TO APPEAR IN PROC. OF IEEE ICIP'14

Fast L^1 Gaussian Convolution via Domain Splitting

Shin Yoshizawa and Hideo Yokota

RIKEN, Japan

FAST L^1 GAUSSIAN CONVOLUTION VIA DOMAIN SPLITTING

Shin Yoshizawa and Hideo Yokota

Image Processing Research Team, RIKEN, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan
e-mail: {shin,hyokota}@riken.jp

ABSTRACT

This paper proposes a fast and accurate approximation algorithm to convolve a L^1 Gaussian function with images. Our new algorithm is based on splitting a pixel domain into representative regions where we can efficiently perform discrete convolutions. Our algorithm is applicable to non-uniform pixels with linear computational complexity. We examine it numerically in terms of speed, precision, and quality. We also introduce a novel edge-aware filter by using our algorithm.

Index Terms— Laplace Distribution, Fast Discrete Convolution, Edge-aware Filtering, Domain Transform

1. INTRODUCTION

Gaussian convolution is a very powerful, fundamental tool that has a wide variety of useful applications in signal and image processing. Computing a discrete analog of Gaussian convolution, also called a *Gauss transform* [1], quickly and accurately is important, and it has been studied for linear and nonlinear image filtering, e.g., edge-aware smoothing [2, 3, 4] and kernel density estimation [5]. Unfortunately, performing an exact Gauss transform requires quadratic computational complexity. A naive truncation algorithm often produces undesired artifacts because it leads to a *sinc*-like kernel shape in its Fourier domain similar to a box kernel (Fig. 1). Therefore, various approximation methods have been suggested such as recursive filtering [6, 7, 8, 9, 10], polynomial splines [11], Taylor expansions [2], trigonometric kernels [12], Fast Fourier Transforms (FFT) [13, 3], kd-trees [14], and Fast Gauss Transforms (FGT) [1, 15]. However, the box kernel is still popular [16, 2, 17, 18] because of its high applicability to regular image structures, and better approximations are in demand for integrating large image domains with Gaussian kernels.

In this paper, we propose a novel, fast, and accurate approximation algorithm for discrete Gaussian convolutions on images. Our algorithm is based on splitting a pixel domain by using representative points (*poles*) and decomposing the equation into terms that can be pre-computed efficiently. Our main idea is to use the L^1 norm instead of the popular L^2 norm for distances between image pixels. The L^1 norm Gaussian $G(x) = \exp(-\frac{|x|}{\sigma})$ at a point $x \in \mathbb{R}$, where $\sigma > 0$ is the standard deviation, also known as Laplace distribution in statistics and probability theory [19], has many of the same useful properties as the L^2 Gaussian. It is separable, and its integral over $\mathbb{R}$ gives a finite number $\int_{-\infty}^{\infty} G(x)dx = 2\sigma$. Moreover, the Fourier spectrum of $G(\cdot)$, which becomes a rational quadratic function $(\sigma\sqrt{2/\pi})/(1+\sigma^2x^2)$ and is frequently employed for filter kernels, does not have extrema, except for the point at $x = 0$. Thus, the normalized convolution $\int G(x-y)I(y)dy / \int G(x-y)dy$ of an image intensity $I = I(x) \in \mathbb{R}$ at x combined with a separable implementation yields us nice smoothing results (Fig. 1). Our algorithm is applicable to non-uniform pixels with linear computational complexity (constant w.r.t. σ). We examine our algorithm numerically and compare it with FFT, B-spline, FGT, naive truncation, and box kernel approaches in terms of speed, approximation precision, and visual quality. We also introduce a novel edge-aware filter by using our algorithm.

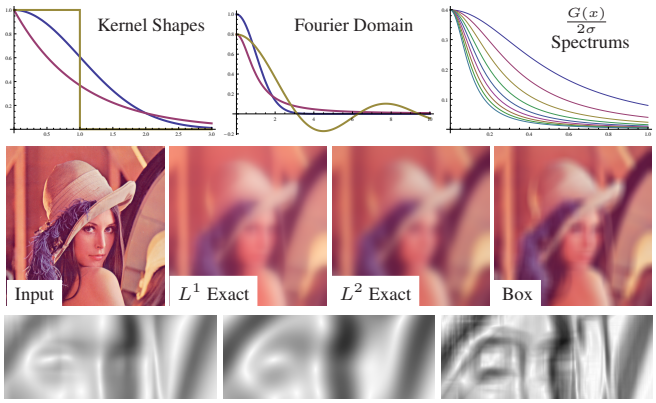


Fig. 1: Convolutions via L^1 and L^2 Gaussians and box kernels. The bottom images are the corresponding gradient magnitudes $|\nabla I|$.

