GPU-friendly Approximate Bilateral Filter for 3D Volume Data

Koichi Yano*, Kenjiro Sugimoto* and Sei-ichiro Kamata*
* Waseda University, Fukuoka, Japan
E-mail: {k1_yano@ruri, ksugimoto@aoni}.waseda.jp

***Abstract*—This paper presents an approximate Bilateral Filter(BF) with a GPU-friendly architecture for 3D volume data. The bilateral filter (BF) for 3D volume data such as medical images highly costs due to an enormous number of voxels to be processed. Existing acceleration methods called constant-time, or $O(1)$, BF are inappropriate for GPU processing because they consist of a combination of $O(1)$ spatial filters not to fit to parallel processing. The proposed method realizes a fast approximation 3D-BF by focusing two points: (1) the BF is decomposed into multiple Gaussian Filters and (2) GPU processing is suitable for convolution. As a consequence, proposed method achieved fast and high approximate accuracy in various window size.**

I. Introduction

The Bilateral Filter (BF), named by Tomasi and Manduchi [1] and also proposed as SUSAN [2] or non-linear Gaussian filter [3], is a fundamental method for edge-preserving smoothing in image processing. The BF is applied to denoising [4], stereo matching [5] and medical image processing [6]. Since the BF highly costs as compared with linear filters, it is an important task for many applications to accelerate the BF.

3D volume data have been widely used recent years in 3D medical imaging and 3D data modeling. Since volume data generally contain some noise such as Fig.1, it is demanded to use the BF for denoising volume data. As a major difficulty, 3D-BF highly costs due to an enormous number of voxels and the computational complexity increased with the number of dimensions. Specifically, if the BF has window length n, the computational complexity per pixel/voxel increases from n^2 in 2D to n^3 in 3D. In fact, the state-of-the-art constant-time ($O(1)$) BF [6], [7] requires tens of seconds for processing 3D data. Here, the order $O(1)$ indicates that processing time per pixel/voxel does not depend on window length but still depends on the number of dimensions. Hence, we desire an accelerated BF for 3D volume data.

GPU accelerates image processing by calculating all pixels in parallel. GPU based approach would be also effective to accelerate 3D-BF; however, most existing GPU implementations of the BF have focused on 2D images, not 3D volume data. The efficient GPU implementation of the naive BF [8] cannot be applicable to 3D case because of the limitation of memory size. The approximation method for GPU [9] targeted 2D case and takes time for achieving high approximate accuracy. Since the 3D-BF using GPU [10] discussed denoising accuracy and acceleration with parameter adjustment, it did not care about change of window size. Moreover, the existing $O(1)$ BF does not fit to the architecture of GPU processing due to the algorithm structure of computing pixels sequentially. Thus, although 3D-BF on GPU seems to show its high potential, it has not been explored in detail yet in the literature.

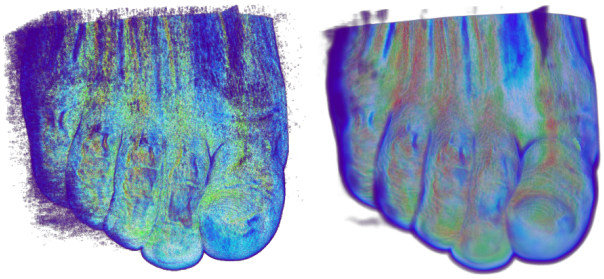


Fig. 1. A 3D medical image [6] and the denoised image that shows denoising performance of BF.

This paper presents an approximate BF with a GPU-friendly architecture for 3D volume data. The $O(1)$ BF has poor parallelism due to consisting the $O(1)$ GFs. On the other hand, Gaussian convolution is suitable for parallel processing. In the proposed method, $O(1)$ GF is calculated as naive convolution by using GPU. The proposed method needs to calculate multiple GFs; however, the processing time is faster than naive BF implemented in GPU since it is suitable for use with GPU. As a result, the proposed method was tens of times faster than the GPU implementation of naive BF.

