Efficient Computational Scheduling of Box and Gaussian FIR Filtering for CPU Microarchitecture

Norishige Fukushima*, Yoshihiro Maeda*, Yuki Kawasaki*, Masahiro Nakamura*,
Tomoaki Tsumura*, Kenjiro Sugimoto† and Sei-ichiro Kamata†
* Nagoya Institute of Technology, Japan
† Waseda University, Japan

***Abstract*—In this paper, we propose efficient computational scheduling of box and Gaussian filtering. These filters are fundamental tools and used for various applications. The computational order of the naïve implementations of these FIR filters are $O(r^2)$, where r is the kernel radius. A separable implementation reduces the order into $O(r)$ but requires twice times of filtering. A recursive representation dramatically sheds the order into $O(1)$ but also needs twice or more times filtering. The efficient representation curtails the number of arithmetic operations; however, the influence of data I/O for the computational time becomes dominant. In this paper, we optimize the computational scheduling of $O(1)$ box and Gaussian filters to competently utilize cache memory for reducing the computational time of data I/O. Experimental results show that the proposed scheduling has higher computational performance than the conventional implementation.**

I. Introduction

Box and Gaussian filters are essential tools in image processing. Applications of these filters are widely used in not only simple smoothing, but stereo matching [1], feature description, e.g., SIFT [2] and SURF [3], saliency map [4], and image quality metrics of SSIM [5]. These fundamental filters are further utilized for advanced filtering of edge-preserving filtering, such as accelerations of bilateral filtering [6], [7], [8], [9], [10], [11], [12], guided image filtering [13] and its variants [14], [15], [16], [17], [18], [19], [20], [21], and domain transform filtering [22]. In addition, the filters are used in the application of edge-preserving filtering, such as high dynamic range imageing [7], haze removing [23], free viewpoint view synthesis [24], and advanced stereo matching/optical flow estimations [25].

The box and Gaussian filters are 2D finite impulse response (FIR) filters. The computational order is $O(r^2)$, where r is the kernel radius. Computation of these filters has some practical solutions. A traditional approach is a separable implementation. The 2D filters are decomposed into a combination of 1D FIR filters, and its computational order is $O(r)$. A recursive representation of the box [26] and Gaussian [27], [28] filters further reduce the computational order, $O(1)$, i.e., the computational time of the filtering does not depend on the kernel radius. The efficient representation curtails the number of arithmetic operations; consequently, the influence of data I/O for the computational time becomes dominant.

This work was supported by JSPS KAKENHI (JP17H01764, 18K18076).

The capability of arithmetic operations and data I/O depends on computer architectures. Currently, domain-specific programming languages, such as Halide [29], [30] and Darkroom [31], focuses on optimizing computational scheduling in image processing for more complexing computing architectures. The languages effectively deal with how to compute data with parallelization, vectorization, and computing sequences. For complex computational scheduling, the language cannot handle well; hence, hand turning is required, yet. For example, appropriate turning extracts more computational performance in the FIR filtering case [32].

In this paper, we propose efficient computational scheduling of the box and Gaussian filters for CPUs. Moreover, we focus the case of multi-channel images, such as color/hyperspectral image, and high dimensional feature data. These filters have not dependencies in channels; thus, we can vectorize and parallelize processing for channel domains less overhead. Note that multichannel filtering requires many channels for parallelization, and the multichannel filtering has a low spatial locality in data accesses; therefore, the trade-off between parallelisms in the channel and spatial domains is essential.

Note that there are several efficient algorithms of the box and Gaussian filters for GPUs [33], [34], [35], [36]; however, these algorithms are aimed for massively parallel computing that requires hundreds of cores.

II. Conventional Work

A. Box filtering and its computational scheduling

There are several implementations of box filtering, such as naïve convolution, separable convolution, integral image, and summed area table. Table I summarizes each implementation with the proposed method described in Sec. 3.

