

FAST AND ACCURATE BILATERAL FILTERING USING GAUSS-POLYNOMIAL DECOMPOSITION

Kunal N. Chaudhury

Department of Electrical Engineering, Indian Institute of Science, Bangalore, India

ABSTRACT

The bilateral filter is a versatile non-linear filter that has found diverse applications in image processing, computer vision, computer graphics, and computational photography. A common form of the filter is the Gaussian bilateral filter in which both the spatial and range kernels are Gaussian. A direct implementation of this filter requires $O(\sigma^2)$ operations per pixel, where σ is the standard deviation of the spatial Gaussian. In this paper, we propose an accurate approximation algorithm that can cut down the computational complexity to $O(1)$ per pixel for any arbitrary σ (constant-time implementation). This is based on the observation that the range kernel operates via the translations of a fixed Gaussian over the range space, and that these translated Gaussians can be accurately approximated using the so-called Gauss-polynomials. The overall algorithm emerging from this approximation involves a series of spatial Gaussian filtering, which can be efficiently implemented (in parallel) using separability and recursion. We present some preliminary results to demonstrate that the proposed algorithm compares favorably with some of the existing fast algorithms in terms of speed and accuracy.

Index Terms— Bilateral filter, approximation, Gauss-polynomial, convolution, fast algorithm.

1. INTRODUCTION

The bilateral filter of Tomasi and Maduchi [1] is a particular instance of an edge-preserving smoothing filter. The origins of the filter can be traced back to the work of Lee [2] and Yaroslavsky [3]. The SUSAN framework of Smith and Brady [4] is also based on a similar idea. The relation between the bilateral and other closely related filters is surveyed in [5]. The bilateral filter has turned out to be a versatile tool that has found widespread applications in image processing, computer graphics, computer vision, and computational photography. A detailed survey of some of these applications can be found in [6]. More recently, the bilateral filter has received renewed attention in the context of image denoising [7, 8]. The original bilateral filter [1] has a straightforward extension to signals of arbitrary dimension and, in particular, to video and volume data [6]. Thus, while we will limit our discussion to images in this paper, the ideas that we present next can also be extended to higher-dimensional signals.

Consider a discrete image $\{f(\mathbf{i}) : \mathbf{i} \in I\}$ where I is some finite rectangular domain of \mathbb{Z}^2 . The Gaussian bilateral filtering of this image is given by

$$f_{\text{BF}}(\mathbf{i}) = \frac{\sum_{\mathbf{j} \in \Omega} g_{\sigma_s}(\mathbf{j}) g_{\sigma_r}(f(\mathbf{i} - \mathbf{j}) - f(\mathbf{i})) f(\mathbf{i} - \mathbf{j})}{\sum_{\mathbf{j} \in \Omega} g_{\sigma_s}(\mathbf{j}) g_{\sigma_r}(f(\mathbf{i} - \mathbf{j}) - f(\mathbf{i}))}, \quad (1)$$

K. N. Chaudhury was partially supported by a Startup Grant provided by the Indian Institute of Science. Correspondence: kunal@ee.iisc.ernet.in.

where both the *spatial* and *range* kernels are Gaussian,

$$g_{\sigma_s}(\mathbf{i}) = \exp\left(-\frac{\|\mathbf{i}\|^2}{2\sigma_s^2}\right) \quad \text{and} \quad g_{\sigma_r}(t) = \exp\left(-\frac{t^2}{2\sigma_r^2}\right). \quad (2)$$

In practice, the domain of the spatial kernel Ω is restricted to some neighbourhood of the origin. Typically, Ω is a square neighbourhood, $\Omega = [-W, W] \times [-W, W]$ where $W = 3\sigma_s$ [1]. We refer the reader to [1, 6] for a detailed exposition on the working of the filter.