2. L^1 GAUSSIAN WITH DOMAIN SPLITTING

Consider a set of n points $\{x_j\}$ on $\mathbb{R}$ where $x_1 \le x_2 \le ... \le x_n$. For a fixed point x_1, the L^1 norm distance between x_i and x_j is decomposed by splitting its domain as $|x_i - x_j| =$

$$\begin{cases} |x_i - x_1| - |x_j - x_1| & \text{if} \quad x_1 \le x_j \le x_i : x_j \in D_1, \\ -|x_i - x_1| + |x_j - x_1| & \text{if} \quad x_1 \le x_i < x_j : x_j \in D_2. \end{cases}$$

Thus, an L^1 Gaussian $G(x_i - x_j)$ can also be represented by

$$\begin{cases} \exp(-\frac{|x_i-x_1|}{\sigma} - \frac{-|x_j-x_1|}{\sigma}) = \frac{G(x_i-x_1)}{G(x_j-x_1)} & \text{if} \quad x_j \in D_1, \\ \exp(-\frac{-|x_i-x_1|}{\sigma} - \frac{|x_j-x_1|}{\sigma}) = \frac{G(x_j-x_1)}{G(x_i-x_1)} & \text{if} \quad x_j \in D_2. \end{cases}$$

Consider an L^1 Gauss transform $J(x_j) = \sum_{i=1}^{n} G(x_i - x_j)I(x_i)$ with $I(x) \in \mathbb{R}$ at x_j. By splitting the domain into D_1 and D_2, we introduce a novel representation of $J(x_j)$ by

$$J(x_j) = I(x_j) + \{G(x_j - x_1)\sum_{i=1}^{j-1} \frac{1}{G(x_i - x_1)} I(x_i)\} + \\ + \{\frac{1}{G(x_j - x_1)} \sum_{i=j+1}^{n} G(x_i - x_1) I(x_i)\}. \quad (1)$$

Equation (1) can be calculated by $O(3n)$ operations (big O notation [20]) for all $j = 1, 2, ..., n$ (linear computational complexity), since $\sum_{i=1}^{j-1} \frac{1}{G(x_i - x_1)} I(x_i)$ and $\sum_{i=j+1}^{n} G(x_i - x_1) I(x_i)$ are pre-computed once for all $j = 1, 2, ...n$. Sorting $\{x_j\}$ is not necessary when $\{x_j\}$ consists of image pixel coordinates including transformed non-uniform values (e.g., [17, 18]). Computing equation (1) efficiently and accurately is not trivial if some numerical issue exists, for instance, $\frac{1}{G(x_i - x_1)}$ may cause an overflow for huge values of $|x_i - x_1|$.

3. FAST AND ACCURATE APPROXIMATION

Inspired by the Fast Multipole Method (FMM) [21] and FGT, but with a different approach, we introduce a set of m representative poles $\{\alpha_k\}$ on $\mathbb{R}$ instead of using the fixed point x_1 to avoid the above-mentioned numerical problem.

Assuming that $\alpha_1 < \alpha_2 < ... < \alpha_m$, the domain splitting of $|x_i - x_j|$ around the pole α_k is given by $|x_i - x_j| =$

$$\begin{cases} |x_i - \alpha_k| - |x_j - \alpha_k| & \text{if } \begin{cases} \alpha_k \le x_j \le x_i, \\ x_i \le x_j \le \alpha_k \end{cases} : x_j \in \Omega_1, \\ -|x_i - \alpha_k| + |x_j - \alpha_k| & \text{if } \begin{cases} \alpha_k \le x_i < x_j, \\ x_j < x_i \le \alpha_k \end{cases} : x_j \in \Omega_2, \\ |x_i - \alpha_k| + |x_j - \alpha_k| & \text{if } \begin{cases} x_j < \alpha_k \le x_i, \\ x_i \le \alpha_k < x_j \end{cases} : x_j \in \Omega_3. \end{cases}$$

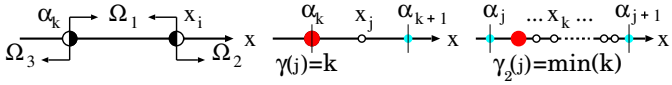


Fig. 2: Domains and illustrations of $\gamma(\cdot)$ and $\gamma_2(\cdot)$.