In the traditional FIR convolution, whose computational order is $O(r^2)$, there is no limitation in vectorization and parallelization. The definition is as follows;

$$O(x,y) = \sum_{i=-r}^{r}\sum_{j=-r}^{r} I(x+i, y+j), \tag{1}$$

where $I(x,y), O(x,y)$ are input and output images on pixel coordinates x, y, respectively. The output O is normalized as follows;

$$O'(x,y) = \frac{1}{(2r+1)^2} O(x,y). \tag{2}$$

To appear in *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2018.*

TABLE I: Comparison of each implementation of box and Gaussian filtering. (S) and (C) indicate spatial and channel domain, respectively. Bold fonts imply the proposed scheduling.

	Parallelization (S)	vectorization (S)	Parallelization (C)	vectorization (C)	No. Stage	Order
2D FIR (Both)	unlimited	unlimited	unlimited	unlimited	1	$O(r^2)$
Sep. 1D FIR (Both)	unlimited	unlimited	unlimited	unlimited	2	$O(r)$
Integral (Box)	only 2nd stage	only 2nd stage	unlimited	unlimited	2	$O(1)$
SSAT (Box)	unlimited, but noticeable overhead	vertical: unlimited horizontal: inefficient	unlimited	unlimited	2	$O(1)$
OP-SAT (Box)	redundant processing	impossible	unlimited	unlimited	1	$O(1)$
Sliding (Gauss)	unlimited, but noticeable overhead	vertical: unlimited horizontal: inefficient	unlimited	unlimited	2	$O(1)$
OP-Sliding (Gauss)	redundant processing	impossible	unlimited	unlimited	1	$O(1)$

One multiplication operation is only required for this operation, since the normalization value is constant. The other implementations also perform this normalization process.

The separable implementation, whose order is $O(r)$, is faster than the naïve. The scheduling decomposes a 2D filter into two 1D filters. The implementation of the separable FIR filtering, which is vertical and then horizontal filtering, is as follows;

$$J(x,y)=\sum_{i=-r}^{r} I(x,y+i), \quad O(x,y)=\sum_{j=-r}^{r} J(x+j,y), \quad (3)$$

where J is an intermediate image. The separable FIR requires two stages, but tiling and interleaving of the two stages maximize the cache efficiency. Furthermore, the tiling and interleaving do not take much overhead. Therefore, this traditional method could be the fastest method in the small kernel radius case.

Integral image processing is $O(1)$ and requires two-stage processing, which are generating integral images and are smoothing images. The first stage generates an integral image II, and then the second stage smooths an output by using the integral image.

$$II(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(x,y), \quad (4)$$

$$\begin{aligned} O(x,y) = {} & II(x+r,y+r) - II(x-r-1,y+r) \\ & - II(x+r,y-r-1) + II(x-r-1,y-r-1). \end{aligned} \quad (5)$$

The first stage cannot be vectorized and needs redundant processing for parallelization. The second stage, conversely, can be fully vectorized and parallelized; thus, the first stage is a bottleneck. The integral image can adaptively change filtering radius per pixel. This characteristic has an advantage for specific applications, e.g., face detection; however, the filtering speed is not high in term of simple smoothing.

Separable summed area table (*SSAT*) is also $O(1)$ computing and also requires two stages, which include horizontal and vertical filtering. SSAT is defined as follows;

$$J(x,y)=J(x,y-1)+I(x,y+r)-I(x,y-r-1), \quad (6)$$

$$O(x,y)=O(x-1,y)+J(x+r,y)-J(x-r-1,y). \quad (7)$$

The recursive representation extinguishes the summation, and then the filtering consists two additions and two subtractions. The algorithm has a dependency in the computational pixel order; hence, there are some limitations in parallelization and vectorization. The vertical filtering can be vectorized without limitation and be parallelized along the raw order. The horizontal filtering can be parallelized along the column order but has limitations in vectorization [37]. For vectorization, we should transpose overall images or partial-block images; however, the transposing operation itself is cost-consuming. The other solution is loop unrolling vertical direction and then gathering vertical pixels. The gathering cost is higher than the usual loading operations, but this method has the lighter cost than the transpose. Further, the computational sequence of separable filters of horizontal and vertical direction is also essential for cache efficiency. The parallel granularity of both horizontal and vertical filters are fine grain; therefore, the overhead of parallelization is large. More cache efficient solutions, e.g., tiling and interleaving, require redundant computation.