1.1. Fast Bilateral Filter

It is clear that a direct implementation of (1) requires $O(\sigma_s^2)$ operations per pixel. In general, the directly computed bilateral filter is slow for practical settings of σ_s [6]. To address this issue, researchers have come up with several fast algorithms [9, 10, 11, 12, 13, 14, 15, 16] that are based on some form of approximation and yield various levels of speed and accuracy. We refer the interested reader to [6, 15] for a survey of algorithms for fast bilateral filtering. The ultimate goal in this regard is to reduce the complexity to $O(1)$ per pixel, that is, the run time of the implementation should not depend on σ_s . This is commonly referred to as a constant-time implementation. The constant-time algorithms in [12, 16, 17, 18] are particularly relevant to the present work. The authors here proceed by approximating the Gaussian range kernel using polynomial and trigonometric functions, and demonstrate how the bilateral filter can be decomposed into a series of spatial Gaussian filters as result. Now, since a Gaussian filter can be implemented in constant-time using separability and recursion [19], the overall approximation can therefore be computed in constant-time.

1.2. Present Contribution

In this paper, we propose a fast $O(1)$ algorithm for computing (1) which is motivated by the line of work in [12, 16, 17, 18]. In particular, we present a novel approximation for the range term in (1) that allows us to decompose the bilateral filter into a series of fast Gaussian convolutions. The fundamental difference between the above papers and the present approach is that instead of approximating the Gaussian and then translating the approximation, we directly approximate the translated Gaussians using the so-called Gauss-polynomials. The advantages of the proposed approximation are the following:

- It is generally much more accurate than the polynomial approximation in [12].
- For a fixed approximation degree (to be defined shortly), it leads to exactly half the number of Gaussian filterings than that required by the approximation in [16, 18], and hence has a smaller run time.
- It does not involve transcendental functions such as $\cos(\omega x)$ and $\sin(\omega x)$ which are used in [16, 18]. It only involves polynomials (and just a single Gaussian) and hence can be efficiently implemented on hardware [20, 21]. This is partly what motivated the present work.

Moreover, we also show how the proposed approximation can be improved by first centering the range data, then applying the approximation algorithm, and finally adding back the centre to the processed range data.

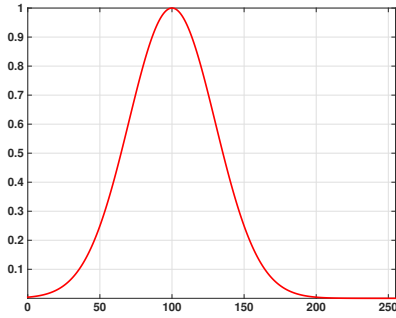
2. GAUSS-POLYNOMIAL DECOMPOSITION

The main idea in [12, 16] was to approximate the range kernel in (2) using appropriate polynomials and trigonometric functions. By using these approximations in place of the Gaussian kernel, it was shown that the numerator and denominator of (1) can be approximated using a series of Gaussian filtering.

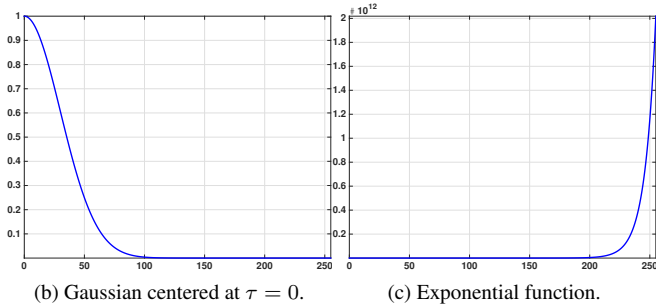
The present idea is to consider the translates of the range kernel $g_{\sigma_r}(t - \tau)$ that appear in (1), where $t = f(i - j)$ and $\tau = f(i)$ take values in some intensity range, say, $[L, U]$. For example, $L = 0$ and $U = 255$ for an 8-bit grayscale image. We can write

$$g_{\sigma_r}(t - \tau) = \exp\left(-\frac{\tau^2}{2\sigma_r^2}\right) \exp\left(-\frac{t^2}{2\sigma_r^2}\right) \exp\left(\frac{\tau t}{\sigma_r^2}\right). \quad (3)$$

