VECTOR ADDRESSING FOR NON-SEQUENTIAL SAMPLING IN FIR IMAGE FILTERING

Norishige Fukushima, Teppei Tsubokawa, and Yoshihiro Maeda

Nagoya Institute of Technology, Japan

ABSTRACT

Image filtering is fundamental in image processing. The acceleration is essential since image resolution has highly increased. For the acceleration, image subsampling is a general approach for any filtering. However, this approach has a drawback in accuracy due to image aliasing. Several papers moderate this problem by subsampling image or filtering kernel non-sequential sampling. In this paper, we improve the sampling combined with image and kernel subsampling. We also accelerate the work of non-sequential sampling by vector addressing of hardware acceleration. Experimental results show that the proposed method accelerate bilateral filtering and adaptive Gaussian filtering. Also, the proposed vector addressing accelerate both filters.

***Index Terms*—** randomized algorithm, randomized subsampling, edge-preserving filtering, vector addressing

1. INTRODUCTION

Filtering is fundamental tools for image processing. The filtering usually has $O(Sr^2)$ order, where S and r are image size and filtering radius, respectively. There are several approaches to reduce the order of filtering radius. Separable filtering reduces the order $O(r^2)$ into $O(r)$, and recursive filtering decrease it into $O(1)$.

Image subsampling accelerates general image processing. The subsampling reduces the order into $O(\log(Sr^2))$, and this method accelerates any image processing. The approach has low accuracy since subsampling loses high-frequency information. Only kernel subsampling moderates the drawback [1]. Also, randomized sampling suppresses signal aliasing. The filtering cost also depends on image size; thus, this method is limited to high-resolution images. Important image subsampling [2] performs filtering at notable parts in images, i.e., edge and texture. The method reduces the number of processing pixels. Then, the method interpolates other parts.

Hardware acceleration is also essential. Parallelization and vectorization are effective in CPU implementation. Parallelization is easily applied in image filtering, but vectorization depends on the kind of filters. Usually, we sequentially store and access images in memory. However, the approaches of [1] and [2] have non-sequential accessing for memory; thus, its vectorization is not trivial.

In this paper, we conjunction with the conventional approaches of [1] and [2] to improve the acceleration performance in high-resolution images. Also, the formulation is generated form bilateral filtering to general FIR filtering. Moreover, we propose an effective vectorization for these filtering with the non-sequential accessing.

This work was supported by JSPS KAKENHI (JP17H01764, JP18K19813).

2. RELATED WORK

Finite impulse response (FIR) filtering is defined as follows:

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = \frac{1}{\eta} \sum_{\boldsymbol{q} \in N(\boldsymbol{p})} f(\boldsymbol{p}, \boldsymbol{q}) \boldsymbol{I}(\boldsymbol{q}), \tag{1}$$

where $\boldsymbol{I}$, $\bar{\boldsymbol{I}}$ is an input image and an output image, respectively. $\boldsymbol{p}$, $\boldsymbol{q}$ is a target pixel and a reference pixel. f is a weight function. N represents a set of reference pixels in the kernel. η is a normalization term.

There are several techniques for accelerating the FIR filters. The order of separable filtering is $O(r)$. The kernel of the typical filters, e.g., box, Gaussian, Laplacian, and Sobel, have separability; thus, the approach accelerates filtering without approximation. For any spatially invariant filtering, we can construct separable kernels [3] through the singular value decomposition. For spatially invariant filtering, recursive filtering reduces the computational order into $O(1)$. Gaussian filtering and box filtering have optimized representations. Box filtering is accelerated by summed area table [4, 5]. Gaussian filtering are approximated by infinite impulse response (IIR) filtering [6, 7], iterative box filtering [8], and sliding DCT approximation [9, 10].