Note that there is no limitation for vectorization and parallelization of each scheduling of box filtering in channel domains.

B. Gaussian filtering and its computational scheduling

In Gaussian filtering, there is the similar implementation of box filtering, such as naïve and separable convolutions and sliding implementation, which is similar to SSAT. Also, Gaussian filtering has the other efficient solutions. FFT (fast Furrier transform) is a traditional solution. The order is $O(S \log S)$, where S is image size. IIR (infinite impulse response) representation is the more efficient solution. The order is $O(1)$; however, the IIR requires twice or more image scans for each separable direction; hence, the 2D image convolution requires four image scans at least.

The sliding DCT based approaches [27], [28], [38] reduce the scanning times with approximating Gaussian FIR filtering. We denote this approach *Sliding*. The Gaussian filter can be approximated in the summation of DCT convolutions, $O = \sum_k^n C_k * I$, where C_k is the k-th DCT convolution kernel and n is the number of coefficients. The DCT convolution has a recursive representation introduced by the second-order shift property. Let be $C * I = F$;

$$F_k(x,y+1) = 2a_1F_k(x,y) - F_k(x,y-1) + a_r\delta(x,y), \quad (8)$$

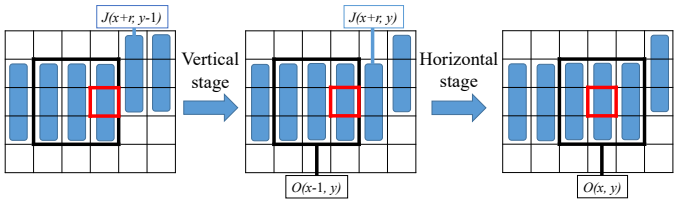
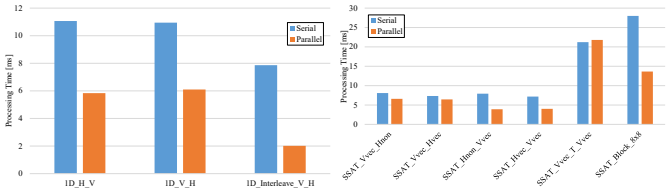


Fig. 1: Computational scheduling of OP-SAT.



(a) Separable FIR (b) SSAT

Fig. 2: Computational time of separable FIR convolution and SSAT with/without parallelization. "H_V" and "V_H" indicate horizontal filtering and then vertical one, and its reverse order. "interleave_V_H" means interleaving vertical filtering and then horizontal one per each scan line. "vec" and "non" means applying vectorization or not and "T" means transposed operation and "Block_8x8" means block transposing.

where $a_{1,r}$ are DCT coefficients, and δ is defined by;

$$\delta(x,y) = I(x,y-r) + I(x,y+r). \tag{9}$$

This is the DCT-III based definition. The other representations of the DCT I, II, and IV are shown in [28]. The DCT convoluted output is summed to a horizontal filtering result, $J(x,y) = \sum_k^n F_k(x,y)$. Then vertical filtering is as defined in the same manner;

$$G_k(x+1,y) = 2a_1 G_k(x,y) - G_k(x-1,y) + a_r\delta(x,y), \tag{10}$$

$$\delta(x,y) = J(x-r,y) + J(x+r,y). \tag{11}$$

Finally, we sum up the outputs of vertical DCT filtering,

$$O(x,y) = \sum_k^n G_k(x,y). \tag{12}$$

III. Proposed Computational Scheduling

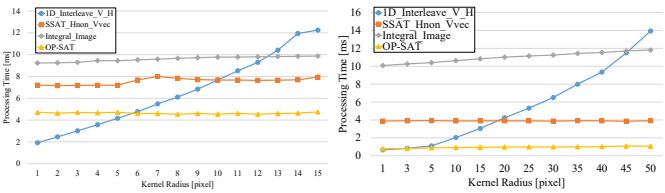
We introduce an extension of SSAT, which reduces the number of scanning stages to one. Then, we extend this approach for sliding Gaussian filtering.

