Accelerate bilateral filter using Hermite polynomials

Longquan Dai, Mengke Yuan and Xiaopeng Zhang

The bilateral filter (BF) as an edge-preserving lowpass filter is a valuable tool in various image processing tasks, including noise reduction and dynamic range compression. However, its computational cost is too high to apply in the real-time processing tasks as the range kernel, which acts on the pixel intensities, making the averaging process nonlinear and computationally intensive, particularly when the spatial filter is large. Using the well-known Hermite polynomials, a BF accelerating method is proposed, which reduces the computational complexity from $O(r^2n)$ to $O(n)$, where r denotes the filter size of a BF and n is the total number of pixels in an image.

Introduction and related work: The bilateral filter (BF) [4] is a valuable tool for many computer vision and computer graphic applications such as denoising, demosaicing and optical-flow estimation. To preserve the detail information in the image, a BF computes its results by a weighted average of the pixels in a neighbourhood with signal-dependent coefficients. This is significantly different from the traditional FIR lowpass filter that employs a predefined convolution kernel to smooth images. Although the change endows the BF the ability to detect edges and to only average the pixels on the same side of an edge, the advantages come at a cost as the BF substitutes the efficient linear convolution with the time-consuming nonlinear convolution.

The brute-force implementation of a BF calculates the filtering results I^{bf} with $O(r^2n)$ complexity by directly performing the nonlinear convolution on the target image I as illustrated in (1) and (2)

$$I_p^{\mathrm{bf}} = \frac{1}{W_p^{\mathrm{bf}}} \sum_{q\in\Omega_p} G_{\sigma_s}(p-q)G_{\sigma_r}(I_p-I_q)I_q \tag{1}$$

$$\text{with}\quad W_p^{\mathrm{bf}} = \sum_{q\in\Omega} G_{\sigma_s}(p-q)G_{\sigma_r}(I_p-I_q) \tag{2}$$

where $G_{\sigma_s}(p-q) = \mathrm{e}^{-\sigma_s\|p-q\|^2}$ and $G_{\sigma_r}(I_p-I_q) = \mathrm{e}^{-\sigma_r(I_p-I_q)^2}$ are the linear Gaussian spatial kernel and the nonlinear Gaussian range kernel, respectively, Ω_p denotes the neighbourhood of the pixel p. From (1) and (2), we can easily find out that both the nonlinearity and the signal-dependent coefficients of the BF are introduced by the range kernel G_{σ_r} term as the parameters of G_{σ_r} involves the intensity image I. If the nonlinear kernel G_{σ_r} can be 'linearised', the smoothing results of BF could be computed efficiently. Specifically, for an image of size n, we can compute the linear convolution for arbitrary window size Ω_p with arbitrary kernel at the cost of n log n operations using the Fourier transform. Moreover, this can be further reduced to a total of $O(n)$ operations [5] by the recursive implementation of the Gaussian filter.

In the literature, Porikli [2] employed the Taylor series to linearise the nonlinear term of the BF. Roughly speaking, the Gaussian function can be approximated by the first M terms of its Taylor expansion with arbitrary accuracy, as illustrated in (3)

$$\mathrm{e}^{-\sigma_r x^2} = \sum_{i=0}^{M} (-1)^i \frac{x^{2i}}{i!} \tag{3}$$

Then, applying the binomial formula $(x+y)^i = \sum_{k=0}^{i}\binom{i}{k}x^{i-k}y^k$ to (3), we can write $\mathrm{e}^{-\sigma_r(I_p-I_q)^2}$ as

$$\mathrm{e}^{-\sigma_r(I_p-I_q)^2} = \sum_{i=0}^{M}\frac{(-1)^i}{i!}\sum_{k=0}^{2i}\binom{2i}{k}I_p^{2i-k}I_q^k \tag{4}$$

Furthermore, plugging (4) into (1), the value I_p^{bf} can be figured out by (5)

$$I_p^{\mathrm{bf}} = \frac{1}{W_p^{\mathrm{bf}}}\sum_{i=0}^{M}\sum_{k=0}^{2i}\frac{(-1)^i}{i!}\binom{2i}{k}I_p^{2i-k}\sum_{q\in\Omega_p}G_{\sigma_s}(p-q)I_q^{k+1} \tag{5}$$

Note that the inner sum $\sum_{q\in\Omega_p}$ of (5) is just the linear convolution performed on the image I^{k+1}. According to [5], the operation could be completed in constant time. Additionally, the outer sums $\sum_{i=0}^{M}\sum_{k=0}^{2i}$ of (5) are pointwise operations that can also be computed in linear time. In a similar way, we can reduce the computational complexity of the denominator W_p^{bf} of BF to $O(n)$ by substituting (4) with G_{σ_r} in (2).