For a fixed translation τ , this is a function of t . Notice that the first term is simply a scaling factor, while the second term is a Gaussian centered at the origin. In fact, the second term essentially contributes to the ‘‘bell’’ shape of the translated Gaussian. The third term is a monotonic exponential function which is increasing or decreasing depending on the sign of τ . This term helps in translating the Gaussian to $t = \tau$. The decomposition in (3) plays an important role in the rest of the discussion and is illustrated in Figure 1.



(a) Gaussian centered at $\tau = 100$.



(b) Gaussian centered at $\tau = 0$.

(c) Exponential function.

Fig. 1. An instance of the decomposition in (3) corresponding to $\tau = 100$ and $\sigma_r = 30$. We used $L = 0$ and $U = 255$ corresponding to the intensity range of an 8-bit grayscale image. Up to a scaling factor of about $4e-3$, (a) is a product of (b) and (c).

Consider the Taylor series of the exponential function,

$$\exp\left(\frac{\tau t}{\sigma_r^2}\right) = \sum_{n=0}^N \frac{1}{n!} \left(\frac{\tau t}{\sigma_r^2}\right)^n + \text{higher-order terms}. \quad (4)$$

By dropping the higher-order terms, we obtain the following approximation of (3):

$$\exp\left(-\frac{\tau^2}{2\sigma_r^2}\right) \exp\left(-\frac{t^2}{2\sigma_r^2}\right) \left[\sum_{n=0}^N \frac{1}{n!} \left(\frac{\tau t}{\sigma_r^2}\right)^n \right]. \quad (5)$$

Being the product of a Gaussian and a polynomial of degree N , we will henceforth refer to (5) as a ‘‘Gauss-polynomial’’ of degree N .

At this point, we note that one of the proposals in [12] was to approximate $g_{\sigma_r}(t - \tau)$ using its Taylor polynomial. The fundamental difference with our approach is that instead of approximating the entire Gaussian, we approximate one of its component, namely the monotonic exponential in (3). The intuition behind this is that a polynomial eventually goes to infinity as one moves away from the origin. This makes it difficult to approximate the Gaussian function on its asymptotically-decaying tails. As against this, the exponential function in (3) is monotonic and hence can be more accurately approximated using polynomials. This is explained with an example in Figure 2. In particular, notice in Figure 2 (b) that the Gauss-polynomial approximation is fairly accurate over the entire range of interest and is comparable to the raised-cosine approximation [16].

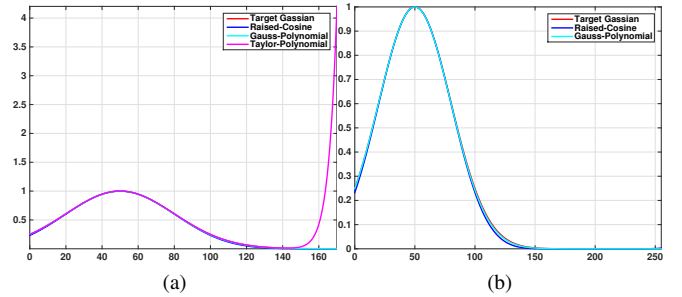


Fig. 2. Approximation of $g_{\sigma_r}(t - \tau)$ where $\sigma_r = 30$ and $\tau = 10$. The raised-cosine [16], the Taylor polynomial [12], and the Gauss-polynomial have the same degree $N = 10$. In subfigure (a), notice how the Taylor polynomial quickly goes off to $+\infty$ as one moves away from the origin. For this reason, we restricted the plot to the interval $[0, 170]$, although the desired approximation range is the full dynamic range $[0, 255]$. The approximation over $[0, 255]$ for the raised-cosine and the Gauss-polynomial are separately provided in subfigure (b).