A. Box Filtering

We propose computational scheduling for the box filter, where we merged the two stages in SSAT to one. We call the approach, one pass summed area table (*OP-SAT*). OP-SAT interleaves the computational order of horizontal and vertical filters per a pixel. The scheduling is represented as;

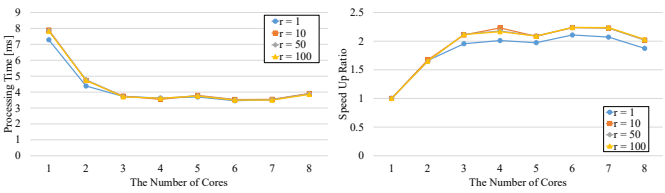
$$J(x+r,y) = J(x+r,y-1) + I(x+r,y+r) - I(x+r,y-r-1), \tag{13}$$

$$O(x,y) = O(x-1,y) + J(x+r,y) - J(x-r-1,y). \tag{14}$$



(a) Serial (b) Parallel

Fig. 3: Computational time of Box Filtering w.r.t. kernel radii (r) with/without parallelization.



(a) Processing time (b) Parallel efficiency

Fig. 4: Processing time and parallel efficiency w.r.t. the number of cores. (SSAT_Hnon_Vvec).

Fig. 1 visually demonstrates the scheduling. Let compute a 3×3 filter centering a red rectangle pixel (x,y). Blue rectangles show already averaged 1×3 parts. At first, we compute the right 1×3 mean of $J(x+r,y)$ in (12). Then, we compute the output by adding $J(x+r,y)$ and subtracting the oldest vertical averaging in (13). This scheduling can compute the average by two additions and two subtractions for summation and one multiplication for normalization. Moreover, the proposed scheduling can perform filtering with one stage by the raster scan order. Therefore, the method is more cache efficient.

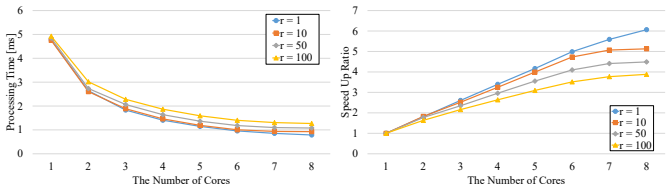
Note that this approach cannot vectorize the filter in the spatial domain; however, this method is cache efficient and can vectorize pixels in the channel domain. Moreover, this approach can spatially parallelize filtering by splitting overlapped image with redundant computation. Besides, the parallelization is coarse-grained; thus, the parallelized efficiency is higher than SSAT.

B. Gaussian Filtering

There are box-averaging-like parts in the representation of sliding DCT for Gaussian filtering. In this paper, we use the proposed one pass scheduling for the sliding approach, which is named *OP-Sliding*. In (11), we need $J(x+r,y)$; however, this part is not computed in the raster scan order; thus, we compute this at first and then compute (10);

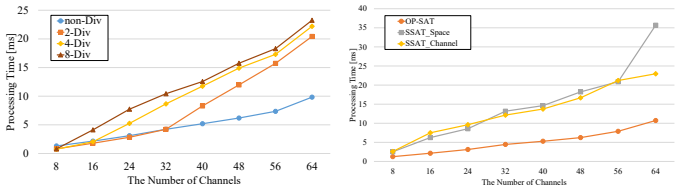
$$J(x+r,y) = \sum_k^n 2a_1 F_k(x,y-1) - F_k(x,y-2) + a_r\delta(x,y-1). \tag{15}$$

This is the almost same process of (13).



(a) Processing time (b) Parallel efficiency

Fig. 5: Processing time and parallel efficiency w.r.t. the number of cores. (OP-SAT).



(a) Parallerization trade-off (b) Comparison of each method

Fig. 6: Parallelization trade-off between spatial and channel domain and comparison results with SSAT with spacial or channel parallerization.

IV. Experimental Result

A. Box Filtering

We compared each scheduling of box filtering at first. Filtering radius was $r = 10$, and image size was 1920×1080. We used single floating values for each implementation and computed by Intel Core i7 6700 (4 cores, 8 threads) with Visual Studio 2015. The code was optimized by OpenMP and AVX/AVX2.

