Fast bilateral filtering using recursive moving sum

Masaki Igarashi¹ᵃ), Masayuki Ikebe¹, Sohsuke Shimoyama¹, and Junichi Motohisa¹

¹ Graduate School of Information Science and Technology, Hokkaido University
  Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan

ᵃ) igarashi@impulse.ist.hokudai.ac.jp

Received October 28, 2011; Published April 1, 2012

Abstract: We propose a constant-time algorithm for a bilateral filter. Bilateral filter can be converted into the operation of three-dimensional (3D) convolution. By using recursive moving sum, we can reduce the number of calculations needed to construct a pseudo-Gaussian filter. Applying one-dimensional Gaussian filter to the 3D convolution, we achieved a constant-time bilateral filter. We used a 3-GHz CPU without SIMD instructions, or multi-thread operations. We confirmed our proposed bilateral filter to be processed in constant time. In practical conditions, high PSNR values over 40 dB are obtained.

Key Words: image processing, bilateral filter, constant-time algorithm

1. Introduction

In image processing, image filters are widely used for various purposes such as reducing noise, enhancing images, and detecting edges. Tomasi and Manduchi proposed a bilateral filter that was an edge preserving smoothing filter [1]. It has been used for several applications such as noise reduction, dynamic range compression in Retinex [3]. The output images are obtained by using weight averaging in the spatial and range (intensity) domains.

Until recently, the bilateral filter was computationally expensive because brute-force implementation has a $O(n^2)$ computational cost per output pixel ($n$: filter size). Several researchers have studied how to speed up the bilateral filter. Porikli proposed the $O(1)$ bilateral filter using an integral histogram [5]. Yang et al. also proposed a $O(1)$ algorithm on the basis of Durand and Dorsey’s method [2, 6]. Paris and Durand studied the bilateral filter as 3D convolution. This method uses fast Fourier transform (FFT) for the convolution operation, so it requires a large number of calculations.

The purpose of this study is to propose the yet another constant-time algorithm for the bilateral filter by extending Paris’s method.

2. Bilateral filtering

In this section, we explain the basis of the bilateral filter. Main notations in this paper are summarized in Table I.

The bilateral filter uses a spatial and a range kernel to update each pixel in an image. Let $p \in S$
denote a pixel position in the image and $I_p$ be the intensity of pixel $p$. Bilateral filter updates intensity value as follows:

$$\forall p \in S$$

$$I_p^{bf} = \frac{1}{K_p} \sum_{q \in S} w_s(p, q) w_r(p, q) I_q$$

$$K_p = \sum_{q \in S} w_s(p, q) w_r(p, q),$$

where $w_s$ and $w_r$ are the weighting functions in the spatial and range domains, respectively. $I_p^{bf}$ is the result of bilateral filter. $K_p$ is used for normalization. Usually, a constant and a Gaussian weighting function are used for $w_s$ and the Gaussian weighting function is used as $w_r$ (Eqs. (3)–(5)).

$$w_s(p, q) = G_{\sigma_s}(|p - q|) \quad \text{or} \quad 1$$

$$w_r(p, q) = G_{\sigma_r}(|I_p - I_q|)$$

$$G_\sigma(x) = \exp \left(-\frac{x^2}{2\sigma^2}\right).$$

A two-dimensional (2D) bilateral filter can be rewritten as 3D convolution [4] as follows:

$$\forall x, y, z \in S \times R$$

$$I_{x,y,z}^* = \begin{cases} I_{x,y} (z = I_{x,y}) \\ 0 (z \neq I_{x,y}) \end{cases}$$

$$K_{x,y,z}^* = \begin{cases} 1 (z = I_{x,y}) \\ 0 (z \neq I_{x,y}) \end{cases}$$

$$\begin{pmatrix} I_{x,y,z}^* \\ K_{x,y,z}^* \end{pmatrix} = \sum_{x' \in W} \left[ G_{\sigma_s}(x' - x) \begin{pmatrix} I_{x',y,z}^* \\ K_{x',y,z}^* \end{pmatrix} \right]$$

$$\begin{pmatrix} I_{x,y,z}^* \\ K_{x,y,z}^* \end{pmatrix} = \sum_{y' \in H} \left[ G_{\sigma_r}(y' - y) \begin{pmatrix} I_{x,y',z}^* \\ K_{x,y',z}^* \end{pmatrix} \right]$$

$$\begin{pmatrix} I_{x,y,z}^* \\ K_{x,y,z}^* \end{pmatrix} = \sum_{z' \in R} \left[ G_{\sigma_r}(z' - z) \begin{pmatrix} I_{x,y,z'}^* \\ K_{x,y,z'}^* \end{pmatrix} \right]$$

$$I_p^{bf} = \frac{I_{x,y,I_p}^{bf}}{K_{x,y,I_p}^{bf}}.$$
3. Proposed bilateral filtering method

