Multigrid Bilateral Filtering

Qingyun SHE\(^{(a)}\), Zongqing LU\(^{(b)}\), Nonmembers, Weifeng LI\(^{(c)}\), Member, and Qingmin LIAO\(^{(d)}\), Nonmember

SUMMARY The bilateral filter (BF) is a nonlinear and low-pass filter which can smooth an image while preserving detail structures. However, the filter is time consuming for real-time processing. In this paper, we bring forward a fresh idea that bilateral filtering can be accelerated by a multigrid (MG) scheme. Our method is based on the following two facts. a) The filtering result by a BF with a large kernel size on the original resolution can be approximated by applying a small kernel sized (3×3) version on the lower resolution many times on the premise of visual acceptance. Early work has shown that a BF can be viewed as nonlinear diffusion. The desired filtering result is actually an intermediate status of the diffusion process. b) Iterative linear equation techniques are sufficiently mature to cope with the nonlinear diffusion equation, which can be accelerated by the MG scheme. Experimental results with both simulated data sets and real sets are provided, and the new method is demonstrated to achieve almost twice the speed of the state-of-the-art. Compared with previous efforts for finding a generalized representation to link bilateral filtering and nonlinear diffusion by adaptive filtering, a novel relationship between nonlinear diffusion and bilateral filtering is explored in this study by focusing attention on numerical calculus.

key words: bilateral filter, multigrid, nonlinear diffusion

1. Introduction

Filtering is a fundamental operation of computer vision. More advanced filtering methods aim at preserving the signal details while removing the noise. This is achieved by a locally adaptive recovery paradigm, such as anisotropic diffusion (AD) [1]–[6], weighted least squares (WLS) [7], and robust estimation (RE) [8]. All these methods share the fact that local relations between the samples dictate the final result, and most of these methods resort to an iterative algorithm.

Recently, Tomasi and Manduchi [9] proposed an alternative non-iterative filter—bilateral filter. This filter is merely a weighted average of local neighborhood samples, where the weights are computed based on spatial and radiometric distances between the center sample and its neighbors. This filter is also locally adaptive, and it was shown to provide similar and possibly better results, compared to those obtained by the previously mentioned iterative approaches. The bilateral filter (BF) has been adopted for many applications, such as image denoising [9], [10], relighting and texture manipulation [11], dynamic range compression [12], illumination correction [13], and photograph enhancement [14]–[16]. The method has also been adapted to other domains, such as mesh fairing [17], [18], volumetric denoising [19] and video processing [20], [21]. The implementation of the BF is very simple (we will give a brief introduction of the BF in part II), and it is easy to adapt to a given context as long as a distance can be computed between two pixel values. The BF is non-iterative, thereby achieving satisfying results with only a single step.

Although the bilateral filter has been proven to be very useful, it is time consuming and nonlinear, and traditional accelerations, such as performing convolution after a fast Fourier transform (FFT), are not applicable. Nonetheless, solutions have been proposed to speed up the execution of the BF [12], [22]–[28]. There are two categories of acceleration schemes [22]: specialized filters that perform an exact computation but have limited performance, and approximated filters that demonstrate more general performance but cannot produce exact results.

As an exact computation, Elad used Gauss-Seidel iterations to accelerate the convergence of iterative filtering [26]. [24] described an efficient algorithm to incrementally compute the intensity histogram of the square windows surrounding each pixel. Porikli [25] demonstrated that BFs can be computed at a constant time with respect to the filter size for three types of BFs. For box spatial and arbitrary range kernels, an integral histogram is used to avoid redundant operations, and interactive speed is achieved by quantizing the input image using a small number of bins.

At the cost of an approximate result, several authors propose fast methods for addressing more general scenarios. In [24], Weiss iterated his filter based on square windows to obtain a smoother profile, thereby removing ripple defects and avoiding shocks that sharpen edges and result in a cartoon look.

Durand and Dorsey [12] linearized the BF which makes possible the use of the FFT, and downsampled the data to accelerate the computation to a second or less for one megapixel, which is a key factor for speedup. Once the data is downsampled, a direct convolution is more efficient than FFT because the kernel is sufficiently small. Paris and Durand [22] expressed the BF with linear operations and obtained much of the resulting speedup from downsampling. However, their formulation relies on a more principled ex-
pression based on a new, higher-dimensional interpretation of images. This affords a solid signal processing perspective on the BF. Their method [22] actually becomes faster as the size increases due to greater subsampling, but the exact output is dependent on the phase of subsampling. [28] extended the work of Durand and Frédo [12], and they discretized the image intensities into a number of values and computed a linear filter for each such value, the output of which is defined as the Principle Bilateral Filter Image Component (PBFIC). The final output is then a linear interpolation between the two closest PBFICs. Zhang Ming and Gunturk [23] extended the BF to a multiresolution BF, where bilateral filtering is applied to the approximation (low-frequency) subbands of a signal decomposed, which were accomplished using a wavelet filter bank. The multiresolution BF is combined with wavelet thresholding to form a new image denoising framework. Fattal and Raanan [29] extended the BF into image enhancement rather than filtering. They computed a multiscale decomposition based on the BF and then reconstructed an enhanced image that combined the detailed information at each scale across all the input images.

