Accelerated Bilateral Filtering with Block Skipping

Chao Tian, *Senior Member, IEEE*, and Shankar Krishnan

***Abstract*—We propose a method to accelerate Yang's real-time $O(1)$ bilateral filtering algorithm, based on the observation that in the original algorithm, some of the computation can be strategically eliminated. To identify such computation, the algorithm steps are analyzed in conjunction with its recursive Gaussian filtering component. By block partitioning the image, the procedure to isolate these unnecessary computation is simplified, and the proposed algorithm only needs to skip some of the image blocks when performing recursive linear filtering. The resultant accelerated algorithm is able to achieve 1.5$\sim$5 times speedup, depending on the image statistics and the filtering parameters. The proposed algorithm only marginally degrades the accuracy of the filtering, and the simplicity and small memory footprint of Yang's original algorithm are largely maintained.**

***Index Terms*—Linear filter, linear interpolation.**

I. Introduction

The bilateral filter was introduced by Tomasi and Manduchi [1] as an edge preserving smoothing filter. Unlike the Gaussian filter which simply smooths in a spatially invariant manner, the bilateral filter has a factor which controls the smoothing using the intensity (color) domain information. This added control enables the bilateral filtering to effectively preserve the strong edges while smoothing away the undesirable noises, leading to many applications such as image denoising [2], tone mapping [3], flash/no-flash fusion [4] and stereo matching [5].

The benefit of edge preserving smoothing is not without its cost. The bilateral filter is essentially a non-linear filter, and thus is computationally expensive, usually requiring a naive implementation several minutes to filter a typical megapixel image. Over the years, several methods have been developed to approximate and accelerate the computation of bilateral filtering [3], [6]–[9], and as a result we have seen several orders of magnitude speedup.

Among the fastest existing bilateral filtering algorithms is the *real-time $O(1)$ bilateral filtering* algorithm recently proposed by Yang et al. [8], where $O(1)$ refers to a constant factor of computation per image pixel. This algorithm generates the filtered image by first computing the Principle Bilateral Filtered Image Component (PBFIC) through recursively linearly filtering two intermediate images, then linearly interpolating between two adjacent PBFICs. The main step in computing the PBFICs is spatial invariant linear filtering, which is well understood and much less expensive computationally than non-linear filtering. This algorithm is able to produce very accurate bilateral filtered result with a small memory footprint.

In this letter, we provide a further improvement of Yang's algorithm, based on the observation that in Yang's original algorithm, some of the computation for the PBFICs are wasteful. To reduce such wasteful computations, we carefully analyze the computation steps in conjunction with the recursive Gaussian filtering procedure to isolate them, and skip some of the image blocks to avoid a large portion of such computation. The resultant algorithm is able to achieve 1.5 to 5 times speed up, depending on the image statistics and the filtering parameters. The proposed improvement only marginally degrades the accuracy of the filtering, and at the same time the simplicity and small memory footprint of Yang's original algorithm are largely maintained.

C. Tian and S. Krishnan are with AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932 (email: {tian,krishnas}@research.att.com).

II. Bilateral Filtering and Yang's Algorithm

A. Bilateral Filtering

Mathematically, the bilateral filtering operation performs the following computation for each pixel $\vec{x}$ in an image

$$I^B(\vec{x}) = \frac{\sum_{\vec{y}\in N(\vec{x})} f_S(\vec{x},\vec{y}) \cdot f_R(I(\vec{x}),I(\vec{y})) \cdot I(\vec{y})}{\sum_{\vec{y}\in N(\vec{x})} f_S(\vec{x},\vec{y}) \cdot f_R(I(\vec{x}),I(\vec{y}))}, \quad (1)$$

where $I(\vec{x})$ is the intensity value at pixel location $\vec{x}$, $N(\vec{x})$ is a specific neighborhood of $\vec{x}$, $f_S(\vec{x},\vec{y})$ is the spatial filtering kernel depending only on the pixel locations $\vec{x}$ and $\vec{y}$, and $f_R(I(\vec{x}),I(\vec{y}))$ is the range filtering kernel depending only on the intensity $I(\vec{x})$ and $I(\vec{y})$. The original bilateral filtering uses Gaussian-like filter of different variances for the spatial filtering kernel $f_S(\vec{x},\vec{y})$ and range filtering kernel $f_R(I(\vec{x}),I(\vec{y}))$,