In this section, we propose a method of bilateral filtering. We define the recursive moving sum. Constant-time convolution is constructed by using recursive moving sum. Thus we can achieve a constant-time bilateral filter on the basis of constant-time convolution.

3.1 Recursive moving sum

Here we define the recursive moving sum. Then we describe how much weight recursive moving sum gives to input data.

Let \( x_{n} \in \{0, 1, \ldots, N-1\} \) denote the input data sequence, where \( N \) is the length of the sequence. In image processing, this sequence means brightness values of pixels in an image. We define the recursive moving sum \( s_{n}^{m} \) as follows:

\[
\begin{align*}
  s_{n}^{0} &= x_{n} \\
  s_{n}^{m} &= \sum_{i=-r}^{r} s_{n+i}^{m-1} \quad (m \geq 1, r \in \mathbb{N}),
\end{align*}
\]

where we call \( r \) base radius. Recursive moving sum calculates the moving sum recursively. \( m \) means repeat count of the moving sum. Here we are not concerned with handling the elements located out of sequence (\( x_{i<0}, x_{i\geq N} \)).

Let us take \( s_{n}^{1} \) for example.

\[
\begin{align*}
  s_{n}^{1} &= \sum_{i=-r}^{r} s_{n+i}^{0} \\
  &= x_{n-r} + x_{n-r+1} + \cdots + x_{n+r-1} + x_{n+r}.
\end{align*}
\]

\( s_{n}^{1} \) is the summation from \( x_{n-r} \) to \( x_{n+r} \).

\( s_{n}^{2} \) is calculated as follows:

\[
\begin{align*}
  s_{n}^{2} &= \sum_{i=-r}^{r} s_{n+i}^{1} \\
  &= s_{n-r}^{1} + s_{n-r+1}^{1} + \cdots + s_{n+r-1}^{1} + s_{n+r}^{1} \\
  &= x_{n-2r} + x_{n-2r+1} + \cdots + x_{n-1} + x_{n} + x_{n-1} + x_{n-2} + \cdots + x_{n+r-1} + x_{n+r-2} + \cdots + x_{n+r} \\
  &= x_{n-4} + 2x_{n-3} + 3x_{n-2} + 4x_{n-1} + 5x_{n} + 4x_{n+1} + 3x_{n+2} + 2x_{n+3} + x_{n+4}.
\end{align*}
\]

This handling is dependent on the implementations (e.g. \( x_{i} = x_{i-1} \) or \( x_{i} = 0 \) etc. if \( i < 0 \)).
\[ s_n^3 = \sum_{i=1}^{1} s_{n+i}^2 \]
\[ = \sum_{i=-1}^{1} \{ s_{n+i-1}^1 + s_{n+i}^1 + s_{n+i+1}^1 \} \]
\[ = \sum_{i=-1}^{1} \{ x_{n+i-2} + 2x_{n+i-1} + 3x_{n+i} + 2x_{n+i+1} + x_{n+i+2} \} \]
\[ = x_{n-3} + 3x_{n-2} + 6x_{n-1} + 7x_n + 6x_{n+1} + 3x_{n+2} + x_{n+3}. \] (16)

The center weights, such as the coefficient of \( x_n \), \( x_{n-1} \) and \( x_{n+1} \), are large. Meanwhile, the weights to the data apart from center tend to be small. We show the weight map of recursive moving sum in Fig. 1. By increasing \( m \), the weight of \( s_m^n \) approaches Gaussian distribution in accordance with central limit theorem.

We can adjust the standard deviation of Gaussian by controlling the radius of recursive moving sum. Equations (17) and (18) show the relation between radius \( r \), repeat count \( m(= 2, 3) \), and standard deviation \( \sigma \).

\[ \frac{1}{\sqrt{2\pi\sigma^2}} = \frac{1}{2r + 1} \] (m = 2) (17)
\[ \frac{1}{\sqrt{2\pi\sigma^2}} = \frac{3r^2 + 3r + 1}{(2r + 1)^3} \] (m = 3) (18)

3.2 Constant-Time convolution by using recursive moving sum

Next, we describe the recursive moving sum is calculated in constant time.