Thus, the L^1 Gaussian $G(x_i - x_j)$ is decomposed by $\frac{G(x_i - \alpha_k)}{G(x_j - \alpha_k)}$, $\frac{G(x_j - \alpha_k)}{G(x_i - \alpha_k)}$, and $G(x_j - \alpha_k)G(x_i - \alpha_k)$ for x_j, which belong to Ω_1, Ω_2, and Ω_3, respectively. By using this domain splitting with the poles $\{\alpha_k\}$, equation (1) becomes

$$J(x_j) = I(x_j) + C_j + D_j + E_j, \quad (2)$$

$$C_j = \{G(x_j - \alpha_{\gamma(j)}) \sum_{i=\gamma_2(\gamma(j))}^{j-1} \frac{1}{G(x_i - \alpha_{\gamma(j)})} I(x_i)\} + \\ + \{\frac{1}{G(x_j - \alpha_{\gamma(j)})} \sum_{i=j+1}^{\gamma_2(\gamma(j)+1)-1} G(x_i - \alpha_{\gamma(j)}) I(x_i)\},$$

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

$$A_k = \sum_{i=\gamma_2(k)}^{\gamma_2(k+1)-1} \frac{I(x_i)}{G(x_i - \alpha_k)}, \quad B_k = \sum_{i=\gamma_2(k)}^{\gamma_2(k+1)-1} G(x_i - \alpha_k) I(x_i)$$

where $\gamma(j) = k$ such that $\alpha_k \le x_j < \alpha_{k+1}$ and $\gamma_2(j) = \min(k)$ such that $\alpha_j \le x_k < \alpha_{j+1}$, see Fig. 2.

To avoid numerical instability, the inequality $\exp(\frac{|\alpha_{k+1} - \alpha_k|}{\sigma}) < \text{MAX}$ should be held where MAX is the maximum value of precision (i.e., double floating-point precision in our numerical experiments). Therefore, we choose the distances between poles by $|\alpha_{k+1} - \alpha_k| = \varphi\sigma \log(\text{MAX})$, where $\varphi \in (0, 1)$ is a parameter. In practice, we use $\varphi = 0.5$ and poles $\{\alpha_k\} = \{0, w/m, 2w/m, 3w/m, ..., (m-1)w/m\}$ where $w = |x_n - x_1|$. In addition, m is determined by $m = [\frac{x_n - x_1}{\alpha_{k+1} - \alpha_k}]$, where [] is the ceiling function, which results in $m \ll n$ in general cases.

The terms A_k, B_k, and C_j can be computed to $O(n)$ for the set $\{x_j\}$. The terms D_j and E_j are also L_1 Gauss transforms that need $O(nm + m)$. Note that equation (2) is numerically stable because of how the $\{\alpha_k\}$ are chosen.

Since $G(\alpha_k - x_j)$ becomes numerically zero if $|\alpha_k - x_j| > \sigma \log(\text{MAX})$, we approximate D_j and E_j by

$$D_j \approx G(x_j - \alpha_{\gamma(j)-1}) A_{\gamma(j)-1}, \quad (3)$$

$$E_j \approx G(x_j - \alpha_{\gamma(j)+1}) B_{\gamma(j)+1} \quad (4)$$

where $D_j \approx 0$ if $\gamma(j) < 2$, $E_j \approx 0$ if $\gamma(j) > n - 1$, and their complexities reduce from $O(nm + m)$ to $O(m)$.

The approximation error at x_j is given by

$$\text{Error}_j = \sum_{k=1}^{\gamma(j)-2} G(x_j - \alpha_k) A_k + \sum_{k=\gamma(j)+2}^{m} G(x_j - \alpha_k) B_k.$$

This implies that the precision of our approximation is equivalent to truncating a sum within a $3\varphi\sigma \log(\text{MAX})$ region:

$$J(x_j) \approx \sum_{i=\gamma_2(\gamma(j)-1)}^{\gamma_2(\gamma(j)+1)-1} G(x_i - x_j) I(x_i). \quad (5)$$

This approximation is very accurate as long as $\max_i(I(x_i)) < \exp(\varphi \log(\text{MAX})) = \text{MAX}^{\varphi}$, which is the most frequent case scenario in image processing applications. Equation (5) with a naive truncation algorithm requires time-consuming computations of $O(3n\varphi\sigma \log \text{MAX})$, but our algorithm requires only $O(4n + 3m)$ operations (const. $O(1)$ w.r.t. σ).