Additional processing is required for spatially variant filtering, such as adaptive parameter filtering and edge-preserving filtering. Spatially variant filtering is decomposed into several spatially invariant filters. Otherwise, the filter is forcefully applied separable filtering. For example, bilateral filtering is decomposed into multiple Gaussian filtering. There several approaches for this acceleration [11, 12, 13, 14, 15, 16]. These approximations accelerate the filter for grayscale images but do not work well for color images due to the curse of dimension. [17] moderates the problem, but the proceeding speed is real-time performance. Forcefully separable filtering [18, 19] does not have the curse of dimension. The order of these approaches is $O(r)$; thus, the acceleration is not suitable for large filtering kernel.

3. ACCELERATION WITH NON-SEQUENTIAL ACCESSING

3.1. Randomized Kernel Subsampling

Banterle et al. [1] subsample image kernels for accelerating bilateral filtering. The generalized representation for FIR filtering can be defined as follows:

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = \frac{1}{\eta} \sum_{n=1}^{|\boldsymbol{R}_{\boldsymbol{p}}|} f(\boldsymbol{p}, \boldsymbol{R}_{\boldsymbol{p}}(n)) \boldsymbol{I}(\boldsymbol{R}_{\boldsymbol{p}}(n)), \tag{2}$$

where $\boldsymbol{R}$ returns randomize reference pixels from $N(\boldsymbol{p})$, and $|\boldsymbol{R}|$ is the number of pixels to be referred. When $|\boldsymbol{R}| < N(\boldsymbol{p})$, the number of processing pixels in the kernel is reduced. Figure 1 shows

To appear in *Proc. IEEE International Conference on Image Processing (ICIP), 2019.*

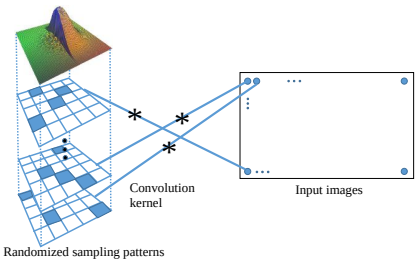


Fig. 1: Overview of randomized kernel subsampling.

an overview of this filter. We prepare random matrices for randomized sampling, and then we apply the sampling at filtering. We use Poisson-disk subsampling [20, 21] for kernel subsampling. Note that the method only subsample filtering kernel and the image part is full sample.

3.2. Importance Image Sampling

Lou et al.[2] convolve only important pixels. The method reduces the computational cost from image size into the number of processing pixels. The definition is shown in

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = M(\boldsymbol{p})\frac{1}{\eta}\sum_{\boldsymbol{q}\in N(\boldsymbol{p})} f(\boldsymbol{p},\boldsymbol{q})\boldsymbol{I}(\boldsymbol{q}), \tag{3}$$

where $M(\boldsymbol{p}) = m \in \{0, 1\}$ represents an importance map, which contains processing flags. If $M(\boldsymbol{p})$ is 0, the processing of the pixel is omitted.

The type of importance map depends on the type of image filtering. For edge-preserving filtering, edge information is essential. We use the textureness measure of Bae et al. [22] for this case. At first, we extract high frequency signals.

$$\boldsymbol{I}_h = |\boldsymbol{I} - \boldsymbol{G} * \boldsymbol{I}|, \tag{4}$$

where $\boldsymbol{G} * \boldsymbol{I}$ represents a Gaussian convoluted image. $|\cdot|$ shows the absolute function. Then, the high-pass image is also Gaussian convoluted.

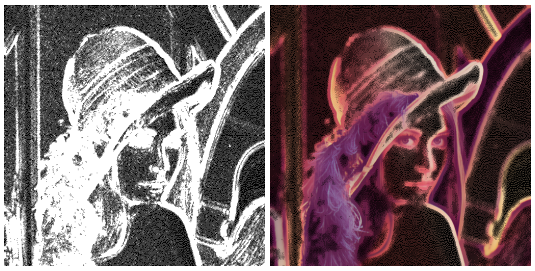
$$\bar{\boldsymbol{I}}_h = \boldsymbol{G} * \boldsymbol{I}_h. \tag{5}$$

In [22], joint bilateral filtering [23, 24] was used, but we use the Gaussian convolution for acceleration. Next, we remap the map to control the number of processing pixels.