We can convert Eq. (12) into a recurrence formula:

\[ s_{n}^{0} = x_{n} \]
\[ s_{n}^{m} = \sum_{i=-r}^{r} s_{i}^{m-1} \] (m \(\geq\) 1)
\[ s_{n}^{m} = s_{n-1}^{m} + s_{n+r}^{m-1} - s_{n-r-1}^{m-1} \] (m \(\geq\) 1, \(n \geq\) 1). (19)

When successive values of recursive moving sum are calculated, current sum \( s_n^m \) is calculated by adding/subtracting two elements \( s_{n+r}^{m-1}, s_{n-r-1}^{m-1} \) to/from the old sum \( s_{n-1}^{m} \), meaning a full summation each time is unnecessary.
Let us show concrete examples with focusing on the number of operations such as additions and subtractions. First, consider the calculation of $s_n^1$. In the case of original definition (Eq. (12)), $s_n^1$ is calculated as follows:

\[
\begin{aligned}
\ s_0^1 &= x_{-r} + x_{-r+1} + \cdots + x_{r-1} + x_r \quad \Rightarrow \quad 2r \text{ additions} \\
\ s_1^1 &= x_{-r+1} + x_{-r+2} + \cdots + x_r + x_{r+1} \quad \Rightarrow \quad 2r \text{ additions} \\
\ &\vdots \\
\ s_n^1 &= x_{n-r} + x_{n-r+1} + \cdots + x_{n+r-1} + x_{n+r} \quad \Rightarrow \quad 2r \text{ additions}
\end{aligned}
\]

It takes $2r$ times additions to calculate each term. When we use the recurrence formula (Eq. (19)), $s_n^1$ is expanded as follows:

\[
\begin{aligned}
\ s_0^1 &= x_{-r} + x_{-r+1} + \cdots + x_{r-1} + x_r \quad \Rightarrow \quad 2r \text{ operations} \\
\ s_1^1 &= s_0^1 + s_{r+1}^0 - s_{-r}^0 \quad \Rightarrow \quad 2 \text{ operations} \\
\ &\vdots \\
\ s_n^1 &= s_{n-1}^1 + s_{n+r}^0 - s_{n-r-1}^0 \quad \Rightarrow \quad 2 \text{ operations}
\end{aligned}
\]

The first term $s_0^1$ requires $2r$ additions. To calculate subsequent terms, meanwhile, it takes only two operations per term. If $n$ is sufficiently-large, $2r$ operations for the first term is negligibly small. Generally, $s_n^m$ is obtained by $2^{m+1} - 2$ times calculation per term. Considering the appearance of same term during calculation, it takes $m(m + 1)$ operations to obtain $s_n^m$. By using the definition on the basis of recurrence relation, the total number of calculations for recursive moving sum depend less on $r$.

As stated above, $s_n^m$ is calculated in constant time and gives Gaussian-like weight to the input data. Consequently, we can achieve pseudo Gaussian filter processed in constant time by using $s_n^m$. Figure 2 describes the scheme of calculation of $s_n^2$. We show a pseudo code of the algorithm for pseudo Gaussian filter corresponding to $s_n^3$ in Procedure 1.

3.3 Constant-Time bilateral filtering using fast convolution
According to Eqs. (6)–(11), the bilateral filter is processed by 3D convolution. Paris and Durand used FFT for 3D convolution. On the other hand, we apply the method of the constant-time convolution on the basis of recursive moving sum to the bilateral filter. We present a pseudo-code of the algorithm for the bilateral filter in Procedure 2.

3.4 Down sampling
For more speeding up, we used down sampling technique. By shrinking the image before 3D convolution, the convolution operation is processed faster. Procedure 3 shows the pseudo code of the bilateral filter with down sampling.
Procedure 1 convolve\((x, r)\): Pseudo Gaussian using constant-time convolution (e.g. \(s_n^3\)).