II. Related work

A. Bilateral Filter

The BF [1]–[3] is a generalization of the Gaussian filter (GF) that is a target pixel/voxel is additionally weighted by luminance values around the current pixel/voxel and then normalized. Let $\boldsymbol{p} \in S$ be a voxel coordinate in the 3D image area $S \subset \mathbb{Z}^3$ and $f_{\boldsymbol{p}} \in \mathbb{R}$ its voxel value. The BF is defined by

$$\hat{f_{\boldsymbol{p}}} := \frac{\sum_{\boldsymbol{q} \in S} g_s(\boldsymbol{p}, \boldsymbol{q})\, g_r(f_{\boldsymbol{p}}, f_{\boldsymbol{q}})\, f_{\boldsymbol{q}}}{\sum_{\boldsymbol{q} \in S} g_s(\boldsymbol{p}, \boldsymbol{q})\, g_r(f_{\boldsymbol{p}}, f_{\boldsymbol{q}})}, \tag{1}$$

where $g_s(\cdot)$ and $g_r(\cdot)$ denote weight functions called spatial kernel and range kernel, respectively. The most widely-used spatial and range kernels are the Gaussian kernel

$$g_s(\boldsymbol{p}, \boldsymbol{q}) = e^{\frac{-||\boldsymbol{q}-\boldsymbol{p}||_2^2}{2\sigma_s^2}}, \qquad g_r(a, b) = e^{\frac{-(b-a)^2}{2\sigma_r^2}}, \tag{2}$$

where $\sigma_s > 0$ and $\sigma_r > 0$ are parameters called *spatial scale* and *range scale*, respectively. The computational complexity of the BF depends on the window area supported by spatial kernel. Gaussian spatial kernel is used, the window area is truncated at a position where the kernel is attenuated sufficiently. We calculate the 3D window area by $(2\lceil 3\sigma_s \rceil + 1)^3$.

B. Constant-time bilateral filter

The constant-time ($O(1)$) BF [6], [7], [11] is an approximate BF whose computational complexity is $O(1)$ per pixel/voxel. Most $O(1)$ BF is approximated by an appropriate combination of multiple GFs where each GF is implemented by $O(1)$ GF [12]–[16]. This framework can be summarized as follows: Let us consider approximating range kernel by the separable form

$$g_r(a,b) \approx \sum_{k=0}^{K-1} \phi_k(a)\psi_k(b), \tag{3}$$

and then, by substituting (3) for (1), we obtain

$$\hat{f}_{\boldsymbol{p}} \approx \frac{\sum_{k=0}^{K-1} \phi_k(f_{\boldsymbol{p}}) \sum_{\boldsymbol{q}\in S} g_s(\boldsymbol{p},\boldsymbol{q})\{\psi_k(f_{\boldsymbol{q}}) f_{\boldsymbol{q}}\}}{\sum_{k=0}^{K-1} \phi_k(f_{\boldsymbol{p}}) \sum_{\boldsymbol{q}\in S} g_s(\boldsymbol{p},\boldsymbol{q})\{\psi_k(f_{\boldsymbol{q}})\}}. \tag{4}$$

In (4), each summation $\sum_{\boldsymbol{q}\in S}$ can be interpreted as applying GF to an image since the terms $\{\cdot\}$ indicate intermediate images generated from the input image. Thus, the BF is decomposed into $2K$ of GFs in total of numerator and denominator. Because of separability of Gaussian spatial kernel (2), a D-dimensional GF can be separated into D of 1D-GFs. As a whole, the $O(1)$ BF has computational complexity of $O(K)$ per pixel/voxel.

