d) Band 81. (e) Band 82.

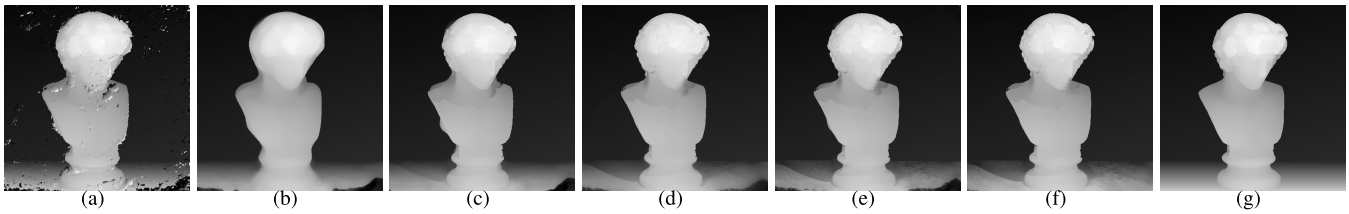

Fig. 9. The denoising results for the upper left viewpoint of estimated disparity in the dataset *antinous*. (a) The estimated disparity in the initial estimation process of [36]. (b) UWM filter. (c) FWM-$\mathcal{O}(r)$ filter. (d) CT-$\mathcal{O}(1)$ filter. (e) GPU-$\mathcal{O}(r)$ filter with $r_a = 0$. (f) GPU-$\mathcal{O}(r)$ filter with $r_a = 4$. (g) Ground truth disparity.

VIII. Conclusion

In this paper, we proposed a novel fast WM filter with the pointwise guided filter kernel. The proposed method can efficiently update the weighted histogram, which is the bottleneck in the real-time processing of WM filters. Fast computation is achieved by combining a sliding window approach with a median tracking technique. While conventional methods struggle to handle multidimensional, multichannel, and high precision images, the proposed method can manage them without data reduction. The experimentation results showed that the proposed method realizes faster computation than conventional WM filters. Furthermore, the proposed method, which calculates weights based on the guided filter kernel, has a higher noise reduction performance than the conventional method which calculates weights using the Gaussian kernel.

References

[1] M. Hilts and C. Duzenli, "Image filtering for improved dose resolution in CT polymer gel dosimetry," *Med. Phys.*, vol. 31, no. 1, pp. 39–49, Jan. 2004.

[2] C. P. Behrenbruch, S. Petroudi, S. Bond, J. D. Declerck, F. J. Leong, and J. M. Brady, "Image filtering techniques for medical image post-processing: An overview," *Brit. J. Radiol.*, vol. 77, pp. 126–132, Dec. 2004.

[3] S. Gan, S. Wang, Y. Chen, X. Chen, and K. Xiang, "Separation of simultaneous sources using a structural-oriented median filter in the flattened dimension," *Comput. Geosci.*, vol. 86, pp. 46–54, Sep. 2016.

[4] Y. Chen, S. Zu, Y. Wang, and X. Chen, "Deblending of simultaneous source data using a structure-oriented space-varying median filter," *Geophys. J. Int.*, vol. 222, no. 3, pp. 1805–1823, Jun. 2020.

[5] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. 6th Int. Conf. Comput. Vis.*, Jan. 1998, pp. 839–846.

[6] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, Aug. 2004.

[7] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, no. 3, p. 96, Jul. 2007.

[8] I. Eichhardt, D. Chetverikov, and Z. Janko, "Image-guided ToF depth upsampling: A survey," *Mach. Vis. Appl.*, vol. 28, nos. 3–4, pp. 267–282, 2017.

[9] K.-J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, Apr. 2006.

[10] Q. Yang, "Stereo matching using tree filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 4, pp. 834–846, Apr. 2015.

[11] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *Proc. IEEE Conf. (CVPR)*, Jun. 2011, pp. 3017–3024.

[12] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1397–1409, Jun. 2013.

[13] D. Sun, S. Roth, and M. J. Black, "Secrets of optical flow estimation and their principles," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 2432–2439.

[14] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time weighted median filtering for stereo matching and beyond," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 49–56.

[15] W. Wu, L. Li, and W. Jin, "Disparity refinement based on segment-tree and fast weighted median filter," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2016, pp. 3449–3453.

[16] Z. Shi, Y. Li, C. Zhang, M. Zhao, Y. Feng, and B. Jiang, "Weighted median guided filtering method for single image rain removal," *EURASIP J. Image Video Process.*, vol. 2018, no. 1, pp. 1–8, May 2018.