$$\boldsymbol{M}' = \min(s \cdot \bar{\boldsymbol{I}}_h, v_{\max}), \tag{6}$$

where s amplify the value; thus, the number of important pixels becomes large, when s is high. Notice that $v_{\max}$ prevents overflow, e.g., $v_{\max} = 255$ for 8-bit and $v_{\max} = 1.0$ for floating number. For control the number of processing pixels by percentage, we can use the histogram of $\bar{I}_h$ instead of using the direct parameter of s. Finally, we binarize the grayscale importance map by dithering. For dithering, we use error diffusion of Floyd-Steinberg.

$$\boldsymbol{M} = D_{\text{FS}}(\boldsymbol{M}') \tag{7}$$



(a) Importance map (b) Resulting image

Fig. 2: Importance map and resulting image of subsampled bilateral filtering. 55% pixels are processed.

where D_{FS} shows the dithering function.

Figure 2 shows the importance map and the resulting image. We can process almost edge and texture parts. Next, we interpolate the gap pixels in the image. Weighted filtering can interpolate the value [25].

$$\bar{\boldsymbol{I}} = \frac{\boldsymbol{L} * \boldsymbol{I} \cdot \boldsymbol{M}}{\boldsymbol{L} * \boldsymbol{M}}, \tag{8}$$

where $\boldsymbol{L}$ is Laplacian smoothing convolution, and $\boldsymbol{M}$ is importance map. The Laplacian kernel is $\exp(-|x|/\sigma)$. The filter is represented by IIR filtering; thus, this interpolation is fast. The previous work [22] used Gaussian convolution for this interpolation, but Laplacian smoothing can perform more accelerate interpolation.

For non-edge-preserving filtering, such as parameter adaptive filtering of adaptive Gaussian filtering, we use Poisson-disk subsampling [20, 21] for importance map sampling. The approach is also used in patch-based filtering [26]. Note that the method only subsamples filtering images and the kernel part is full sample.

4. PROPOSED METHOD

4.1. Kernel and Image Subsampling

In this paper, we jointly consider Eqs. (2) and (3) for acceleration. The definition is as follows:

$$\bar{\boldsymbol{I}}(\boldsymbol{p}) = M(\boldsymbol{p})\frac{1}{\eta}\sum_{n=1}^{|\boldsymbol{R}_{\boldsymbol{p}}|} f(\boldsymbol{p}, \boldsymbol{R}_{\boldsymbol{p}}(n))\boldsymbol{I}(\boldsymbol{R}_{\boldsymbol{p}}(n)). \tag{9}$$

After filtering, we interpolate the filtering by Eq. (8).

Note that the simple downsampling approach has similarities to the proposed approach. Image processing is accelerated by processing pipeline, i.e., subsampling, filtering, and then upsampling. The pipeline decimates processing pixels in an image by a regular grid and reference pixels in a kernel by a regular grid, too. Therefore, M and R have grid patterns in Eq. (9). sub-up has ,however, relationship between M and R. On the contrary, we can separate the image and kernel sampling. This approach has grid sampling for image sampling; thus, we can use typical image upsampling, e.g., linear and cubic.

4.2. Vectorization

Image processing has five nested loops; x and y image loops, x and y kernel loops, and color loops. For vectorization, we usually use loop unrolling for the loops and then we vectorize the unrolled processes.

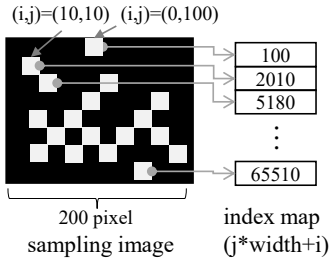


Fig. 3: Generating index map for image subsampling.

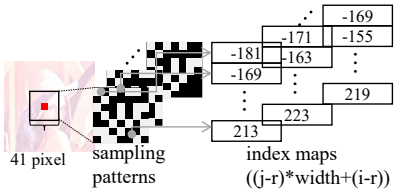


Fig. 4: Generating index maps of kernel subsampling.