For example, the Compressive BF [7] is a state-of-the-art $O(1)$ BF that decomposes Gaussian range kernel into cosine functions. Since Gaussian range kernel is an even function whose spectrum attenuates exponentially, it is well approximated by a sum of a few cosine terms as

$$g_r(a,b) \approx \sum_{m=0}^{M} \alpha_m \cos\left(\omega_0 m\,(a-b)\right) \tag{5}$$

where $\omega_0 = 2\pi/T$ indicates a basic angular frequency, T is cycle length and α_m is Fourier coefficient, which are obtained by optimization. Using addition theorem in (5),

$$\begin{aligned} g_r(a,b) \approx & \sum_{m=0}^{M} \alpha_m \cos\left(\omega_0 m a\right)\cos\left(\omega_0 m b\right) \\ & + \sum_{m=1}^{M} \alpha_m \sin\left(\omega_0 m a\right)\sin\left(\omega_0 m b\right). \end{aligned} \tag{6}$$

Comparing (6) with (3), we find out that $\phi_k(a)$ corresponds to $\alpha_m \cos(\omega_0 m a)$ and $\alpha_m \sin(\omega_0 m a)$ and that $\psi_k(b)$ corresponds to $\cos(\omega_0 m b)$ and $\sin(\omega_0 m b)$. Thus, the Compressive BF consists of $4M+1$ convolutions.

III. Proposed method

A. Analysis

The naive BF has the following problems in GPU implementation. For accelerating GPU computation, it is important to reduce memory access cost by aggressively using the shared memory (SM) on GPU. For example, an operation such as convolution or the BF accesses the same voxel many times. We can reduce access cost to voxel data by transferring in advance a subimage of an input image onto the SM. However, the size of SM is generally small as compared with global memory on GPU, e.g., just 48 KB (=12,288 floating-point numbers) in a relatively-new GPU. This limits the size of a subimage to 23^3 ($= 12167 < 12288$) voxels in 3D case. Since the window size has to be smaller than the subimage, the above implementation method works well only when $\sigma_s < 3$. In 2D-BF, Jonas [8] accelerated it by choosing an appropriate size for subimage under the size limitation of SM. However, the limitation interferes with establishing an efficient 3D-BF.

Another problem is that GPU structure is not suitable for the conventional $O(1)$ BF. As described above, $O(1)$ BF consists of multiple $O(1)$ GF which runs voxel by voxel. $O(1)$ GF calculates new voxel by using the value of voxel that has already been calculated. This sequential calculation is difficult to parallelize by GPU; therefore, $O(1)$ BF as combination of $O(1)$ GF is also difficult to calculate efficiently.

B. Methodology

Algorithm 1 Procedure of the proposed method

▷ $\mathbf{f}$: target image, σ_s: spatial scale, σ_r: range scale, τ: tolerance
function PROPOSEDMETHOD($\mathbf{f}, \sigma_s, \sigma_r, \tau$)
  ▷ Parameter Setting
  Set the optimum value for M, T
  ▷ DC-component filtering ($m = 0$)
  $\mathbf{b}_0 \leftarrow \mathbf{1}$
  $\mathbf{b} \leftarrow$ GAUSSIANFILTER_ON_GPU($\mathbf{f}, \sigma_s$)
  ▷ AC-component filtering ($m \geq 1$)
  for $m \leftarrow 1$ to M **do**
    $a_m \leftarrow 2\exp\left(-\frac{1}{2}(\omega_0 m \sigma_r)^2\right)$
    $\mathbf{c}, \mathbf{s} \leftarrow \cos(\omega_0 m \mathbf{f}), \sin(\omega_0 m \mathbf{f})$
    ▷ Partial decompression for denominator β_0
    $\mathbf{\Psi}_{\mathrm{c}} \leftarrow$ GAUSSIANFILTER_ON_GPU($\mathbf{c}, \sigma_s$)
    $\mathbf{\Psi}_{\mathrm{s}} \leftarrow$ GAUSSIANFILTER_ON_GPU($\mathbf{s}, \sigma_s$)
    $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + a_m\{\mathbf{c} \otimes \mathbf{\Psi}_{\mathrm{c}} + \mathbf{s} \otimes \mathbf{\Psi}_{\mathrm{s}}\}$
    ▷ Partial decompression for numerator β
    $\mathbf{\Psi}_{\mathrm{c}} \leftarrow$ GAUSSIANFILTER_ON_GPU($\mathbf{c} \otimes \mathbf{f}, \sigma_s$)
    $\mathbf{\Psi}_{\mathrm{s}} \leftarrow$ GAUSSIANFILTER_ON_GPU($\mathbf{s} \otimes \mathbf{f}, \sigma_s$)
    $\mathbf{b} \leftarrow \mathbf{b} + a_m\{\mathbf{c} \otimes \mathbf{\Psi}_{\mathrm{c}} + \mathbf{s} \otimes \mathbf{\Psi}_{\mathrm{s}}\}$
  end for
  return $\mathbf{b} \oslash \mathbf{b}_0$
end function