**Input:** Data array \(x\), radius \(r\)

**Output:** Filtered data array \(y\)

1: \(N \leftarrow \text{length of array } x\)
2: **for** \(n = 0 \text{ to } N - 1\) **do**
3:   **if** \(n = 0\) **then**
4:     # Initialize variables based on the definition (Eq. (12)).
5:     \(g_{1L} \leftarrow s_{2r-2}, \quad g_{1R} \leftarrow s_{r-1},\)
6:     \(g_{1L} \leftarrow s_{r}, \quad g_{1R} \leftarrow s_{2r},\)
7:     \(g_{2L} \leftarrow s_{2r-1}, \quad g_{2R} \leftarrow s_{r}, \quad g_{3} \leftarrow s_{0}\)
8:   **else**
9:     \(g_{1L} \leftarrow g_{1L} + x[n - r - 2] - x[n - 3r - 3]\)
10:    \(g_{1R} \leftarrow g_{1R} + x[n + r - 1] - x[n - r - 2]\)
11:    \(g_{1L} \leftarrow g_{1L} + x[n + r - 1] - x[n - r - 2]\)
12:    \(g_{1R} \leftarrow g_{1R} + x[n + 3r] - x[n + r - 1]\)
13:    \(g_{2L} \leftarrow g_{2L} + g_{1R} - g_{1L}\)
14:    \(g_{2R} \leftarrow g_{2R} + g_{1R} - g_{1L}\)
15:    \(g_{3} \leftarrow g_{3} + g_{2R} - g_{2L}\)
16: **end if**
17: **end for**
18: \(y[n] \leftarrow \frac{g_{3}}{(2r + 1)^{3}} \quad \# \text{Normalization.}\)

4. Experimental results and discussion

We tested the proposed bilateral filter method and evaluated the processing time and image quality. These tests were run on a PC with Intel Core i7 CPU (3.07 GHz) and 12 GB main memory. Our implementation was written in C++. We used \(s_n^2\) as pseudo Gaussian of proposed bilateral filter. We did not use multithread operations, or SIMD instructions. The implementation of Paris and Durand’s technique we used is available at the below website\(^2\).

The computation times are given in Fig. 3(a). Our proposed method is much faster than that of Paris and Durand’s one. Calculation times of both methods lengthened as images size increased. Given that \(n\) is an image size, time complexity of Paris and Durand’s method is more than \(O(n \log n)\) due to using FFT. Meanwhile, time complexity of our method is \(O(n)\).

Next, we examined the relationship between the processing time and standard deviation of the Gaussian. Figure 3(b) shows the measurement results. We found that the processing time of the proposed method depend less on the standard deviation. In Paris and Durand’s method, however, processing time varies in accordance with the standard deviation.

Table II shows the dependence of the processing time on spatial and range sampling factor. Processing time becomes shorter in accordance with the increase of spatial and range sampling factor. We confirmed that the operation of down and up sampling occupy the large amount of calculation time when the sampling factor is large.

We analyzed the filter accuracy by performing filtering operation and by calculating the Peek Signal-to-Noise Ratio (PSNR) defined in Eq. (20). We calculated PSNR by using the output images of the proposed bilateral filter and the exact bilateral filter which uses Gaussian spatial and range weight. We used twenty test images. Figures 4 and 5 show PSNR of bilateral filters with a pseudo-Gaussian kernel.

Dependency of spatial radius is shown in Fig. 4(a). PSNR values are located near the 55dB to 60dB, and depend less on spatial radius. Figure 4(b) shows the dependence on range radius. In accordance with the increase in range radius, PSNR value decrease.

Then, we examined the relationship between sampling factor and PSNR accuracy. Figure 5 show

\(^2\)http://people.csail.mit.edu/sparis/bf/
Procedure 2 bilateralFilter($I, r_s, r_t$): Fast bilateral filter using constant-time convolution.

**Input:** Image $I$, radius $r_s, r_t$

**Output:** Filtered image $I^{bf}$

1: # Initialization.
2: for all $x, y, z \in W \times H \times R$ do
3:   $I^*[x][y][z] \leftarrow 0$
4:   $K^*[x][y][z] \leftarrow 0$
5: end for

6: # Mapping to 3D space.
7: for all $x, y \in W \times H$ do
8:   $I^*[x][y][I[x][y]] \leftarrow I[x][y]$
9:   $K^*[x][y][I[x][y]] \leftarrow 1$
10: end for

11: # 3D Convolution along each axis.
12: for all $y, z \in H \times R$ do
13:   $I'[:][y][z] \leftarrow \text{convolve}(I^*[:][y][z], r_s)$
14:   $K'[:][y][z] \leftarrow \text{convolve}(K^*[:][y][z], r_s)$
15: end for

16: for all $z, x \in R \times W$ do
17:   $I''[:][z] \leftarrow \text{convolve}(I'[:][z], r_s)$
18:   $K''[:][z] \leftarrow \text{convolve}(K'[:][z], r_s)$
19: end for

20: for all $x, y \in W \times H$ do
21:   $I^{bf}[x][y] \leftarrow \frac{I''[:][y][I[x][y]]}{K''[:][y][I[x][y]]}$
22: end for

the dependence of spatial sampling factor and range one. These results show that the large sampling factor degrades the filter accuracy.