Firstly, we compared each separable FIR for grayscale images. In this case, we can apply parallelization and vectorization for spatial domain. Fig. 2a shows the computational time of the separable FIR with various computational orders with/without parallelization. Note that each implementation is vectorized. The result shows that interleaving implementation is the fastest, and each method has high parallelizability.

Next, we compared each SSAT implementation in Fig. 2b. Horizontal and then vertical filtering is the fastest due to the cache efficiency. Note that the horizontal vectorization did not affect. Also, transposed approaches have much overhead; therefore, these are slower than non vectorized solutions.

Then, we compared the fastest separable FIR and SSAT, integral image, and the proposed scheduling in Fig. 3. In this experiment, we changed the kernel radius. In the serial processing, the separable FIR is the fastest until 11×11 $(r = 5)$. In the parallel processing, the separable FIR is the fastest until 3×3 $(r = 1)$. In the other case, OP-SAT is the fastest. OP-SAT cannot be vectorized, but the method is faster than SSAT, which is known as the most efficient implementation.

We analyzed, additionally, parallelization efficiency for SSAT and OP-SAT in Figs. 4a, 4b with several filtering radius cases $(r = 1, 10, 50, 100)$. OP-SAT has higher parallel efficiency than SSAT. OP-SAT increases overheads concerning the size of the kernel radius, while SSAT is almost flat. Focusing the actual computing time, however, the proposed method is still faster than SSAT, even if we set the large kernel.

Next, we compared the multi-channel case. We set filtering radius $r = 10$, and image size was 512×512. OP-SAT can parallelize spatial and channel domains; thus, we optimize the trade-off between the parallelization. Note that vectorization is utilized for channel domain. We used a CPU with 8 threads and compared four dividing cases. "non-div" means no spatial parallelization, but parallelizing channels, and "2-div" means spatially parallelizing with two threads and the other is used for channels. "4-div" means the spatially four threads case, and "8-div" means all threads are used for spatial parallelization. Fig. 6 shows the result of the trade-off. For small channel cases, spatial parallelization is adequate; however, channel parallelization is efficient in many channel cases.

Finally, we compared the proposed method with SSAT of the channel and spatial parallelization in Fig. 6b. The proposed method was the optimized version of the parallelization result in Fig. 6a. Note that integral image and separable FIR implementations were omitted because these implementations are slower than SSAT and OP-SAT. The proposed method is faster than the both SSAT parallelized implementations.

B. Gaussian Filtering

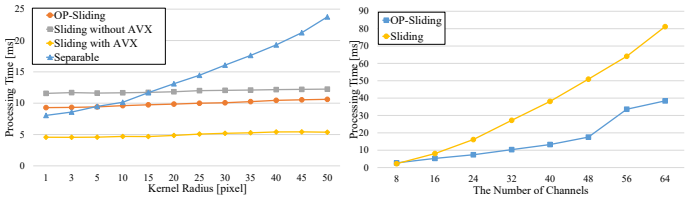
For Gaussian filtering, we compared with separable FIR, recursive separable representation, and the proposed method of one pass representation. We selected representative experimental results for Gaussian filtering. We used a grayscale image (1920×1080), and a multi-channel image (512×512).

Fig. 7a shows computational time to the filtering radius in a grayscale image. Note that all implementations were parallelized, and separable FIR was vectorized. Vectorized sliding is the fastest because OP-Sliding cannot be vectorized. Comparing with SSAT box filtering, sliding Gaussian has higher computational intensity; thus, vectorizing spatial domain is more effective. Looking back at the box case, vectorizing SSAT has a minor effect. The proposed scheduling is faster than the non-vectorized sliding; hence, OP-Sliding is more cache efficient than the conventional sliding approach, and OP-Sliding has a room of vectorization unit.

Fig. 7b shows computational time to the number of channels in a multi-channel image. Parallelization is performed in channel domain for OP-Sliding and spatial domain for Sliding, and all method are vectorized in the channel domain. The proposed scheduling can utilize vectorization units for the channel domain and more cache efficient scheduling; therefore, OP-Sliding is faster than the conventional one. In this experiment, we omitted the separable filtering result, since this approach is not fast.