Similar to $O(1)$ BF, the proposed method uses (4), which consists of multiple GFs. In order to efficiently compute it on GPU, we use naive convolution instead of $O(1)$ GF as

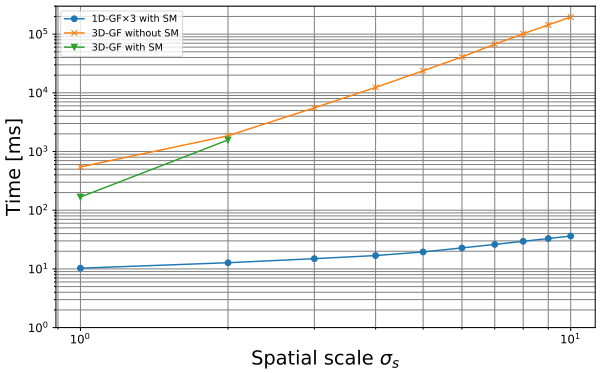


Fig. 2. Processing time of 3D-GF with/without SM and 1D-GF $\times$ 3 with SM when image size is 512^3.

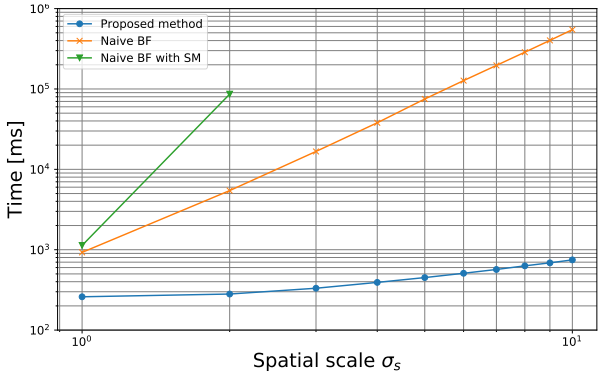


Fig. 3. Processing time of 3D-BF with/without SM and the proposed method when image size is 512^3.

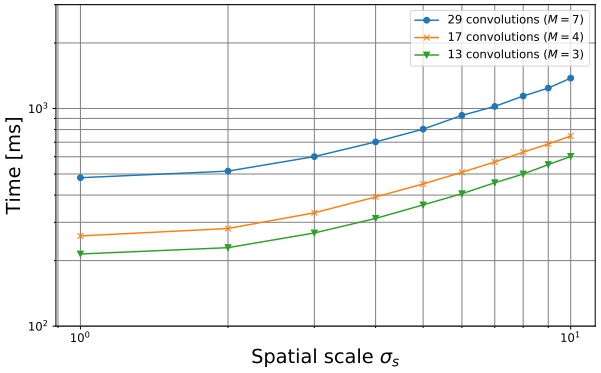


Fig. 4. Processing time of the proposed method when $M = 3, 4, 7$. The parameter $M = 4$ equals to Fig.3.

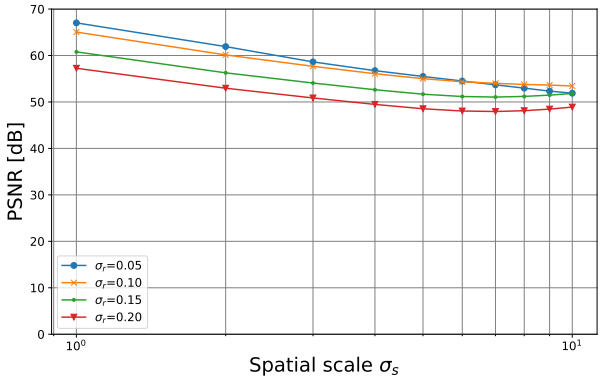


Fig. 5. Approximate accuracy between naive BF and the proposed method when $M = 7, 4, 3, 3$ against $\sigma_r = 0.05, 0.10, 0.15, 0.20$, respectively.