$$f_S(\vec{x},\vec{y}) = \exp\big(-||\vec{x}-\vec{y}||^2/2\sigma_S^2\big), \quad (2)$$

$$f_R(I(\vec{x}),I(\vec{y})) = \exp\big(-(I(\vec{x})-I(\vec{y}))^2/2\sigma_R^2\big), \quad (3)$$

where σ_S and σ_R are the spatial kernel variance and the range kernel variance, respectively. The kernel functions $f_S(\vec{x},\vec{y})$ and $f_R(I(\vec{x}),I(\vec{y}))$ have since been generalized to other filters, e.g., to the box-filter spatial kernel.

B. Yang's Real-Time $O(1)$ Algorithm

Yang's real-time $O(1)$ bilateral filtering algorithm can be described as follows. For a digital image, there are only a finite number of possible intensity values, i.e., $I(\vec{x}) \in \{0,1,\ldots,N-1\}$. For each intensity value $k \in \{0,1,\ldots,N-1\}$ in a given image, the following two quantities are only functions of the pixel location $\vec{y}$

$$W_k(\vec{y}) \triangleq f_R(k,I(\vec{y})), \quad J_k(\vec{y}) \triangleq W_k(\vec{y}) \cdot I(\vec{y}). \quad (4)$$

Now (1) can be written as

$$I^B(\vec{x}) = I^B_{I(\vec{x})}(\vec{x}), \quad (5)$$

where by denoting $I(\vec{x}) = k$

$$I^B_k(\vec{x}) \triangleq \frac{J'_k(\vec{x})}{W'_k(\vec{x})} = \frac{\sum_{\vec{y}\in N(\vec{x})} f_S(\vec{x},\vec{y}) \cdot J_k(\vec{y})}{\sum_{\vec{y}\in N(\vec{x})} f_S(\vec{x},\vec{y}) \cdot W_k(\vec{y})}. \quad (6)$$

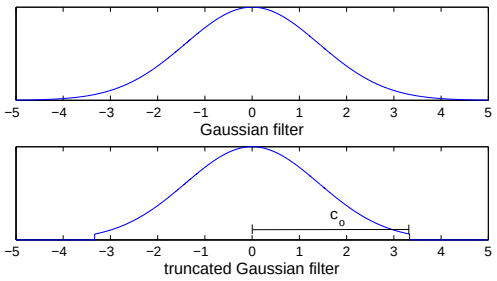

Fig. 1. The Gaussian filter and its truncated version.

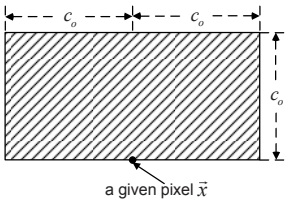

Fig. 2. The coherent region for the forward vertical pass.

Both the numerator and the denominator in (6) are spatial-invariant linear Gaussian filtering, which can be computed efficiently using the recursive method in [10], [11]. This gives the bilateral filtered value for pixels of intensity k. By sweeping through all possible intensities, the filtered value for each pixel can be determined. Each of these $I_k^B(\vec{x})$ images is called a Principle Bilateral Filtered Image Component (PBFIC).

The algorithm in [8] is built on the observation that there is no need to compute PBFIC for all the intensity levels, but instead only PBFICs for $K < N$ levels (denoted as $\{L_0, L_1, \ldots, L_{K-1}\}$ where L_0 and L_{K-1} are the lower and upper limits of the dynamic range, respectively) need to be computed; for notational simplicity, we shall write them as $\text{PBFIC}(L_0), \text{PBFIC}(L_1), \ldots, \text{PBFIC}(L_{K-1})$. The final bilateral filtered value $I^B(\vec{x})$ at location $\vec{x}$ can then be linearly interpolated using the values at location $\vec{x}$ of $\text{PBFIC}(L_{i-1})$ and $\text{PBFIC}(L_i)$ for any pixel $\vec{x}$ such that $I(\vec{x}) \in [L_{i-1}, L_i)$, i.e., using the linear combination of them with weights $\frac{L_i - I(\vec{x})}{L_i - L_{i-1}}$ and $\frac{I(\vec{x}) - L_{i-1}}{L_i - L_{i-1}}$, respectively. An advantage of this algorithm is its small memory footprint, because one could sweep through $\{L_0, L_1, \ldots, L_{K-1}\}$, and only two adjacent PBFICs need to be kept such that for any pixels whose intensities fall into this range, their filtered values can be correctly obtained.