3.1. Multidimensional Algorithm and Edge-preserving

Although our domain splitting technique can be extended to multidimensional cases, we employ a separable implementation for 2D images because of its simplicity and because the L^1 Gaussian is also separable like L^2, i.e., there should not be much difference between their results for linear filtering.

For a given image with a user-specified σ, we first perform a normalized Gauss transform $J(x_j)/\sum_{i=1}^{n} G(x_j - x_i)$ for $\{x_j\}, j = 1, 2, ..., n$ for each column. Then we perform a normalized Gauss transform for each row by using the filtered values of the columns. The color channels and denominator are separately processed.

For each Gauss transformation, the poles $\{\alpha_k\}$ are calculated first. The terms A_j, B_j, and C_j are computed for $j = 1, 2, ..., n$. Then, the transformed values are obtained by evaluating equation (2) by using equations (3) and (4) for $j = 1, 2, ..., n$. For linear filtering with uniform pixels, $\{\alpha_k\}$ and $\exp(\cdot)$ are pre-computed once a dimension. Fig. 3 shows a smoothing example via our algorithm with varying σ.

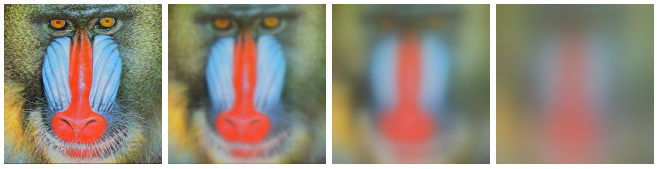


Fig. 3: Smoothing via our algorithm with $\sigma \in \{5, 20, 55\}$.

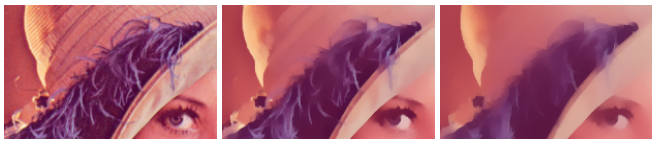


Fig. 4: Edge-aware filtering via our algorithm with $\frac{\sigma}{\phi} \in \{5, 20\}$.

We also adapt our algorithm to an edge-aware L^1 Gaussian filter based on the domain transformation technique [17]. The main idea is to use geodesic distances on the image manifold [22] instead of Euclidean distances between pixels.

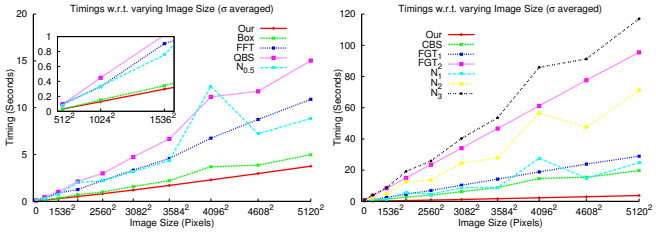
Consider a point $\mathbf{p} = (x_p, \mathbf{I}(x_p))$ on the image manifold $(x, \mathbf{I}(x)) \in \mathbb{R}^4, x \in \mathbb{R}$ where $\mathbf{I} = \mathbf{I}(x) \in \mathbb{R}^3$ is a color vector at x. The distance from the origin $(0, \mathbf{I}(0))$ to $\mathbf{p}$ on the image manifold gives a transformation $T(x_p) : \mathbb{R}^4 \to \mathbb{R}$ where

$$T(x_p) = \int_0^{x_p} \sqrt{1 + \lambda^2 |\nabla \mathbf{I}(t)|^2}\, dt, \quad \lambda = \phi(\sigma_s/\sigma), \quad (6)$$

σ_s is the standard deviation of the image intensity, and $\phi > 0$ is a parameter that controls edge-awareness. Convolving $\mathbf{I}$ with $G(\cdot)$ on the domain $\{T(x_j)\}$ provides an edge-aware filter because the gradient magnitude $|\nabla \mathbf{I}(t)|$ yields a large value at the image edges. Because we only assume the orders of $\{x_j\}$ and $\{\alpha_k\}$, non-uniform values $\{T(x_j)\}$, where $T(x_1) \le T(x_2) \le ... \le T(x_n)$, are employed in our edge-aware filter (Fig. 4). In contrast to [17], which established a box kernel with L^1 geodesic distance, our filter consists of the L^1 Gaussian kernel with the L^2 geodesic distance (6).