Image-loop unrolling is usually better computational performance than the kernel-loop and color-loop one [27]. We can always vectorize the color-loop when a processing image has color channels. The unrolling has, however, low parallelization efficiency. Since the number of channels is 3, but the width of SIMD channels is 4, 8 or 16.

The proposed method and the conventional methods [1, 2] have non-sequential accesses in filtering. The vectorization usually requires sequential accessing. In the image subsampling [2], the kernel-loop access is sequential, but the image-loop access is not sequential. In the kernel subsampling [1] and the proposed method, both kernel-loop and image-loop accesses are not sequential. For non-sequential accessing, we can use *set* intrinsic, which is scaler loading. The operation is remarkably slower than the sequential load.

The problem of the non-sequential accessing is solved by vector addressing [28], such as *gather* and *scatter*. The vector addressing realizes vector loading and storing for non-sequential data. Currently, we can use the loading operation of *gather* from the 5-th generation of Intel Core i CPU. Also, we can use the storing operation of *scatter* from the 7-th generation of Intel Core i CPU, which is, however, limited in high-end CPU or Xeon. The operations require a head pointer of I/O memory and an index array, which contains relative address from the head for accessing pixel. The intrinsics are as follows;

- *_mm256_i32gather_ps(float* src, __m256i index, 4)*
- *_mm256_i32scatter_ps(float* dst, __m256i index, 4)*

src and *dst* are head pointers of the source and destination buffers. *index* is a partial index array loaded to the SIMD register. The last augment is $sizeof(float)$. We can vectorize arbitrary loop with the vector addressing.

4.2.1. non-sequential sampling for image subsampling

In image subsampling, the kernel-loop is natively sequential; thus, we can use sequential loading for this loop. The width of the kernel-loop is not multiples of the SIMD width, i.e., 4, 8, or 16, since the kernel width is usually an odd number. We usually take scaler processing for exception handling of the remainder loop. With the vector addressing, however, we can vectorize the peel loop.

The image-loop is not sequential in image subsampling. For this case, we generate an index map for image loop. Figure 3 shows the generation of the index map. The index map is generated from the importance map of Eq. (6). Gathering with the index map, we can unroll image-loop for vectorization.

4.2.2. non-sequential sampling for kernel sampling

In kernel subsampling, the kernel-loop is not sequential. We can take a similar approach of the image-loop in image subsampling for this case. First, we generate multiple index maps for kernel loops, since we should switch the random accessing patterns. Then, we randomly select the pattern and then gather with the index map for the unrolling kernel-loop. Figure 4 shows the generation of the index maps.

For the image-loop vectorization, the process needs an additional step. The index patterns are stored for kernel sampling, but we should switch the pattern per pixel along image-loop. Thus, we select the number of SIMD width patterns, i.e., eight patterns, and then we gather the first set of elements in the set of the patterns. Finally, we also gather from the second to the final elements in the patterns.

4.2.3. non-sequential sampling for proposed method

In image-and-kernel subsampling, the kernel-loop vectorization has the same process as the kernel subsampling. In the image-loop unrolling, we obtain the relative addresses for reference pixels by the same process as the kernel subsampling. In this process, the center pixels are not sequential; thus, we gather the address of the pixel by the same process as the image subsampling. Combined with both addressing, the proposed method is well vectorized.

4.2.4. scatter intrinsic for image sampling

When an image is non-structurally subsampled, data storing is also non-sequential. The scatter intrinsic is adequate for this case. However, the scatter is not supported by the most CPU; thus, we should use scalar accessing in unsupported CPUs.