We note that for a fixed degree N , the accuracy of the Gauss-polynomial approximation in (5) depends on τ . Indeed, when $\tau = 0$, there is nothing to approximate since the exponential function reduces to a constant in this case. On the other hand, we see from (5) that the magnitude of the higher-order terms increases with increase in $|\tau|$, and the approximation accuracy drops as a result. This is demonstrated with an example in Figure 3 (a). Of course, the approximation can be improved by using Gauss-polynomials of higher degree. However, we note that for a fixed degree, the approximation accuracy can be improved simply by reducing the maximum $|\tau|$ that appears in (5). We propose the following ‘‘centering’’ trick in which we set, for example,

$$t_c = \text{mean} \{f(i) : i \in I\} = \frac{1}{|I|} \sum_{i \in I} f(i).$$

We next translate each pixel intensity by t_c , which has the effect of centering the transformed intensity range at the origin. For example,

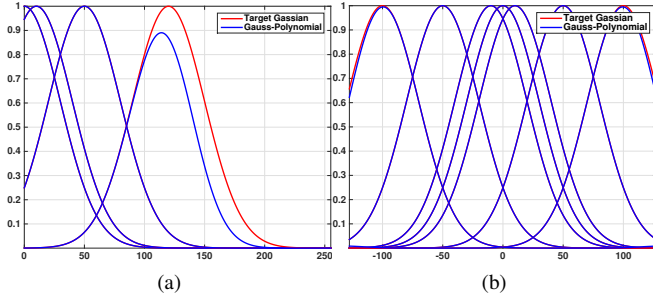


Fig. 3. (a) Approximation of $g_{\sigma_r}(t-\tau)$, $\sigma_r = 30$, over $[0, 255]$ using Gauss-polynomials of degree 20. We vary τ over 0, 10, 50, 120. (b) Approximation over the centered interval $[-255/2, 255/2]$, where we vary τ over $-100, -50, 0, 10, 50, 100$. This demonstrates how the approximation accuracy can be improved by simply centering the range interval at the origin.

when $L = 0$ and $U = 255$, the maximum $|\tau|$ equals 255. However, if we center the intensity range $[0, 255]$, say at $t_c = 100$, then we can effectively reduce the maximum $|\tau|$ to 155. The Gauss-polynomial approximations obtained after the centering are shown in Figure 3 (b). The above idea of centering is compatible with the bilateral filter precisely because of the following property of the bilateral filter.

Proposition 2.1 *If $h(\mathbf{i}) = f(\mathbf{i}) - t_c$, then*

$$f_{\text{BF}}(\mathbf{i}) = h_{\text{BF}}(\mathbf{i}) + t_c \quad (\mathbf{i} \in I). \quad (6)$$

This is a simple consequence of the fact that the range kernel depends only on the intensity difference, and that for a fixed range term, (1) preserves constant functions. In other words, we can first centre the intensity range, apply the bilateral filter, and finally add back the centre to the output.

3. FAST BILATERAL FILTER

We now present the constant-time implementation of (1) using Gauss-polynomials. Suppose that N is the degree of the polynomial in (5). For $n = 0, \dots, N+1$, define the images

$$G_n(\mathbf{i}) = \left(\frac{f(\mathbf{i})}{\sigma_r}\right)^n \quad \text{and} \quad F_n(\mathbf{i}) = \exp\left(-\frac{f(\mathbf{i})^2}{2\sigma_r^2}\right) G_n(\mathbf{i}). \quad (7)$$

Denote the Gaussian filtering of $F_n(\mathbf{i})$ by $\bar{F}_n(\mathbf{i})$, that is,

$$\bar{F}_n(\mathbf{i}) = (F_n * g_{\sigma_s})(\mathbf{i}) = \sum_{\mathbf{j} \in \Omega} g_{\sigma_s}(\mathbf{j}) F_n(\mathbf{i} - \mathbf{j}). \quad (8)$$