shown in Algorithm 1. This approach can solve the two aforementioned problems. First, we avoid the problem of low parallelism using naive convolution. It is well known that GPU calculates convolutions fast due to their high parallelism. Therefore, the computational time of GF on GPU would be faster than that of $O(1)$ GF if window size is not extremely large. Second, we can considerably relax the problem of SM size in 3D case. Because of the separability of Gaussian kernel, 3D-GF can be decomposed into three 1D-GFs. Conceptually, in the case of 1D convolution, SM can support much larger σ_s because subimages are also 1D. If SM size is 48 KB, it can support $\sigma_s = 100$ or more in theory. In addition, since the computational complexity increases linearly with dimension increase, the proposed method is still effective for 3D cases.

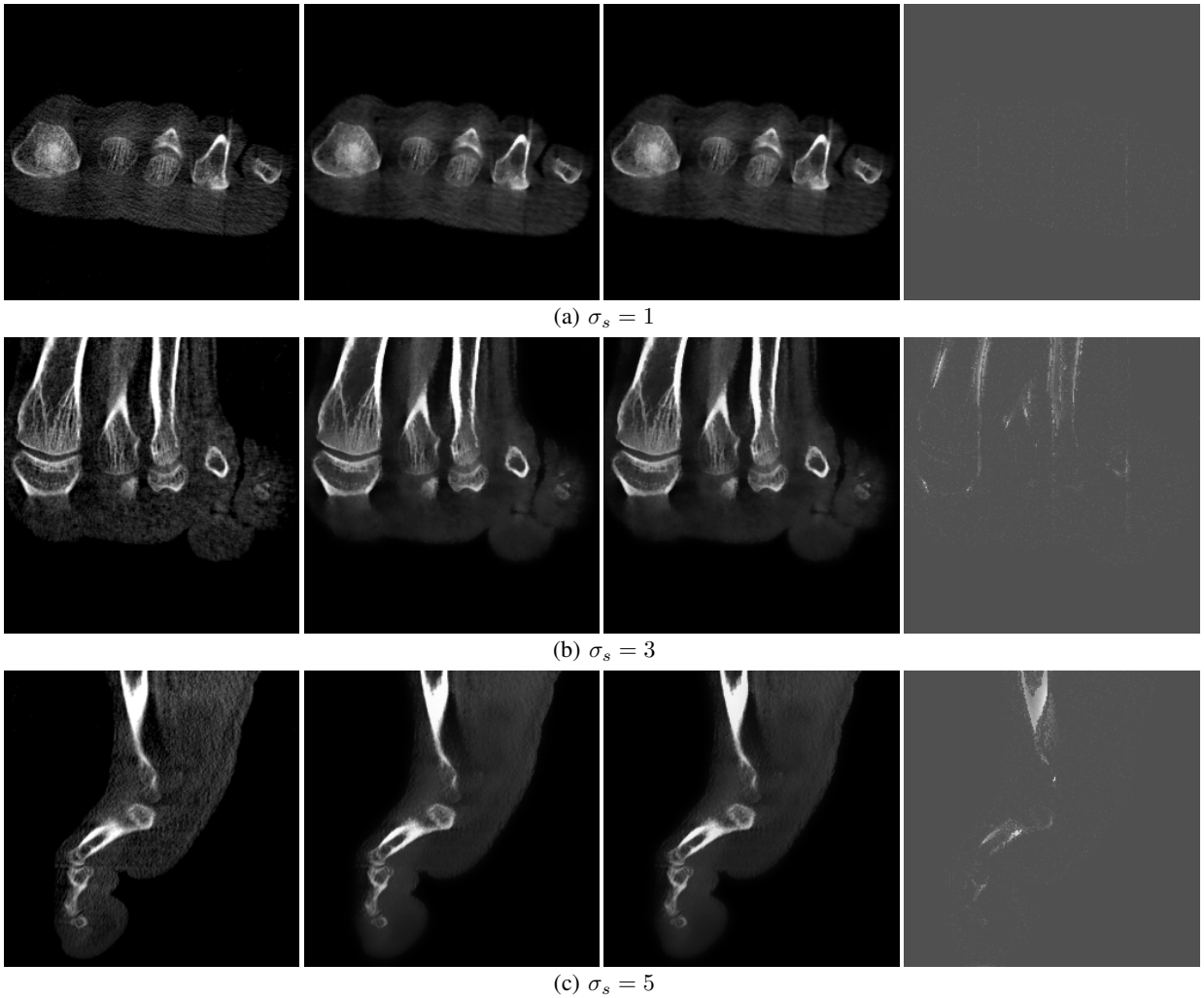
Computational complexity per voxel is $O(\sigma_s^3)$ in naive 3D-BF, $O(K)$ in $O(1)$ BF, and $O(K\sigma_s)$ in the proposed method. Although the computational complexity of the proposed method depends on window length, the computational time would be faster than that of $O(1)$ BF in practice. Since the total computational time of the proposed method is dominated by the multiple GFs, it would nearly equal to $2K$ of GFs.