Chaudhury *et al.* [1] argued that the Taylor expansion is inaccurate for approximating the Gaussian function as it only uses polynomials to approximate an exponential function. Instead, they exploited trigonometric functions to approximate the Gaussian function by proving

$$\mathrm{e}^{-\sigma_r x^2} = \lim_{N\to\infty}\left[\cos\left(\frac{\sigma_r x}{\sqrt{2N}}\right)\right]^N \tag{6}$$

Equation (6) implies that for a sufficiently large M, the value on the right part approaches the value of the Gaussian function on the left part

$$\mathrm{e}^{-\sigma_r x^2} = \left[\cos\left(\frac{\sigma_r x}{\sqrt{2M}}\right)\right]^M \tag{7}$$

Jointly employing the binomial formula and the angle addition formula $\cos(x-y) = \cos(x)\cos(y) + \sin(x)\sin(y)$, Chaudhury *et al.* transform $\mathrm{e}^{-\sigma_r(I_p-I_q)^2}$ into (8)

$$\mathrm{e}^{-\sigma_r(I_p-I_q)^2} = \sum_{k=0}^{M}\binom{M}{k}C_p^{M-k}S_p^kC_q^{M-k}S_q^k \tag{8}$$

where $C = \cos\left(\sigma_r I/\sqrt{2M}\right)$ and $S = \sin\left(\sigma_r I/\sqrt{2M}\right)$. Then, plugging (8) into (1), Chaudhury *et al.* calculate I_p^{bf} by

$$I_p^{\mathrm{bf}} = \frac{1}{W_p^{\mathrm{bf}}}\sum_{k=0}^{M}\binom{M}{k}C_p^{M-k}S_p^k\sum_{q\in\Omega_p}G_{\sigma_s}(p-q)C_q^{M-k}S_q^kI_q \tag{9}$$

Similar to (5), the inner sum $\sum_{q\in\Omega_p}$ of (9) is the linear convolution performed on the image $C^{M-k}S^k$ and the outer sum $\sum_{k=0}^{M}$ is the pointwise operation. The running cost of the denominator W_p^{bf} of BF can also be reduced to $O(n)$ in the same way. Therefore, we say that the overall complexity is $O(n)$.

Although both Porikli and Chaudhury claimed that the computational cost of their methods is linear, the actual running cost is markedly different. Specifically, the method of Porikli involves computing the auxiliary images I^{k+1}, whereas the method of Chaudhury needs to compute the auxiliary images $C^{M-k}S^k$. Thus, unlike Porikli's method which only performs multiplication and addition operations, Chaudhury's method not only performs multiplication and addition operations, but also calculates the cosine function and the sine function. As is known to all, the floating point addition or subtraction requires 6 clock cycles, multiplication requires 8 clock cycles and division requires 30–44 clock cycles on the Intel ×86 processor. However, the cosine or sine function or exponential function requires between 180 and 280 clock cycles. Therefore, Chaudhury's method must pay for more running time. The extra cost is not a waste as it is spent on pursuing more accurate filtering results. Now, our question is whether we can reduce the extra cost as in Porikli's method while producing accurate results as in the method of Chaudhury. The answer is yes. Our method takes in the advantages of the two methods.

Proposed method: We employ the Hermite polynomials together with the exponential function to approximate $\mathrm{e}^{-\sigma_r(I_p-I_q)^2}$. The Hermite polynomials are a classical orthogonal polynomial sequence that arises in probability, combinatorics and physics. We list the first five terms in Table 1 as examples to give the flavour of the polynomials. More information can be found in [3] and references therein. To derive the approximation formula, we take advantage of the generating function. Specifically, the Hermite polynomials are given by the exponential generating function

$$\mathrm{e}^{(2xt-t^2)} = \sum_{i=0}^{\infty}\frac{H_i(x)}{i!}t^i \tag{10}$$

Here, we truncate the series and select the first M terms to approximate $\mathrm{e}^{(2xt-t^2)}$ as illustrated in (11)

$$\mathrm{e}^{(2xt-t^2)} = \sum_{i=0}^{M}\frac{H_i(x)}{i!}t^i \tag{11}$$

Then, by employing (11), $\mathrm{e}^{-\sigma_\mathrm{r}(I_p-I_q)^2}$ can be rewritten as

$$\begin{aligned}\mathrm{e}^{-\sigma_\mathrm{r}(I_p-I_q)^2} &= \mathrm{e}^{-\sigma_\mathrm{r}I_p^2}\mathrm{e}^{(2(\sqrt{\sigma_\mathrm{r}}I_p)(\sqrt{\sigma_\mathrm{r}}I_q)-(\sqrt{\sigma_\mathrm{r}}I_q)^2)} \\ &= \sum_{i=0}^{M}\mathrm{e}^{-\sigma_\mathrm{r}I_p^2}H_i(\sqrt{\sigma_\mathrm{r}}I_p)\frac{\sigma_\mathrm{r}^{(i/2)}}{i!}I_q^i \qquad (12)\end{aligned}$$