Since the Gaussian filter is separable, the overall process can be decomposed into horizontal and vertical filtering. The one dimensional recursive algorithm of Deriche [10], [11] includes a forward pass and a backward pass, each one of which is a linear filtering using an infinite impulse response (IIR) filter, and the combination of these two passes leads to an extremely accurate approximation of the Gaussian smoothing.

III. The Accelerated Algorithm

A. Motivation

Consider the case where the image pixel intensity values are all in the range of $[L_0, L_1)$. When Yang's algorithm is used on this image, since the filtered result $I^B(\vec{x})$ at each pixel is completely determined by $\text{PBFIC}(L_0)$ and $\text{PBFIC}(L_1)$, the other PBFICs are useless, and thus computing them is unnecessary and wasteful. This simple observation is the main motivation for the improvement proposed in this work.

B. Necessary Computations and Unnecessary Computations

The extreme case example discussed above may leave the impression that for a given pixel $\vec{x}$, we only need to compute the two levels of PBFICs at those locations $\vec{x}$ where $I(\vec{x})$ is sandwiched in between. This is however not the case when the recursive filtering method is used.

For simplicity, let us consider the one-dimensional case, and assume the IIR filter is given as

$$y[n] = a_0 x[n] + a_1 x[n-1] + b_1 y[n-1] - b_2 y[n-2], \quad (7)$$

where $x[n]$ is the input at location n, and $y[n]$ is the filtered output at location n. Because the recursive IIR filter relies on the (previously computed) causal filtering result when computing the current output, in order to compute $y[n_0]$, all $y[m]$'s for $m < n_0$ essentially need to be computed. This suggests that as long as a single pixel in the image has an intensity in the range $[L_{i-1}, L_i)$, then the whole image-sized $\text{PBFIC}(L_i)$ and $\text{PBFIC}(L_{i-1})$ (or at least some "causal portion" of them) will need to be computed. This does not lead to significant savings in computation.

Note however that since the Gaussian filter has a rather short (fast decaying) tail, it can be truncated aggressively in practice (see Fig. 1). When the Gaussian filtering is computed using the recursive method, this is equivalent to saying that faraway signals can be ignored completely. More precisely, when computing the filtered output of (7) at location n_o using a truncated version of the signal $x[n]$, i.e., using $x'[n] = 0$ for $n < n_o - c$, and $x'[n] = x[n]$ for $n >= n_o - c$, the difference from the true filtered value using $x[n]$ is very small when c is sufficiently large. In practice, a moderate value of c is usually sufficient and we shall denote such a choice as c_o and call it the coherent width. The precise choice of c_o is usually application dependent and we shall return to this point later in the context of our algorithm.

With the coherent width c_o fixed, the previous mentioned difficulty is alleviated. To compute $y[n_0]$ within a small error, only additional $y[m]$'s for $m = n_0 - c_o, n_0 - c_o + 1, \ldots, n_0 - 1$ need to be computed. For two-dimensional filtering on an image, the concept of the coherent length can be generalized to the concept of the coherent region. For example, for the forward vertical filter pass, the coherent region is illustrated in Fig. 2. Combining it with the coherent region of the backward vertical filter pass, as well as the forward and backward horizontal filter passes, the overall coherent region is a square neighborhood of size $(2c_o + 1) \times (2c_o + 1)$ with the pixel at the center. This implies if the intensity value of a pixel $\vec{x}_o$ is sandwiched between two levels L_i and L_{i+1}, then we only need to compute $J'_{L_i}(\vec{x})$, $W'_{L_i}(\vec{x})$, $J'_{L_{i+1}}(\vec{x})$ and $W'_{L_{i+1}}(\vec{x})$ for $\vec{x}$ in a square neighborhood of size $(2c_o+1) \times (2c_o+1)$ with $\vec{x}_o$ at the center. Note that we could potentially distinguish the different coherent regions for the four filter passes, but this complicates the algorithm and does not appear to provide significant gain in practice.