V. Conclusion

In this paper, we proposed effective computational scheduling for FIR filtering of the box and Gaussian filtering for



(a) Grayscale (b) Color

Fig. 7: Processing time of grayscale and multi-channel Gaussian filtering. For the grayspace case, OP-Sliding is not vectorized, and the others are vectorized. For the color case, the radius is constant ($r = 16$), and the filters are vectorized in channel domain. $n = 1$ for sliding DCT.

current CPU micro-architectures. The proposed scheduling optimizes $O(1)$ FIR filters to reduce the number of filtering stages into one. This approach makes the filters more cache efficient and speeds up the box and Gaussian filters in gray and multi-channel images.

REFERENCES

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[4] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[5] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

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

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

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

[9] K. N. Chaudhury, "Constant-time filtering using shiftable kernels," *IEEE Signal Processing Letters*, vol. 18, no. 11, pp. 651–654, 2011.

[10] 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.

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

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

[13] K. He, J. Shun, and X. Tang, "Guided image filtering," in *European Conference on Computer Vision (ECCV)*, 2010.

[14] J. Lu, K. Shi, D. Min, L. Lin, and M. N. Do, "Cross-based local multipoint filtering," in *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] X. Tan, C. Sun, and T. D. Pham, "Multipoint filtering with local polynomial approximation and range guidance," in *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[16] L. Dai, M. Yuan, F. Zhang, and X. Zhang, "Fully connected guided image filtering," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[17] L. Dai, M. Yuan, Z. Li, X. Zhang, and J. Tang, "Hardware-efficient guided image filtering for multi-label problem," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] S. Fujita and N. Fukushima, "High-dimensional guided image filtering," in *International Conference on Computer Vision Theory and Applications (VISAPP)*, 2016, pp. 27–34.

[19] S. Fujita and N. Fukushima, *Extending Guided Image Filtering for High-Dimensional Signals*, vol. 693, pp. 439–453, Springer International Publishing, 2017.

[20] N. Fukushima, K. Sugimoto, and S. Kamata, "Guided image filtering with arbitrary window function," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[21] Y. Murooka, Y. Maeda, M. Nakamura, T. Sasaki, and N. Fukushima, "Principal component analysis for acceleration of color guided image filtering," in *International Workshop on Frontiers of Computer Vision (IW-FCV)*, 2018.

[22] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Transactions on Graphics*, vol. 30, no. 4, 2011.

[23] K. He, J. Sun, and X. Tang, "Single image haze removal using dark channel prior," in *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[24] N. Kodera, N. Fukushima, and Y. Ishibashi, "Filter based alpha matting for depth image based rendering," in *IEEE Visual Communications and Image Processing (VCIP)*, 2013.

[25] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 504–511, 2013.

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

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

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

[29] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[30] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[31] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan, "Darkroom: compiling high-level image processing code into hardware pipelines.," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 144–1, 2014.

[32] 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.

[33] D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe, "Gpu-efficient recursive filtering and summed-area tables," in *ACM Transactions on Graphics*, 2011, vol. 30, p. 176.

[34] A. Kasagi, K. Nakano, and Y. Ito, "Parallel algorithms for the summed area table on the asynchronous hierarchical memory machine, with gpu implementations," in *International Conference on Parallel Processing (ICPP)*, 2014, pp. 251–260.

[35] G. Chaurasia, J. R. Kelley, S. Paris, G. Drettakis, and F. Durand, "Compiling high performance recursive filters," in *High-Performance Graphics*, 2015, pp. 85–94.

[36] D. Nehab and A. Maximo, "Parallel recursive filtering of infinite input extensions," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 204, 2016.

[37] M. Nakamura and N. Fukushima, "Fast implementation of box filtering," in *International Workshop on Advanced Image Technology (IWAIT)*, 2017.

[38] V. Kober, "Fast algorithms for the computation of sliding discrete sinusoidal transforms," *IEEE Transactions on Signal Processing*, vol. 52, no. 6, pp. 1704–1710, 2004.