5. EXPERIMENTAL RESULTS

We compared the vectorization method of the kernel-loop to that of image-loop for non-sequential addressing in image filtering. Then, we compared the proposed method to the image subsampling method [2] and kernel subsampling method [1] by the trade-off between computational performance and approximation accuracy. For the accuracy metric, we utilized peak signal to noise ratio (PSNR). We accelerated two types of filters; one is bilateral filtering [29], another is adaptive Gaussian filtering [30]. The bilateral weight is as follows:

$$f(\boldsymbol{p},\boldsymbol{q}) := \exp(\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{-2\sigma_s^2})\exp(\frac{\|\boldsymbol{I}(\boldsymbol{p})-\boldsymbol{I}(\boldsymbol{q})\|_2^2}{-2\sigma_r^2}), \quad (10)$$

where $\|\cdot\|_2$ is the L2 norm, and σ_r and σ_s are standard deviations for range and spatial distributions, respectively. The weight of adaptive Gaussian filtering is as follows:

$$f(\boldsymbol{p},\boldsymbol{q}) := \exp(\frac{\|\boldsymbol{p}-\boldsymbol{q}\|_2^2}{-2\sigma_s(\boldsymbol{p})^2}), \quad (11)$$

Table 1: Image subsampling: Ratio of computational time between kernel and image loop vectorization (image / kernel) w.r.t. sampling ratio of image subsampling.

sampling ratio	0.1	0.2	0.3	0.4	0.5
bilateral	1.10	1.14	1.18	1.18	1.19
adaptive Gaussian	1.37	1.50	1.56	1.57	1.59

Table 2: Kernel subsampling: Ratio of computational time between kernel and image loop vectorization (image / kernel) w.r.t. sampling ratio of kernel subsampling.

sampling ratio	0.1	0.2	0.3	0.4	0.5
bilateral	1.65	1.47	1.41	1.30	1.25
adaptive Gaussian	1.64	0.98	0.86	0.81	0.77

Table 3: Image and kernel subsampling: Ratio of computational time between panel and image loop vectorization (image / kernel) w.r.t. sampling ratio of kernel and image subsampling; Up: bilateral filter, Down: adaptive Gaussian filter.

kernel/image	0.1	0.2	0.3	0.4	0.5
0.1	1.95	2.79	3.31	3.60	3.94
	2.11	2.72	3.14	3.45	3.65
0.2	1.90	2.56	2.99	3.17	3.33
	1.94	2.43	2.68	2.80	3.00
0.3	1.82	2.31	2.65	2.82	2.91
	1.80	2.23	2.45	2.58	2.69
0.4	1.74	2.14	2.36	2.50	2.56
	1.70	2.03	2.18	2.26	2.34
0.5	1.66	1.98	2.19	2.29	2.35
	1.59	1.82	1.91	1.98	2.01

where $\sigma_s(\boldsymbol{p})$ is a pixel-dependent parameter found in a parameter map. For example, we change the parameter of the Gaussian distribution using a depth map [31, 32] as the parameter map for refocusing. $\sigma_s(\boldsymbol{p})$ is defined as:

$$\sigma_s(\boldsymbol{p}) = \sigma_s + \alpha \|d - \boldsymbol{D}(\boldsymbol{p})\|_1, \tag{12}$$

where $\boldsymbol{D}$ is the depth map, d is the focusing depth value, and α is a parameter of the range of the depth of field.

The code was written in C++ vectorized with AVX/AVX2, but for scatter intrinsic, we use 256-bit addressing supported from AVX512. OpenMP also parallelized the code. The used computer was Intel Core i9 7980XE, which is Skylake-X microarchitecture. The compiler was Visual Studio 2017. The computation of the weight functions are accelerated by considering subnormal numbers [33]. The size of the test image was the 2264×1512 color image for bilateral filtering and the 896×744 RGB-D image for adaptive Gaussian filtering. The kernel radius was $r = 24$ for the parameter of filtering.

Tables 1 to 3 show the ratio of the computational time of the kernel-loop and image-loop vectorization to the sampling ratio of image and kernel subsampling. In the image subsampling, the kernel-loop vectorization has faster performance. The kernel-loop vectorization is also faster in the kernel subsampling for bilateral filtering. On the contrary, the kernel-loop vectorization is faster for adaptive Gaussian filtering, since the filter requires a pixel-dependent parameter map and the result has low computational cost. In the proposed method, however, the image-loop vectorization is fast.