Substituting $t = f(\mathbf{i} - \mathbf{j})$ and $\tau = f(\mathbf{i})$, and using the Gauss-polynomial approximation (5) in place of $g_{\sigma_r}(t-\tau)$, it can be verified that (after interchanging summations) we can express the numerator of (1) as

$$\exp\left(-\frac{f(\mathbf{i})^2}{2\sigma_r^2}\right) P(\mathbf{i}),$$

where

$$P(\mathbf{i}) = \sigma_r \sum_{n=0}^N \frac{1}{n!} G_n(\mathbf{i}) \bar{F}_{n+1}(\mathbf{i}). \quad (9)$$

Similarly, we can express the denominator of (1) as

$$\exp\left(-\frac{f(\mathbf{i})^2}{2\sigma_r^2}\right) Q(\mathbf{i}),$$

where

$$Q(\mathbf{i}) = \sum_{n=0}^N \frac{1}{n!} G_n(\mathbf{i}) \bar{F}_n(\mathbf{i}). \quad (10)$$

In other words, we can approximate (1) by

$$\hat{f}_{\text{BF}}(\mathbf{i}) = P(\mathbf{i})/Q(\mathbf{i}). \quad (11)$$

Note that we have effectively transferred the non-linearity of the bilateral filter to the intermediate images in (7), which are obtained from the input image using pointwise non-linear transforms. The main leverage that we get from the above manipulation is that, for any arbitrary σ_s , (8) can be computed using $O(1)$ operations per pixel [19]. The overall cost of computing (11) is therefore $O(1)$ per pixel. In other words, we have a constant-time approximation of the bilateral filter. In this regard, we note that the above analysis holds if we replace the spatial Gaussian filter by any other filter (e.g., a box filter) that has a constant-time implementation.

Data: Image $\{f(\mathbf{i}) : \mathbf{i} \in I\}$, and parameters σ_s, σ_r, N .

Result: Approximation $\{\hat{f}_{\text{BF}}(\mathbf{i}) : \mathbf{i} \in I\}$.

```

1  $t_c = \text{mean}\{f(\mathbf{i}) : \mathbf{i} \in I\}$ ;
2  $h(\mathbf{i}) = f(\mathbf{i}) - t_c$ ;
3  $G(\mathbf{i}) = 1$ ;
4  $F(\mathbf{i}) = \exp(-h(\mathbf{i})^2/2\sigma_r^2)$ ;
5  $P(\mathbf{i}) = 0$ ;
6  $Q(\mathbf{i}) = 0$ ;
7  $H(\mathbf{i}) = h(\mathbf{i})/\sigma_r$ ;
8  $\bar{F}(\mathbf{i}) = (F * g_{\sigma_s})(\mathbf{i})$ ;
9  $c = 1$ ;
10 for  $n = 0, 1, \dots, N$  do
11    $Q(\mathbf{i}) = Q(\mathbf{i}) + c \cdot G(\mathbf{i}) \bar{F}(\mathbf{i})$ ;
12    $F(\mathbf{i}) = H(\mathbf{i}) F(\mathbf{i})$ ;
13    $\bar{F}(\mathbf{i}) = (F * g_{\sigma_s})(\mathbf{i})$ ;
14    $P(\mathbf{i}) = P(\mathbf{i}) + c \cdot G(\mathbf{i}) \bar{F}(\mathbf{i})$ ;
15    $G(\mathbf{i}) = H(\mathbf{i}) G(\mathbf{i})$ ;
16    $c = c/(n+1)$ ;
17 end
18  $\hat{f}_{\text{BF}}(\mathbf{i}) = \sigma_r (P(\mathbf{i})/Q(\mathbf{i})) + t_c$ ;

```

Algorithm 1: Gauss-Polynomial Bilateral Filter (GPF).