4. COMPARISON AND EVALUATION

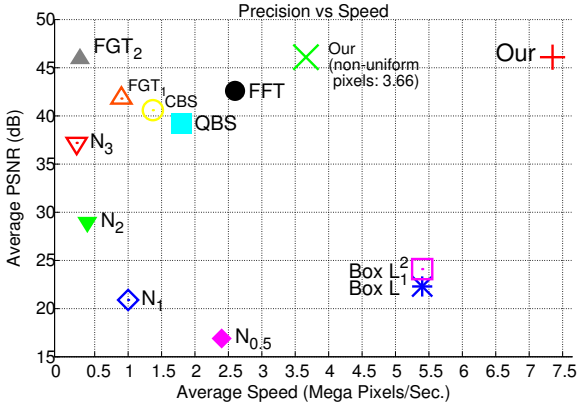
All numerical experiments in this paper were performed on a Core2 Extreme X9770 (3.2 GHz quad core, no parallelization was used) PC with 16GB RAM and 64 bit OS. We compared our algorithm with separable implementations of the FFT using FFTW [23], FGTs [1, 15] with a sharp error estimator [24], quadratic (QBS) and cubic (CBS) cardinal B-splines [25], naive truncations (N_r) within their radii $r \in \{0.5, 1, 2, 3\}\sigma$, and box kernel (Box) in linear filtering. The box kernel filter was implemented by using a moving average method [26] with its box radius equal to σ. We also used two FGT parameter settings, namely fast (FGT_1) and accurate (FGT_2) that employ $(2, 10)$ and $(6, 1)$ for the interaction radius and allowed error of [1], respectively. The naive truncations were performed by convolving a $(2r{+}1)$ length array consisting of pre-computed L^1 Gaussian values. Recursive box filtering with a scaling factor $\sqrt{(k+1)/12}/\sigma$ and 0.5 radius converges to a L^2 Gaussian [27], which when iterated $(k{+}1)$ times yields a k degree cardinal B-spline. We applied the moving average method with 3 and 4 iterations for QBS and CBS, respectively.



	Our	FFT	FGT_1	FGT_2	QBS	CBS
T_1	**0.03**	0.1	0.28	0.89	0.09	0.12
T_2	**3.75**	10.9	28.9	95.5	15.0	19.8

	N_3	N_2	N_1	$N_{0.5}$	Box	Exact
T_1	0.85	0.44	0.15	0.08	0.029	3.3
T_2	117	71	25	8.9	4.99	1.33 **hours**

Fig. 5: Timing comparison measured in seconds. T_1 and T_2 are average timings for the color images with 512^2 and 5120^2 pixels.



	Our	FFT	FGT_1	FGT_2	QBS	CBS
PSNR	**46.1**	42.6	41.8	45.9	39.2	40.6
$E_{\max}$	**0.99**	0.99	1.16	0.91	3.7	4.13
Speed	**7.35**	2.58	0.9	0.28	1.8	1.37

	N_3	N_2	N_1	$N_{0.5}$	Box L_1	Box L_2
PSNR	37.2	29	20.9	16.9	22.3	24.1
$E_{\max}$	4.52	9.07	16.3	21.7	15.7	11.7
Speed	0.23	0.39	1.0	2.44	5.43	

Fig. 6: Performance evaluations. Speed: megapixels/second.

We employed color images with 12 values of σ (every 5 values from 5 to 60) for 11 image sizes (512^2 to 5120^2 pixels). Fig. 5 shows the averaged computational timings over

σ w.r.t. image sizes. Our algorithm achieves linear computational complexity and the fastest speed in these experiments.