IV. Experiment

In our experiments, we used an NVIDIA Geforce GTX 1080Ti GPU and an Intel Xeon E5-1620 v4 @ 3.50 GHz CPU, and 16 GB main memory. We implemented all the methods in C++ with CUDA 6.0. We used Compressive BF [7] as $O(1)$ BF for decomposing BF into naive Gaussian convolutions calculated on GPU where the number of convolutions is $4M + 1$. We used mirror image inversion to reference voxels outside of image area in convolutions. The test image consists of 512^3 voxels with 32-bits floating point. We set spatial scale $\sigma_s \in [1, 10]$ and range scale $\sigma_r = 0.1$.

First, we confirm computational time of GF on GPU. Fig.2 shows experimental results of 3D-GF unused SM (`3D-GF w/o SM`), 3D-GF using SM (`3D-GF w/ SM`), and 3D-GF decomposed into three 1D-GF using SM (`1D×3 w/ SM`). Comparison of the former two shows that using SM accelerates convolutions. SM is not effective in $1 < \sigma_s$ from limit of the SM size. On the other hands, the processing time of `1D-GF×3 w/ SM` is more than 50 times faster than `3D-GF w/o SM` in any σ_s since it is not limited to SM size. Hence, the proposed method implements BF by using `1D-GF×3 w/ SM`.

The computational times of 3D-BF is shown in Fig.3 for comparing 3D-BF unused SM (`3D-BF w/o SM`), 3D-BF using SM (`3D-BF w/ SM`), and the proposed method. In this case, we set to $M = 4$. i.e., decomposed into $4M + 1 = 17$ of GFs. The processing time of the proposed method is about 20 times of that of `1D-GF×3 w/ SM`. This result is close to our expected ratio (17 times). When $\sigma_s \leq 10$, the other processing time is dominated by transferring time of subimages, that is about 191 [ms] on average. The total time is less than 1000



(a) $\sigma_s = 1$

(b) $\sigma_s = 3$

(c) $\sigma_s = 5$

Fig. 6. Cross sectional view of 3D medical images and the results of 3D-BF. From left to right, input, naive BF, proposed method, and difference between naive BF and proposed method. Differential images are amplified 8 times.

[ms] when $\sigma_s = 10$. The processing time is proportional to M as shown in Fig.4. Furthermore, when the proposed method works even with a very large window size, e.g., $\sigma_s = 50$ and we set subimage size appropriately, the computational time was nearly theoretical value 3104 [ms].