The overall algorithm is summarized in Algorithm 1. We will henceforth refer to this as the Gauss-Polynomial-based Bilateral Filter (GPF). Notice that we use centering and (6) to improve the accuracy. Moreover, we efficiently implement steps (7) to (10). In particular, we recursively compute the images in (7) and the factorials in (9) and (10). Notice that steps 2-7, 11-12, 14, 15, and 18 are applied to each pixel (cheap pointwise operations). To avoid confusion, we note that the specification of the some of quantities in Algorithm 1 are somewhat different from the corresponding definitions in (7) - (11).

It is clear that the main computations in GPF are the Gaussian filterings in step 13 (and the initial filtering in step 8). That is, the overall cost is dominated by the cost of computing $N+1$ Gaussian filterings. In this regard, we note that for the same degree N , the number of Gaussian filterings required in [16] is $4(N+1)$. Indeed, we will see in Section 4 for a fixed N , the overall run-time of GPF is

about a third of that of [16]. Finally, we note that GPF involves the evaluation of a transcendental function just once in step 4. Thus, GPF is better suited to hardware implementation [20, 21] compared to the algorithm in [16] which involves the repeated evaluation of cosine and sine functions.

4. EXPERIMENTS

We now present some results concerning the accuracy and run-time of the proposed GPF algorithm. In particular, we compare it with some of the fast algorithms [10, 12, 13, 18]. The experiments were performed using Matlab on an Intel quad-core 2.7 GHz machine with 8 GB memory. We implemented the Gaussian filtering in GPF and [13, 18] using Deriche’s constant-time algorithm [19]. The average run-times of the various fast algorithms are reported in Table 1 for a 256×256 . We do not redundantly report the run-times for different image sizes, since this can roughly be estimated from the run-times in Table 1 (the algorithms scale linearly with the number of pixels). Notice that the run time of GPF and [13, 18] does not change appreciably with σ_s . To evaluate the accuracy, we also report the mean-squared-error (MSE) between the exact implementation of (1) and the result obtained using the fast algorithms in Table 1. In particular, the MSE between two images $f(i)$ and $g(i)$ is defined to be $10 \log_{10}(\text{MSE})$ dB, where $\text{MSE} = (1/|I|) \sum_{i \in I} (f(i) - g(i))^2$. Notice that GPF is competitive with the existing algorithms in terms of accuracy and run-time. In particular, GPF has the smallest run-time, and its MSE is in general better than the rest of the algorithms and comparable to that of the raised-cosine-based approximation in [18]. The degree of the raised-cosine and the Gauss-polynomial filter is 20 for all the experiments (this gives a good tradeoff between accuracy and run-time). In this regard, an open question is how the accuracy of GPF varies with the degree, as a function of σ_s and σ_r . This will be addressed in future work. Note that the run-time of the polynomial approximation in [12] is almost identical to that of the proposed algorithm and is hence not reported. For a visual comparison, we report the result obtain on the *Peppers* image in Figure 4. Notice the visible distortions in subfigure (d) which arises on account of the poor approximation of the Gaussian kernel using Taylor polynomials.

Table 1. The top table comparison of the average run-time of the exact, the proposed, and the fast algorithms in [10, 13, 18] for different values of σ_s and fixed $\sigma_r = 30$. We used the *Peppers* image shown in Figure 4. The settings suggested in [10, 13, 18] were used for the corresponding implementations. The algorithms were implemented using Matlab on an Intel quad-core 2.7 GHz machine with 8 GB memory. The bottom table compares the MSE between the exact implementation of (1) and the respective algorithms.

		Run-Time					
		2	3	4	5	10	15
Exact		1.5s	3.2s	5.3s	8.4s	32.5s	73.2s
[10]		93ms	134ms	191ms	261ms	847ms	1.92s
[13]		112ms	118ms	115ms	116ms	118ms	120ms
[18]		210ms	215ms	220ms	225ms	230ms	250ms
GPF		74 ms	82ms	88ms	89ms	95ms	98ms
		MSE (dB)					
[10]		5.9	7.8	9.1	9.8	12.2	13.1
[13]		-3.3	-1.1	0.5	1.8	6.2	9.2
[18]		-10.5	-6.4	-3.8	-1.7	4.4	7.8
GPF		-9.6	-5.6	-3.1	-1.1	5.1	8.4



Fig. 4. Comparison of the exact implementation of (1) with various fast algorithms for $\sigma_s = 3, \sigma_r = 30$. The degree of the respective functions (raised-cosine, Taylor polynomial, and Gauss-polynomial) used to approximate the range kernel is 20. The MSE between the exact implementation and the various approximations are shown in bold.