We evaluated the maximum error $E_{\max} = \max(|d_i|)$, $i \in \{1, 2, ..., M\}$, where I_i^e and I_i^a are the exact and approximated color values at x_i, $d_i = I_i^e - I_i^a$, and M is the number of 2D image pixels. Moreover, we examined the peak signal-to-noise ratio (PSNR; the larger, the better) [2, 3] by averaging $-10\log_{10}(\frac{1}{M}\sum_{i=1}^{M}(\frac{d_i}{\max(I_i^e, I_i^a)})^2)$ over all color channels. Here, FGTs, QBS, and CBS were compared with the exact L^2 Gaussian convolutions (the others used the exact L^1).

Fig. 6 illustrates the precision versus speed by comparing values of the averaged PSNR and maximum $E_{\max}$ over σ, image sizes, and color channels. As expected from theory, our algorithm, FFT, and FGT_2 provide high quality results according to PSNR and $E_{\max}$. The naive truncations N_r and box kernel result not only in low approximation precision, but also produce undesired visual artifacts (Fig. 7). At first glance, N_3 seems to perform well but it also produces serious artifacts around sharp features and generates false edges similar to the box filter (Fig. 1); see $|\nabla|\nabla I||$ and $|\nabla I|^{\frac{1}{2}}$ in Fig. 7. These naive approximations, including B-splines and aggressive setting of FGT (see Fig. 7, bottom), are not able to suppress oscillations of their kernel shapes in corresponding Fourier domains in the same way as the box kernel. This leads to low-quality results, both numerically (PSNR and $E_{\max}$) and visually. Surprisingly, our algorithm gives better results than FFT w.r.t. both PSNR and visual inspection (Fig. 8). Overall, our algorithm provides the best quality and performance in terms of numerical errors, and visual inspections, and achieves the satisfactory processing speed of 7.35 megapixels per second (3.66 M/sec. for non-uniform pixels).

Figs. 9 and 10 demonstrate applications of our algorithm in edge-aware filtering (which requires integration over a large domain [13, 17]) and detail enhancement, respectively. A simple enhancement technique $I^f + 3(I - I^f)$ is used where I^f is the edge-aware filtered image. In a few seconds, our edge-aware filter successfully processes a megapixel image and produces a nice enhanced image without halo artifacts.

5. CONCLUSION

We have proposed a fast and accurate L^1 Gaussian convolution algorithm for images. Our new algorithm is based on splitting a pixel domain into representative regions in which discrete convolutions are efficiently approximated. Our algorithm, which is applicable to non-uniform pixels with linear computational complexity (constant w.r.t. σ), achieves high performance results in terms of speed, precision, and quality. We also introduced a novel edge-aware filter that uses our algorithm. Since our algorithm can process HDR images without heuristics, applications to computational photography, engineering, and natural science hold future promise. We also would like to apply our algorithm to machine learning such as L^1 regression [28] and kernel density estimation [5].

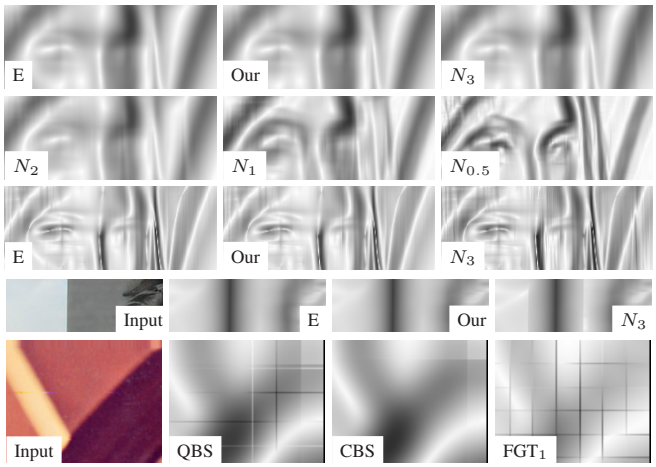


Fig. 7: Visual comparisons and artifacts ($\sigma = 15$). E: Exact. The 3rd and 4th columns from top correspond to $|\nabla|\nabla I||$ and $|\nabla I|^{\frac{1}{2}}$, respectively. The other columns show $|\nabla I|$.