Fig.5 shows approximate accuracy between naive BF and the proposed method in various σ_r. We quantified the accuracy as Peak Signal-to-Noise Ratio (PSNR). Since smaller σ_r becomes lower PSNR in Compressive BF, we set $M = 7, 4, 3, 3$ when $\sigma_r = 0.05, 0.10, 0.15, 0.20$, respectively. The result shows that PSNR decreases with increase of σ_s. From this results, it seems necessary for maintaining approximate accuracy to increase the number of GFs for larger σ_s. If we assume 50 [dB] indicates sufficient accuracy, the proposed method achieves it over a wide range as $\sigma_s \leq 10$.

Furthermore, we experimented with another image that is a real 3D medical image called "Foot" [6] (256^3 voxels with 16-bits integers). Fig.6, Fig.7, and Fig.8 show the result image, processing time, and approximate accuracy, respectively. Each row in Fig.6 aligns the input image, the output images of the naive BF and the proposed method, and their differential images in order from the left. Each column changes σ_s and viewpoint. From Fig.8, approximate accuracy exceeded 50 [dB] when $\sigma_s \leq 5$ and the results cannot be visually distinguished by appearance as shown in Fig.6. The processing time is about 8 times faster than the case of the image size 512^3, which also followed the theoretical expectation $512^3/256^3 = 8$. If the image size is not simply halved in each dimension, we have to flexibly adjust parameter values such as subimage size. Otherwise, the processing time seems to be slightly increased or decreased.

Finally, we show the approximate accuracy against various σ_r in the test image, "Foot" and the Stanford volume data archive "CThead" [17] that is other medical image. Spatial scale σ_s is fixed two and five. Note that each σ_r is multiplied by the maximum luminance value in each images and M is not fixed. As shown in Fig.9, the PSNR alomost exceeded 50[dB] in those cases.

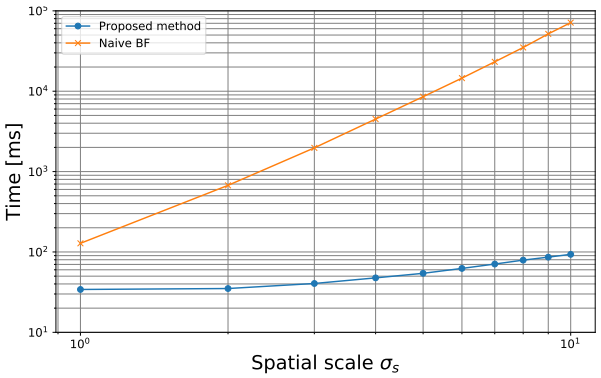


Fig. 7. Processing time of 3D-BF in the medical image "Foot".

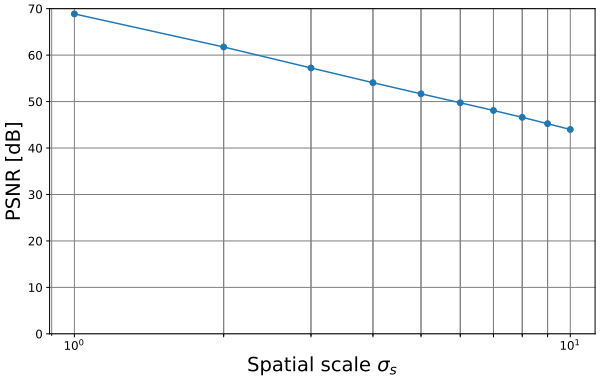


Fig. 8. Approximate accuracy between naive BF and the proposed method in the medical image "Foot".