Furthermore, we plug (12) into (1) and obtain the value I_p^bf by calculating

$$\begin{aligned}I_p^\mathrm{bf} = &\frac{\mathrm{e}^{-\sigma_\mathrm{r}I_p^2}}{W_p^\mathrm{bf}}\sum_{i=0}^{M}\sum_{k=0}^{i}\binom{2i}{k}\frac{H_i(\sqrt{\sigma_\mathrm{r}}I_p)I_p^{i-k}\sigma_\mathrm{r}^{i/2}}{i!} \\ &\sum_{q\in\Omega_p}G_{\sigma_\mathrm{s}}(p-q)I_q^{k+1}\end{aligned} \qquad (13)$$

Once again, the inner sum $\sum_{q\in\Omega_p}$ of (13) is the linear convolution performed on the image I_q^{k+1} and the outer sum of (13) is the pointwise operations. A similar accelerating procedure can also be applied to the denominator W_p^bf of BF. We therefore conclude that the computational complexity of our accelerating algorithm is also $O(n)$.

Table 1: First five Hermite polynomials

0	$H_0(x) = 1$
1	$H_1(x) = 2x$
2	$H_2(x) = 4x^2 - 2$
3	$H_3(x) = 8x^3 - 12x$
4	$H_4(x) = 16x^4 - 48x^2 + 12$

Analysis and experiment: In this section, we implement the three algorithms by MATLAB and exhibit the computational resource consumption of the three methods in the memory limited system and the desktop personal computer (PC). In the memory limited system, all intermediate values need to be computed repeatedly because there is no redundant space to cache them. In (5), (9) and (13), the inner sum of the three algorithms takes two different operations. The first is the convolution that performs on the auxiliary images such as I^{k+1} and $C^{M-k}S^k$. The second is computing the auxiliary images. We only take the second into account, as the cost of the convolution operation of the three methods are the same. Since the inner sum of both Porikli's method and ours only involve the polynomial functions I^{k+1}, the running cost only depends on the upper bound of k. The bounds of Porikli's method and ours are $2M$ and M, respectively. By a simple calculation, we can find that the inner sums of (5) and (13) totally need $nM(2M+1)$ multiplications and $nM(M+1)/2$ multiplications. In contrast, the inner sum of (9) consumes $nM(M+3)$ multiplications plus nM cosine operations and nM sine operations. To calculate the running cost of the outer sum of (5) and (13), we rule out the cost of computing the coefficients in the outer sum as they can be precomputed. Indeed, the outer sum $\sum_{i=0}^{M}\sum_{k=0}^{2i}$ of (5) calculates the value of a polynomial with degree $2M$, then the running cost is $n(M(2M+3)+1)$ multiplications and $n(2M-1)$ additions. Unlike (5), the outer sum $\sum_{i=0}^{M}\sum_{k=0}^{i}$ of (13) can be transformed to an exponential function by a polynomial with degree M. The running cost is thus n exponential operations with $n(M(M+3)/2+2)$ multiplications and $n(M-1)$ additions. In contrast, the outer sum of (9) costs a lot as the operation computes a trigonometric polynomial with degree M. Therefore, the total running cost is $n(M(M+3)+1)$ multiplications, $n(M-1)$ additions, nM cosine operations and nM sin operations. For the convenience of comparison, we list the running cost of computing the numerator (1) in Table 2. The cost of calculating the denominator (2) is intentionally omitted as it is similar to Table 2 for the three methods. We can observe from Table 2 that the computational burden of Chaudhury's method is the heaviest. Although our method involves the exponential function, the method of Porikli will pay more running time on calculating the outer sum $\sum_{i=0}^{M}\sum_{k=0}^{2i}$. Thus, the overall running cost of the two methods is almost the same. Unlike the memory limited system, the desktop PC usually has adequate space to cache all intermediate results. If we could precompute the intermediate values and store them in the memory, lots of running time can be saved. By sticking to the idea, a common way used by Porikli and Chaudhury to accelerate the computing speed is to calculate the auxiliary images I^{k+1}, $C^{M-k}S^k$ and $IC^{M-k}S^k$ in advance. In this manner, we could easily eliminate the running cost of the inner sum and significantly reduce the costs of the outer sum. However, the expense is a huge memory consumption as shown in the 'mem' column of Table 3, whereas the three methods in the memory limited system only occupy n word memory. A closer look at Table 3 suggests that Porikli's method consumes the most computational resource. In contrast, the memory consumption of our method is only half of Chaudhury's method, while the running cost is only a little larger than Chaudhury's best method.