5. CONCLUSION

We presented a fast algorithm for bilateral filtering based on Gauss-polynomial decompositions of the translations of the range kernel. We presented some preliminary results which demonstrated the accuracy and speed of the algorithm in comparison to some of the existing fast algorithms for bilateral filtering. In particular, we saw that the algorithm is comparable to the one in [18], but has a smaller run time (about a third). Moreover, as remarked in the introduction, the proposed algorithm has an edge over [18] in the context of hardware implementation – it is based on polynomials and does not involve the computation of multiple trigonometric functions [21]. We note that the algorithm has a direct extension to other variants of the bilateral filter including the joint and guided filter [22, 23], and can also be extended for handling volume and video data [15].

6. REFERENCES

- [1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *Proc. IEEE International Conference on Computer Vision*, pp. 839-846, 1998.
- [2] J. S. Lee, "Digital image smoothing and the sigma filter," *Computer Vision, Graphics, and Image Processing*, vol. 24, no. 2, pp. 255-269, 1983.
- [3] L. P. Yaroslavsky, *Digital Picture Processing*. Secaucus, NJ: Springer-Verlag, 1985.
- [4] S. M. Smith and J. M. Brady, "SUSAN - A new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45-78, 1997.
- [5] M. Elad, "On the origin of the bilateral filter and ways to improve it," *IEEE Transactions on Image Processing*, vol. 11, no. 10, pp. 1141-1151, 2002.
- [6] P. Kornprobst and J. Tumblin, *Bilateral Filtering: Theory and Applications*. Now Publishers Inc., 2009.
- [7] C. Knaus and M. Zwicker, "Progressive Image Denoising," *IEEE Transactions on Image Processing*, vol. 23, no.7, pp. 3114-3125, 2014.
- [8] N. Pierazzo, M. Lebrun, M. E. Rais, J. M. Morel, and G. Facciolo, "Non-local dual denoising," *Proc. IEEE International Conference on Image Processing*, 2014.
- [9] 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.
- [10] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Proc. European Conference on Computer Vision*, pp. 568-580, 2006.
- [11] B. Weiss, "Fast median and bilateral filtering," *Proc. ACM Siggraph*, vol. 25, pp. 519-526, 2006.
- [12] F. Porikli, "Constant time $O(1)$ bilateral filtering," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [13] Q. Yang, K. H. Tan, and N. Ahuja, "Real-time $O(1)$ bilateral filtering," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 557-564, 2009.
- [14] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian kd-trees for fast high-dimensional filtering," *Proc. ACM Siggraph*, 2009.
- [15] A. Adams, J. Baek, and A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," *Proc. Euro Graphics*, 2010.
- [16] 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.
- [17] K. N. Chaudhury, "Constant-time filtering using shiftable kernels," *IEEE Signal Processing Letters*, vol. 18, no. 11, pp. 651-654, 2011.
- [18] K. N. Chaudhury, "Acceleration of the shiftable algorithm for bilateral filtering and nonlocal means," *IEEE Transactions on Image Processing*, vol. 22, no. 4, pp. 1291-1300, 2013.
- [19] R. Deriche, "Recursively implementing the Gaussian and its derivatives," *Research Report*, INRIA-00074778, 1993.
- [20] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. John Wiley and Sons, 2011.
- [21] J. M. Muller, *Elementary Functions: Algorithms and Implementation*. Birkhauser Boston, 2006.
- [22] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397-1409, 2013.
- [23] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 664-672, 2004.