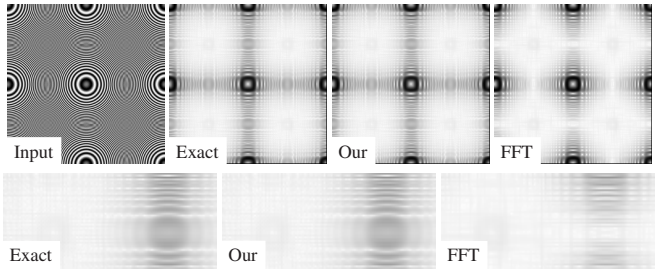


Fig. 8: Visual comparisons of $|\nabla I|$ with FFT ($\sigma = 10$).

Fig. 9: Edge-aware filtering via our algorithm with $\phi = 0.1$ and $\sigma = 20$ took 5.1 seconds (3 iterations [17]) for 2593 x 1945 pixels.

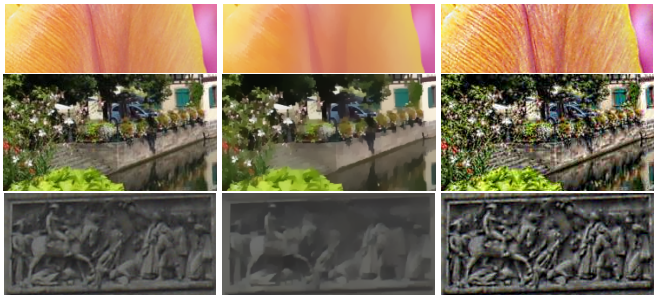


Fig. 10: Edge-aware filtering with detail enhancement via our algorithm with $\phi = 0.1$ and $\sigma = 20$. The left, center, and right images correspond to input, filtered, and enhanced images, respectively.

6. REFERENCES

[1] L. Greengard and J. Strain, "The fast Gauss transform," *SIAM J. Sci. Stat. Comput.*, vol. 12, no. 1, pp. 79–94, 1991.

[2] F. Porikli, "Constant time O(1) bilateral filtering," in *Proc. of IEEE-CS Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2008, pp. 1–8, IEEE-CS.

[3] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 24–52, 2009.

[4] S. Yoshizawa, A. Belyaev, and H. Yokota, "Fast Gauss bilateral filtering," *Comput. Graph. Forum*, vol. 29, no. 1, pp. 60–74, 2010.

[5] A. Elgammal, R. Duraiswami, and L. S. Davis, "Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 11, pp. 1499–1504, 2003.

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

[7] L. Alvarez and L. Mazorra, "Signal and image restoration using shock filters and anisotropic diffusion," *SIAM J. Numer. Anal.*, vol. 31, no. 2, pp. 590–605, 1994.

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

[9] J. Geusebroek, A. Smeulders, and J. van de Weijer, "Fast anisotropic Gauss filtering," *IEEE Trans. Image Process.*, vol. 12, no. 8, pp. 938–943, 2003.

[10] P. Getreuer, "A survey of Gaussian convolution algorithms," *Image Process. On Line*, vol. 3, pp. 276–300, 2013.

[11] M. Unser, "Splines: A perfect fit for signal and image processing," *IEEE Signal Process. Mag.*, vol. 16, no. 6, pp. 22–38, 1999.

[12] K. Chaudhury, D. Sage, and M. Unser, "Fast O(1) bilateral filtering using trigonometric range kernels," *IEEE Trans. Image Process.*, vol. 20, no. 12, pp. 3376–3382, 2011.

[13] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *Proc. of ACM SIGGRAPH*. 2002, pp. 257–266, ACM Press.

[14] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian kd-trees for fast high-dimensional filtering," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 21:1–21:12, 2009.

[15] L. Greengard and X. Sun, "A new version of the fast Gauss transform," in *Proc. of IEEE Int. Cong. of Mathematicians III*, 1998, pp. 575–584.

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

[17] E. Gastal and M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 69:1–69:12, 2011.

[18] E. Gastal and M. Oliveira, "Adaptive manifolds for real-time high-dimensional filtering," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 33:1–33:13, 2012.

[19] S. Kotz, T. Kozubowski, and K. Podgorski, *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*, Birkhäuser Boston, 2001.

[20] P. Bachmann, *Analytische Zahlentheorie*, Leipzig, 1894.

[21] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987.

[22] N. Sochen, R. Kimmel, and R. Malladi, "A general framework for low level vision," *IEEE Trans. Image Process.*, vol. 7, no. 3, pp. 310–318, 1998.