Table 2: Running cost in memory limited system

	Computational complexity of inner sum operation				
	add	mul	exp	sin	cos
Porikli [2]	0	$nM(2M+1)$	0	0	0
Chau [1]	0	$nM(M+3)$	0	nM	nM
Ours	0	$nM(M+1)/2$	0	0	0
	Computational complexity of outer sum operation				
	add	mul	exp	sin	cos
Porikli [2]	$n(2M-1)$	$n(M(2M+3)+1)$	0	0	0
Chau [1]	$n(M-1)$	$n(M(M+3)+1)$	0	nM	nM
Ours	$n(M-1)$	$n(M(M+3)/2+2)$	n	0	0

Table 3: Running cost and memory consumption in desktop PC

	Running cost of outer sum					
	add	mul	exp	sin	cos	mem
Porikli [2]	nM	$nM(2M+1)$	0	0	0	$n(2M+1)$
Chau [1]	nM	$nM(M+1)/2$	0	0	0	$2nM$
Ours	nM	$n(M(M+1)+2)/2$	0	0	0	$n(M+2)$

To illustrate the tradeoff between speed and accuracy, we conduct two experiments. First, we fix $M = 5$ to compare the filtering results of three accelerating methods with the result of the brute-force implementation. The peak signal-to-noise ratio (PSNR) is employed to measure the similarity. The higher score implies the more accurate result. The frames per second (FPS) is exploited to measure the processing speed. Similarly, a larger index means a higher speed. The statistical data is reported in Table 4, where the size of the test image is 256×256. In the second experiment, we keep the PSNR index unchanged and measure the speed of each method. Strictly speaking, the PSNR indices cannot be kept the same as they are affected by various reasons. Here, we tweak M to keep them as close to the same as possible and list the statistical data in Table 4. From the table, we can learn that our method takes in the advantages of the two methods of Porikli and Chaudhury. The speed approaches that of Porikli's method while the approximation accuracy is nearly the same as Chaudhury's method.

Table 4: Tradeoff between speed and accuracy

	Fixed $M = 5$			Fixed PSNR		
	Ours	Porikli [2]	Chau [1]	Ours	Porikli	Chau
PSNR	43.26	42.95	43.32	50.26	50.29	50.25
FPS	6.3	6.5	2.6	2.9	2.3	1.4

Conclusion: In this Letter, we have proposed a novel method to accelerate the BF using the well-known Hermite polynomials. The running cost and memory consumption are almost equal to Porikli's method. More importantly, the approximation accuracy is similar to Chaudhury's method.

Acknowledgments: This work was partially supported by the National Natural Science Foundation of China (61331018, 61332017, 91338202, and 61271430), and partly by the open funding project of the State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (no. BUAA-VR-14KF-10).

© The Institution of Engineering and Technology 2014

31 July 2014
doi: 10.1049/el.2014.2758

Longquan Dai, Mengke Yuan and Xiaopeng Zhang (*NLPR, Institute of Automation Chinese Academy of Sciences, Beijing, People's Republic of China*)

E-mail: lqdai@nlpr.ia.ac.cn

References

1 Chaudhury, K.N., Sage, D., and Unser, M.: 'Fast o(1) bilateral filtering using trigonometric range kernels', *IEEE Trans. Image Process.*, 2011, **20**, (12), pp. 3376–3382

2 Porikli, F.: 'Constant time o(1) bilateral filtering'. IEEE Conf. on Computer Vision and Pattern Recognition, Anchorage, AK, USA, June 2008, pp. 1–8

3 Temme, N.M.: 'Special functions: an introduction to the classical functions of mathematical physics' (J. Wiley & Sons, New York, USA, 1996), ISBN 0-471-11313-1

4 Tomasi, C., and Manduchi, R.: 'Bilateral filtering for gray and color images'. Int. Conf. on Computer Vision, Bombay, India, January 1998, pp. 839–846

5 Young, I.T., and van Vliet, L.J.: 'Recursive implementation of the Gaussian filter', *Signal Process.*, 1995, **44**, (2), pp. 139–151, ISSN 0165-1684