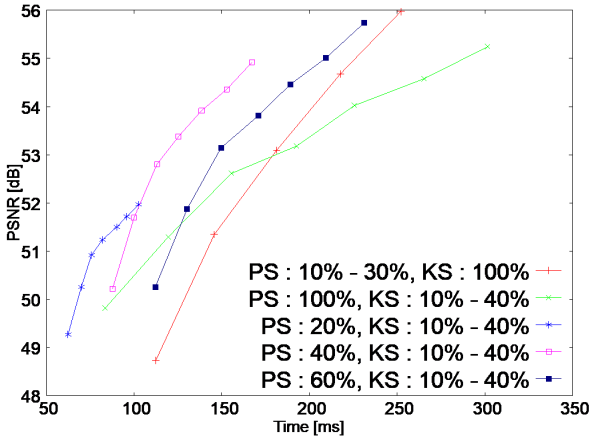


Fig. 5: Bilateral filtering: Trade-off between computational time and PSNR. PS indicates positional sampling ratio and KS indicates kernel sampling ratio.

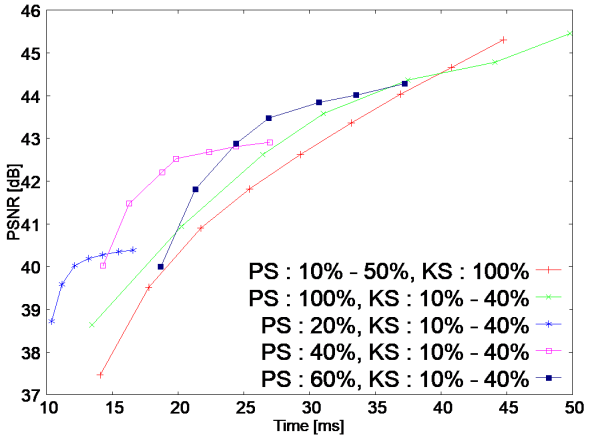


Fig. 6: Adaptive Gaussian filtering: Trade-off between computational time and PSNR. PS indicates positional sampling ratio and KS indicates kernel sampling ratio.

Figures 5 and 6 show the trade-off between the computational performance and the PSNR accuracy in the bilateral filtering and adaptive Gaussian filtering, respectively. The proposed method extracts high performance with less cost. For bilateral filtering, we can subsample the kernel samples more massive than the image samples. By contrast, we can subsample image samples more extensive than the kernel samples for adaptive Gaussian filtering. The fact shows that image subsampling is suitable when the filter has smooth convolution weights.

6. CONCLUSION

In this paper, we proposed an acceleration technique for image filtering by kernel and image subsampling. Also, we proposed a hardware acceleration technique by vector addressing of *gather*, *scatter* for the proposed method and also the conventional methods. Experimental results show that the proposed method more accelerate filters, such as bilateral filtering and adaptive Gaussian filtering, than the conventional method. Moreover, appropriate vector addressing is verified for the proposed and conventional methods.

7. REFERENCES

[1] F. Banterle, M. Corsini, P. Cignoni, and R. Scopigno, "A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain," *Computer Graphics Forum*, vol. 31, no. 1, pp. 19–32, 2012.

[2] L. Lou, P. Nguyen, J. Lawrence, and C. Barnes, "Image perforation:automatically accelerating image pipelines by intelligently skipping samples," *ACM Transactions on Graphics*, vol. 35, no. 153, 2016.

[3] S. Treitel and J. L. Shanks, "The design of multistage separable planar filters," *IEEE Transactions on Geoscience Electronics*, vol. 9, no. 1, pp. 10–27, 1971.

[4] F. C. Crow, "Summed-area tables for texture mapping," in *Proc. ACM SIGGRAPH*, 1984, pp. 207–212.

[5] N. Fukushima, Y. Maeda, Y. Kawasaki, M. Nakamura, , T. Tsumura, K. Sugimoto, and S. Kamata, "Efficient computational scheduling of box and gaussian fir filtering for cpu microarchitecture," in *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2018.*, 2018.