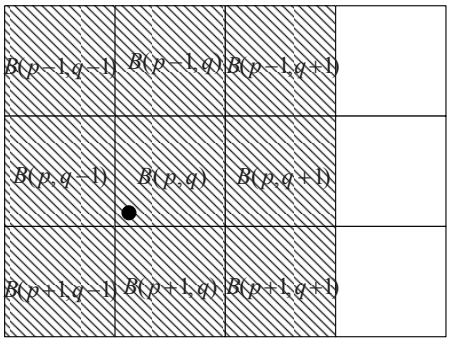

Fig. 3. A pixel (the black dot) has an intensity in the range $[L_{i-1}, L_{i+1})$, then PBFIC(L_i) is computed for all the shaded area covering nine blocks.

Following the above discussion, we could generate computation maps for a given image for the specified levels $L_0, L_1, \ldots, L_{K-1}$ using a morphological dilation operation, and avoid all unnecessary computations following these maps. This naive approach is however counter-productive since directly computing such maps is relatively (to Yang's original algorithm) expensive, defeating the whole purpose of accelerating the algorithm. Next, we shall propose a simple approach to approximate such maps and describe our algorithm.

C. Block Partitioning and Range Finding

The first simplification we propose is to partition the image into blocks of size $c_o \times c_o$, and thus instead of considering whether PBFIC(L_k) needs to be computed for a given pixel in the image, we consider whether it needs to be computed for a given block $B(p,q)$, where q and q are its horizontal and vertical indices. Following the discussion above, it is clear that PBFIC(L_i) should be computed for block $B(p,q)$ if

- There exists a pixel, whose intensity value is in the range $[L_{i-1}, L_{i+1})$, in $B(p,q)$, or
- There exists a pixel, whose intensity value is in the range $[L_{i-1}, L_{i+1})$, in the eight neighboring blocks of $B(p,q)$.

The first case is necessary because of the filter computation for that particular pixel, and the second case is because the pixel in $B(p,q)$ may be in the coherent region of a certain pixel of interest in the neighboring blocks; see Fig. 3.

In a given image block $B(p,q)$, denote the maximum pixel intensity value and the minimum pixel intensity value as $\max(p,q)$ and $\min(p,q)$, respectively. To determine whether there exists a pixel in $B(p,q)$ is in the range $[L_{i-1}, L_{i+1})$ such that PBFIC(L_i) needs to be computed for this block, we make the simplification to assume there is such a pixel unless

$$L_{i-1} \geq \max(p,q) \qquad \text{or} \qquad L_{i+1} < \min(p,q), \tag{8}$$

i.e., to assume the pixels in block $B(p,q)$ take up all the intensity values between $\max(p,q)$ and $\min(p,q)$. Although this is in general not true, it does not appear to make significance difference in practice, but leads to a simpler algorithm.

D. The Proposed Algorithm

We are now ready to describe the proposed algorithm. Different from Yang's original algorithm, we have a pre-filtering stage to find the maximum and minimum values of a block, moreover to find for each block the maximum and minimum values in the region consisting of its own and its neighboring blocks, i.e.,

Pre-filtering Step:

1) For each block $B(p,q)$, find $\max(p,q)$ and $\min(p,q)$;
2) For each block $B(p,q)$, find $\max'(p,q)$ and $\min'(p,q)$ where

$$\max'(p,q) \triangleq \max_{|i-p|\leq 1, |j-q|\leq 1} \max(i,j)$$
$$\min'(p,q) \triangleq \min_{|i-p|\leq 1, |j-q|\leq 1} \min(i,j).$$

For simplicity, we have implicitly assumed above that (i,j) is only enumerated for blocks that exists in the image.