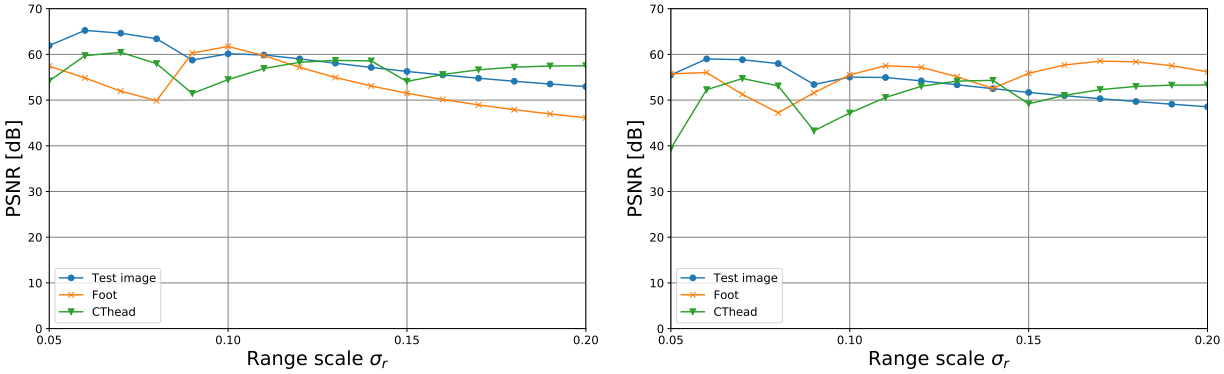


Fig. 9. Approximate accuracy versus σ_r. Left graph sets $\sigma_s = 2$ and right one sets $\sigma_s = 5$.

V. Conclusion

This paper proposed an approximate 3D-BF efficient for GPU. In the experiments of $5 < \sigma_s$, the naive 3D-BF took some minutes; by contrast, the proposed method was able to be processed in less than 1 [s]. The approximate accuracy achieved over 50 [dB], which is visually sufficient. A large window size ($\sigma_s = 50$) can be also performed well. In addition, the proposed method is also considered to be easy to expand to a higher dimension because of separability of Gaussian spatial kernel. The proposed method can be expected to be widely used for 3D data processing such as medical images and high-resolution images.

Acknowledgement

This work was partly supported by JSPS KAKENHI (No.JP18K18076, No.JP16K16092, and No.JP17H01764).

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Jan. 1998, pp. 839–846.

[2] V. Aurich and J. Weule, "Non-linear Gaussian filters performing edge preserving diffusion," in *Mustererkennung 1995, 17. DAGM-Symposium*, 1995, pp. 538–545.

[3] S. M. Smith and J. M. Brady, "SUSAN — a new approach to low level image processing," *Int. J. Comput. Vis. (IJCV)*, vol. 23, no. 1, pp. 45–78, 1997.

[4] A. Buades, B. Coll, and J. M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Modeling and Simulation*, vol. 4, no. 2, pp. 490–530, Jan. 2005.

[5] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, June 2007.

[6] S. Yoshizawa, A. Belyaev, and H. Yokota, "Fast Gauss bilateral filtering," *Computer Graphics Forum*, vol. 29, no. 1, pp. 60–74, Mar. 2010.

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

[8] L. Jonas, "A case study of parallel bilateral filtering on the GPU," *DiVA*, Nov. 2015.

[9] P. Sylvain and D. Fredo, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vis.*, pp. 24–52, Jan. 2009.

[10] H. Mark and E. W. Bethel, "GPU-accelerated denoising of 3D magnetic resonance images," *J. Real-Time Image Process. (JRTIP)*, pp. 713–724, 2017.

[11] Q. Yang, K. H. Tan, and N. Ahuja, "Real-time O(1) bilateral filtering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, June 2009, number 1, pp. 557–564.

[12] K. Sugimoto and S. Kamata, "Efficient constant-time Gaussian filtering with sliding DCT/DST-5 and dual-domain error minimization," *ITE Trans. Media Technol. Appl.*, vol. 3, no. 1, pp. 12–21, 2015.

[13] K. Sugimoto, S. Kyochi, and S. Kamata, "Universal approach for DCT-based constant-time Gaussian filter with moment preservation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 1498–1502.

[14] R. Deriche, "Fast algorithms for low-level vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 1, pp. 78–87, 1990.

[15] I. T. Young and L. J. van Vliet, "Recursive implementation of the Gaussian filter," *Signal Process.*, vol. 44, no. 2, pp. 139–151, June 1995.

[16] L. J. van Vliet, I. T. Young, and P. W. Verbeek, "Recursive Gaussian derivative filters," in *Proc. Int. Conf. Pattern Recognition (ICPR)*, 1998, vol. 1, pp. 509–514.

[17] L. Marc, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, May 1988.