[6] R. Deriche, "Recursively implementating the gaussian and its derivatives," in *Proc. IEEE International Conference on Image Processing (ICIP)*, 1992, pp. 263–267.

[7] L. J. van Vliet, I. T. Young, and P. W. Verbeek, "Recursive gaussian derivative filters," in *Proc. IEEE International Conference on Pattern Recognition (ICPR)*, 1998.

[8] W. M. Wells, "Efficient synthesis of gaussian filters by cascaded uniform filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, , no. 2, pp. 234–239, 1986.

[9] K. Sugimoto and S. Kamata, "Fast gaussian filter with second-order shift property of dct-5," in *Proc. International Conference on Image Processing (ICIP)*, 2013.

[10] K. Sugimoto, S. Kyochi, and S. Kamata, "Universal approach for dct-based constant-time gaussian filter with moment preservation," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[11] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 257–266, 2002.

[12] F. Porikli, "Constant time o(1) bilateral filtering," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[13] Q. Yang, K. H. Tan, and N. Ahuja, "Real-time o (1) bilateral filtering," in *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 557–564.

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

[15] K. Sugimoto and S. Kamata, "Compressive bilateral filtering," *IEEE Trans. on Image Processing*, vol. 24, no. 11, pp. 3357–3369, 2015.

[16] S. Ghosh, P. Nair, and K. N. Chaudhury, "Optimized fourier bilateral filtering," *IEEE Signal Processing Letters*, vol. 25, no. 10, pp. 1555–1559, 2018.

[17] K. Sugimoto, N. Fukushima, and S. Kamata, "Fast bilateral filter for multichannel images via soft-assignment coding," in *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 2016.

[18] T. Q. Pham and L. J. V. Vliet, "Separable bilateral filtering for fast video preprocessing," in *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 2005.

[19] N. Fukushima, S. Fujita, and Y. Ishibashi, "Switching dual kernels for separable edge-preserving filtering," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[20] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Trans. on Graphics*, vol. 5, no. 1, pp. 51–72, 1986.

[21] D. Dunbar and G. Humphreys, "A spatial data structure for fast poisson-disk sample generation," *ACM Trans. on Graphics*, vol. 25, no. 3, pp. 503–508, 2006.

[22] S. Bae, S. Paris, and F. Durand, "Two-scale tone management for photographic look," *ACM Transactions on Graphics*, vol. 25, pp. 637–645, 2006.

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

[24] E. Eisemann and F. Durand, "Flash photography enhancement via intrinsic relighting," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 673–678, 2004.

[25] T. Matsuo, N. Fukushima, and Y. Ishibashi, "Weighted joint bilateral filter with slope depth compensation filter for depth map refinement," in *Proc. International Conference on Computer Vision Theory and Applications (VISAPP)*, 2013.

[26] S. Fujita, N. Fukushima, M. Kimura, and Y. Ishibashi, "Randomized redundant dct: Efficient denoising by using random subsampling of dct patches," in *Proc. SIGGRAPH Asia, Technical Briefs*, 2015.

[27] Y. Maeda, N. Fukushima, and H. Matsuo, "Taxonomy of vectorization patterns of programming for fir image filters using kernel subsampling and new one," *Applied Sciences*, vol. 8, no. 8, 2018.

[28] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-scatter dram: in-dram address translation to improve the spatial locality of non-unit strided accesses," in *Proc. International Symposium on Microarchitecture*, 2015.

[29] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1998.

[30] G. Deng and L. W. Cahill, "An adaptive gaussian filter for noise reduction and edge detection," in *Proc. IEEE Nuclear Science Symposium and Medical Imaging Conference*, 1993.

[31] S. Bae and F. Durand, "Defocus magnification," *Computer Graphics Forum*, vol. 26, no. 3, pp. 571–579, 2007.

[32] W. Zhang and W.-K. Cham, "Single image focus editing," in *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2009.

[33] Y. Maeda, N. Fukushima, and H. Matsuo, "Effective implementation of edge-preserving filtering on cpu microarchitectures," *Applied Sciences*, vol. 8, no. 10, 2018.