The other part of the algorithm largely follows Yang's original algorithm, except when performing the recursive filtering. To be more precise, consider the horizontal forward filtering pass to compute $J'_{L_i}(\vec{x})$ for a given row, which lies within the row of blocks $B(p,1), B(p,2), \ldots, B(p,m)$; recall the block is of size $c_o \times c_o$, then

Horizontal Pass for a Row of pixels: For $q = 1, 2, \ldots, m$

1) If $L_{i-1} < \max'(p,q)$ and $L_{i+1} \geq \min'(p,q)$
   a) If $B(p, q-1)$ was not filtered, initialize the IIR filter;
   b) Filter the row segment within the block $B(p,q)$ using the IIR filter;

The precise initialization settings can be found in [10], [11], but of importance to us is the fact that it only depends on the pixels intensity at the boundary of the current block. Note that the block $B(p,-1)$ clearly does not exist, and thus in the above algorithm it would be considered as not filtered. The above row filtering procedure is done on each row of the image J_{L_i} and W_{L_i}. The other three passes are done similarly, which eventually give J'_{L_i} and W'_{L_i}, and subsequently PBFIC(L_i).

Another benefit of using the block structure is in updating the resultant filtered image $I^B(\vec{x})$. In the original algorithm, when PBFIC(L_i) and PBFIC(L_{i+1}) have been computed, the image is then scanned for any pixel whose intensity falls between these two levels, and the filtered values are updated for these locations using linear interpolation. However, since we have recorded $\max(p,q)$ and $\min(p,q)$, if $\max(p,q) < L_i$ or $\min(p,q) \geq L_{i+1}$, there is no need to search for such pixels within $B(p,q)$. This provides some noticeable but less significant speedup, because the most expensive computation is in computing PBFICs, but not in this updating steps.

IV. Performance Evaluation

We evaluate the performance of the proposed algorithm using a single-thread C/C++ implementation. The computer used has 2G memory and a 3.33GHz Intel i7-980X processor on Windows 7 platform. Our implementation utilizes the popular OpenCV library, and to eliminate any unfair advantage, we implemented Yang's original algorithm using this library, which is slightly faster than the implementation in [8].

The parameters σ_S and σ_R are application dependent, however usually in the range $4 \sim 40$ and $0.04 \sim 0.3$, respectively, for typical applications. The higher the number of levels K, the

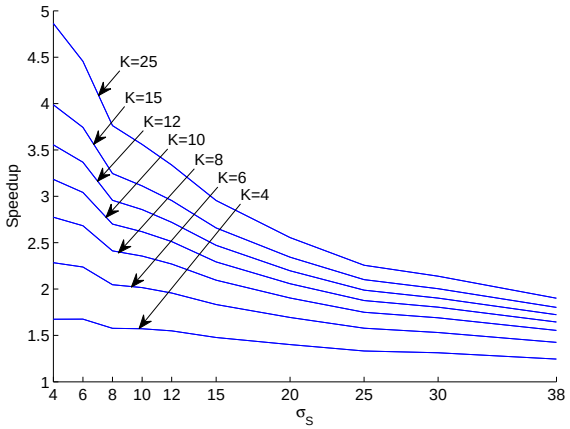

Fig. 4. Average speedup vs. σ_S for various K values.

better the filtering accuracy at the cost of more computation; it is usually sufficient to use K in $4 \sim 25$ when the dynamic range is $0 \sim 255$ depending on the accuracy requirement of the applications (usually 40dB is considered sufficient).

Empirically we found that choosing $c_o = \sigma_S$ offers a good tradeoff between accuracy and efficiency for the typical range of σ_S and σ_R given above, and it is used for the proposed algorithm; for large σ_S values, the advantage of the proposed technique diminishes (see. Fig. 4), and by setting c_o larger, our algorithm reduces to Yang's original algorithm. Thirty randomly chosen images (from Internet) are used, the content of which includes people, natural sceneries, buildings, plants and animals; the images have sizes in the range 0.5$\sim$6 million pixels. The naive implementation of the bilateral filtering is used to compute the ground-truth.

We first compare the accuracy of the proposed algorithm and reference algorithm (each to the ground-truth). The measure used is the relative difference in peak signal-to-noise ratio (PSNR) defined as

$$\text{rd-PSNR} = \frac{\text{PSNR}_{ref}(K, \sigma_S, \sigma_R) - \text{PSNR}_{pro}(K, \sigma_S, \sigma_R)}{\text{PSNR}_{ref}(K, \sigma_S, \sigma_R)},$$