Figure 6 shows the result of applying the proposed method to an image with various radius value. We found that the proposed method achieved smoothing, while preserving the edge in some conditions. We compared the output image of proposed bilateral filter with the output of exact bilateral one in Fig. 7. We cannot distinguish between the resulting images obtained by the proposed method and exact images.

\[
\text{MSE} \equiv \frac{1}{WH} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} (I_{i,j} - J_{i,j})^2
\]

\[
\text{PSNR} \equiv 20 \log_{10} \left( \frac{255}{\sqrt{\text{MSE}}} \right)
\]

(20)
(a) Dependency of image size \((\sigma_s, \sigma_r = 13.17)\).

(b) Dependency of standard deviation \((640 \times 480)\).

**Fig. 3.** Comparison of processing times of two methods (natural image).

<table>
<thead>
<tr>
<th>spatial sampling factor (s_s)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>range sampling factor (s_r)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.03</td>
<td>0.773</td>
<td>0.208</td>
<td>0.0759</td>
<td>0.0420</td>
</tr>
<tr>
<td>2</td>
<td>1.53</td>
<td>0.409</td>
<td>0.124</td>
<td>0.0525</td>
<td>0.0357</td>
</tr>
<tr>
<td>4</td>
<td>0.801</td>
<td>0.220</td>
<td>0.0764</td>
<td>0.0412</td>
<td>0.0328</td>
</tr>
<tr>
<td>8</td>
<td>0.420</td>
<td>0.125</td>
<td>0.0533</td>
<td>0.0353</td>
<td>0.0313</td>
</tr>
<tr>
<td>16</td>
<td>0.236</td>
<td>0.0779</td>
<td>0.0418</td>
<td>0.0327</td>
<td>0.0307</td>
</tr>
</tbody>
</table>

**Table II.** Processing time (s) of proposed method (dependency on sampling factor, natural image: \(640 \times 480\), spatial radius \(r_s = 16\), range radius \(r_r = 16\)).

(a) Dependency on spatial radius \(r_s\).

(b) Dependency on range radius \(r_r\).

**Fig. 4.** PSNR accuracy [dB] of proposed method.
Fig. 5. PSNR accuracy [dB] of proposed method \((r_s, r_r = 4)\).

Fig. 6. Results of proposed bilateral filter with various radius parameter value.
Fig. 7. Comparison of the output images of proposed bilateral filter and exact bilateral filter.
Procedure 3 bilateralFilterDS(I, r_s, r_r, s_s, s_r): Fast bilateral filter with down sampling.

Input: Image I, radius r_s, r_r, sampling factor s_s, s_r
Output: Filtered image I_{bf}

1: W' ← W/s_s
2: H' ← H/s_s
3: R' ← R/s_r
4: I* ← mapTo3D(I)
5: K* ← mapTo3D(K)
6: # Down sampling.
7: I_d ← resize(I*, W', H', R')
8: K_d ← resize(K*, W', H', R')
9: # 3D Convolution.
10: I_c ← convolve3D(I_d)
11: K_c ← convolve3D(K_d)
12: # Up sampling.
13: I_c ← resize(I_c, W, H, R)
14: K_c ← resize(K_c, W, H, R)
15: I_{bf} ← mapTo2D(I_c/K_c)

5. Conclusion
We proposed a constant-time algorithm for a bilateral filter. By formulating recursive moving sum which processed in constant time, we reduced the number of calculations needed to construct pseudo-Gaussian filter. Applying a one-dimensional Gaussian filter to the 3D convolution, we achieved a constant-time bilateral filter.

In our experiments, we confirmed that our proposed bilateral filter to be processed in constant time and high PSNR values are obtained in practical conditions.

Acknowledgments
This work was supported by Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists.

References