[17] Q. Zhang, L. Xu, and J. Jia, "100+ times faster weighted median filter (WMF)," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 2830–2837.

[18] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-27, no. 1, pp. 13–18, Feb. 1979.

[19] T.-S. Le, N.-T. Do, and K. Hamamoto, "Speed up temporal median filter and its application in background estimation," in *Proc. IEEE RIVF Int. Conf. Comput. Commun. Technol., Res., Innov., Vis. Future (RIVF)*, Nov. 2016, pp. 175–180.

[20] D. Cline, K. B. White, and P. K. Egbert, "Fast 8-Bit median filtering based on separability," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2007, p. 281.

[21] S. Perreault and P. Hebert, "Median filtering in constant time," *IEEE Trans. Image Process.*, vol. 16, no. 9, pp. 2389–2394, Sep. 2007.

[22] M. Storath and A. Weinmann, "Fast median filtering for phase or orientation data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 3, pp. 639–652, Mar. 2018.

[23] B. B. Chaudhuri, "An efficient algorithm for running window pel gray level ranking in 2-D images," *Pattern Recognit. Lett.*, vol. 11, no. 2, pp. 77–80, Feb. 1990.

[24] R. M. Sánchez and P. A. Rodríguez, "Bidimensional median filter for parallel computing architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 1549–1552.

[25] A. Adams, "Fast median filters using separable sorting networks," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–11, Aug. 2021.

[26] H. Zhao, D. Gao, M. Wang, and Z. Pan, "Real-time edge-aware weighted median filtering on the GPU," *Comput. Graph.*, vol. 61, pp. 11–18, Dec. 2016.

[27] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, no. 3, p. 103, Jul. 2007.

[28] B. I. Justusson, "Median filtering: Statistical properties," in *Two-Dimensional Digital Signal Prcessing II* (Topics in Applied Physics). Berlin, Germany: Springer, 1981, pp. 161–196.

[29] J. Nieweglowski, M. Gabbouj, and Y. Neuvo, "Weighted medians—Positive Boolean functions conversion algorithms," *Signal Process.*, vol. 34, no. 2, pp. 149–161, Nov. 1993.

[30] G. R. Arce, "A general weighted median filter structure admitting negative weights," *IEEE Trans. Signal Process.*, vol. 46, no. 12, pp. 3195–3205, Dec. 1998.

[31] V. Katkovnik, A. Foi, K. Egiazarian, and J. Astola, "From local kernel to nonlocal multiple-model image denoising," *Int. J. Comput. Vis.*, vol. 86, no. 1, pp. 1–32, Jul. 2009.

[32] N. Fukushima et al., "Efficient computational scheduling of box and Gaussian FIR filtering for CPU microarchitecture," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, Nov. 2018, pp. 875–879.

[33] B. Weiss, "Fast median and bilateral filtering," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 519–526, 2006.

[34] O. Green, "Efficient scalable median filtering using histogram-based operations," *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2217–2228, May 2018.

[35] K. Honauer, O. Johannsen, D. Kondermann, and B. Goldluecke, "A dataset and evaluation methodology for depth estimation on 4D light fields," in *Proc. Asian Conf. Comput. Vis.*, Taipei, Taiwan, 2016, pp. 19–34.

[36] K. Mishiba, "Fast depth estimation for light field cameras," *IEEE Trans. Image Process.*, vol. 29, pp. 4232–4242, 2020.

[37] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1122–1131.

[38] A. Clapes, J. C. S. J. Junior, C. Morral, and S. Escalera, "ChaLearn LAP 2020 challenge on identity-preserved human detection: Dataset and results," in *Proc. 15th IEEE Int. Conf. Autom. Face Gesture Recognit. (FG)*, Nov. 2020, pp. 801–808.

[39] *University of Pavia Dataset*. Accessed: Mar. 17, 2022. [Online]. Available: http://www.ehu.ews/ccwintoco/index.php?title=Hypespectral_Remote_Sensing_Scenes

Kazu Mishiba (Member, IEEE) received the B.E., M.E., and Ph.D. degrees from Keio University, Yokohama, Japan, in 2004, 2006, and 2011, respectively. In 2006, he joined FUJIFILM Company Ltd. He became an Assistant Professor at Keio University in 2011. He is currently an Associate Professor at Tottori University. His research interests include image processing and computer vision.