where $\text{PSNR}_{ref}(K, \sigma_S, \sigma_R)$ and $\text{PSNR}_{pro}(K, \sigma_S, \sigma_R)$ are the PSNRs of the filtered results by the reference algorithm and that by the proposed algorithm, respectively, with parameters (K, σ_S, σ_R) for a given image. The parameters σ_S is sampled as in Fig. 4, σ_R is sampled at $(0.04, 0.06, 0.08, 0.1, 0.12, 0.15, 0.18, 0.25, 0.3)$, and K is sampled at $(4, 6, 8, 10, 12, 15, 25)$. The average (over the test images and the sampled parameters) rd-PSNR is 0.0115 and the variance is 5.97×10^{-4}. The inaccuracy increases as σ_R and σ_S increase, with its dependence on σ_S being more prominent; the rd-PSNR is almost 3% at large σ_R and σ_S when averaged over the test images, while being less than 1% at the other extreme. We deem this kind of accuracy degradation acceptable because at the targeted quality (PSNR 40-55dB), less than 1dB difference is usually negligible for typical applications of the bilateral filter. One can improve the accuracy by choosing a more conservative c_o at the cost of a reduced speedup.

Next we consider the speedup of the proposed method. From the structure of the method, it is clear that the speedup does not depend on the parameter σ_R critically, and thus we plot the average speedup (over the test images and σ_R) vs σ_S for fixed K values in Fig. 4. It can be seen that the smaller the value σ_S, the more effective the proposed method is. This is because $c_o = \sigma_S$ and smaller c_o allows the method to eliminate the unnecessary computations more effectively. The method is also more effective for larger values of K because this usually implies a larger proportion of the PBFIC computations in the original algorithm are unnecessary, which provides the proposed method a larger target to reduce. As an example, on a typical 1.5 mega-pixel image, the proposed algorithm is able to complete in 383ms with parameters $\sigma_R = 0.06$ and $\sigma_S = 15$, while Yang's original algorithm completes in 835ms with 45dB accuracy.

V. Conclusion

We propose a method to accelerate Yang's bilateral filtering algorithm by eliminating unnecessary computations through block-skipping IIR filtering. It is able to achieve speedup of 1.5$\sim$5 times. Although only Gaussian kernel is treated here, the method is general for other IIR spatial filtering kernels with a short tail (e.g., the box filter). It was mentioned in [8] that by downsampling the image, the algorithm can be further sped up. Our method can also incorporate such downsampling, and the amount of speed up is similar to using the proposed method on a smaller image with a smaller value of σ_S.

The proposed method can also be used in a GPU-based implementation, however the speedup is much less significant, usually in the range of 1.1$\sim$1.3 in our CUDA-based implementation. The main reason is that to take advantage of the parallelism in the GPU, we choose c_o to be a multiple of 16 or 32, however, this conservative choice reduces the speedup; together with other architectural bottlenecks in the GPU, the proposed method becomes less effective. As a future work, we plan to investigate whether it is possible to make the choice of c_o to play a less significant role in the overall computation architecture of the GPU-based implementation.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. ICCV '98*, pp. 839–846, 1998.

[2] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," in *Proc. CVPR '07*, 2007.

[3] F. Durand and D. Dorsey, "Fast bilateral fitlering for the display of high-dynamic-range images," in *Proc. Siggraph '02*, pp. 257–266, 2002.

[4] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama, "Digital photography with flash and no-flash image pairs," in *Proc. Siggraph '04*, pp. 664–672, 2004.

[5] K. J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. PAMI*, vol. 28, pp. 650–656, 2006.

[6] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," in *Proc. ECCV '06*, pp. 568–580, 2006.

[7] F. Porikli, "Constant time $O(1)$ bilateral filtering," in *Proc. CVPR '08*, pp. 1–8, 2008.

[8] Q. Yang, K.-H. Tan, and N. Ahuja, "Real-time $O(1)$ bilateral filtering," in *Proc. CVPR '09*, pp. 557–564, 2009. Source code available at http://www.cs.cityu.edu.hk/~qiyang/

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

[10] R. Deriche, "Fast algorithms for low-level vision," *IEEE Trans. PAMI*, vol. 12, no. 1, pp. 78–87, 1990.

[11] R. Deriche, "Recursively implementing the Gaussian and its derivatives," *Tech. Rep. 1893 INRIA, Unit de Recherche Sophia-Antipolis*, 1993.