[23] M. Frigo and S. Johnson, "The design and implementation of FFTW3," in *Proc. of the IEEE*. 2005, pp. 216–231, IEEE, Library: www.fftw.org.

[24] X. Wan and G. Karniadakis, "A sharp error estimate for the fast Gauss transform," *J. Comput. Phys.*, vol. 219, no. 1, pp. 7–12, 2006.

[25] I. Schoenberg, "Cardinal interpolation and spline functions," *J. Approximation Theory*, vol. 2, pp. 167–206, 1969.

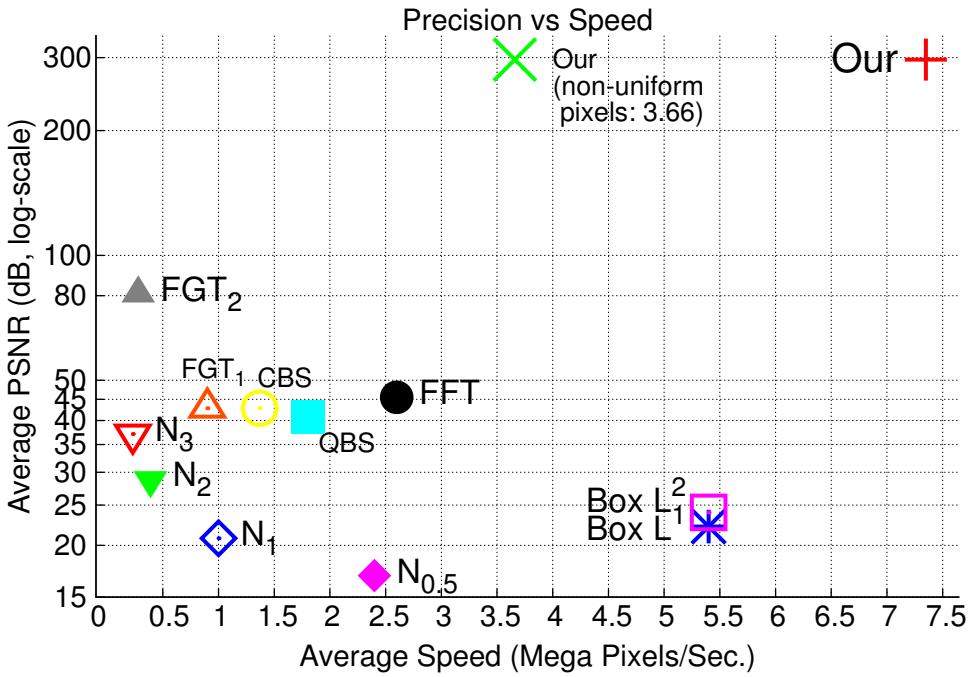
[26] E. Dougherty, *Digital Image Processing Methods*, CRC Press, 1994.

[27] M. Unser, A. Aldroubi, and M. Eden, "On the asymptotic convergence of B-spline wavelets to Gabor functions," *IEEE Trans. Inf. Theory*, vol. 32, no. 2, pp. 864–864, 1992.

[28] F. Bach, "Sparse methods for machine learning," in *Tutorial of IEEE-CS Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2010, IEEE-CS.

Supplemental

In the paper, the precision described in Fig. 6 was measured by using 16 bit images for the exact L^1 and L^2 filterings. Here we show more accurate evaluations in Fig. S.1 by using 64 bit images for the exact L^1 and L^2 filterings, where setting of experiments is same as in Fig. 6. The PSNR of FGT_2 and our (also $E_{\max}$ of FFT and our) results become significantly better than the results reported in Fig. 6.



	Our	FFT	FGT_1	FGT_2	QBS	CBS
PSNR	**296.7**	45.5	42.8	80.7	40.8	42.8
$E_{\max}$	**0.11**$\times 10^{-12}$	0.05	0.77	0.13	3.9	3.61
Speed	**7.35**	2.58	0.9	0.28	1.8	1.37

	N_3	N_2	N_1	$N_{0.5}$	Box L_1	Box L_2
PSNR	37.1	28.8	20.8	16.9	22.2	24
$E_{\max}$	5	9.87	17.1	22.3	16.5	11.7
Speed	0.23	0.39	1.0	2.44	5.43	

Fig. S.1: Performance evaluations. Speed: megapixels/second.