In this study, we focus on the accelerating scheme for the BF. A new algorithm is proposed to accelerate the process of bilateral filtering. This article extends our conference paper [30] and full develops the idea. We provide more detailed description and discussion, including the algorithm pseudo-code. The previous errors and unreasonable assumptions are corrected. We provide quantitative speed evaluation comparisons with existing approximations of the bilateral filter. We conduct new conceptual and quantitative quality evaluation which is not given in previous work. We also describe the difference between MGBF and the multiscale based methods.

The main contributions of this study are as follows.

1. Apply the nonlinear multigrid (MG) scheme to accelerate bilateral filtering.
2. Reconsider the relationship between nonlinear diffusion and bilateral filtering in numerical calculus.

This paper is organized as follows. Section 2 will introduce the BF and its relationship with nonlinear diffusion. In Sect. 3, we mainly present the new algorithm termed the Multigrid Bilateral Filter (MGBF). We reconsider the relationship between the BF and nonlinear diffusion in Sect. 4. Experimental results and conclusions are given in Sect. 5 and Sect. 6, respectively.

2. Related Works

2.1 Bilateral Filter

The BF is a nonlinear, low-pass filter which can smooth an image while preserving the edges (as shown in Fig. 1(A)). It can be traced back to 1995 with the work of Aurich and Weule [31] on nonlinear Gaussian filters. It was later rediscovered by Smith and Brady [32] as part of their SUSAN framework, and Tomasi and Manduchi [9] gave it its current name. Since then, the use of BF has grown rapidly and is now ubiquitous in image-processing applications. The BF is a nonlinear filter which combines domain and range filtering (as shown in Fig. 1(B)). Given an input image \( i(x) \), \( x = (x, y) \in \mathbb{R}^2 \), using a continuous representation notation, the output image \( i'(x) \) is obtained by

\[
i'(x) = k^{-1}(x) \int_{\xi \in \Omega(x)} i(\xi) \cdot c(\xi, x) \cdot s(i(\xi), i(x)) d\xi,
\]

where \( \xi = (\xi_1, \xi_2) \in \Omega(x) \). The convolution mask is the product of the functions \( c \) and \( s \), which represent the closeness (in the domain) and similarity (in the range), respectively. In the BF, the domain weight \( c \) is defined by

\[
c(\xi, x) = exp \left( -\frac{||x - \xi||^2}{2\sigma^2} \right),
\]

where \( \sigma_{\xi} \) is the standard deviation in the domain and \( ||x - \xi|| \) is the Euclidean distance between the pixels \( \xi \) and \( x \). For the range weights between pixels \( \xi \) and \( x \), we use

\[
s(i(\xi), i(x)) = exp \left( -\frac{||i(x) - i(\xi)||^2}{2\sigma^2} \right),
\]

where \( \sigma_s \) is the standard deviation in the range and \( ||i(x) -


\[ i(\xi) \parallel \text{is a measure of the distance between the intensities at pixels } \xi \text{ and } x. \] If the intensities of pixels \( \xi \) and \( x \) vary greatly (as in the case at an edge), the corresponding weight \( s(i(\xi), i(x)) \) will be small, thus reducing the smoothing effect. In (1), \( k(x) \) is the normalization factor

\[ k(x) = \int_{\Omega(x)} c(\xi, x) \cdot s(i(\xi), i(x)) \, d\xi. \tag{4} \]

Although the BF has the advantage of smoothing an image while preserving the edges, it is too computationally intensive for real-time applications. In order to apply the BF in real-time processing, a growing body of research has focused on improving the speed of the filtering process. Up to now, many fast versions have been proposed. Among the fastest versions of the BF, downsampling or multiresolution schemes seems to be the most efficient way to accelerate the process [22]–[24].

### 2.2 Nonlinear diffusion and the BF

Here, it is necessary to mention the works of Danny and Barash [33], and to recall the relationship between nonlinear diffusion and bilateral filtering. We introduce \( i_\tau(x) \) (0 \( \leq \tau < \infty \), where integer \( \tau \) denotes the filtering times, and \( i_0(x) = i(x), i_\tau(x) \) means the \( \tau \)th times bilateral filtering result.

We describe the discrete version of (1) as follows.

\[ i_{\tau+1}(x, y) = \sum_{(x, y) \in \Omega(x, y)} w_{(x, y)}(u, v) \cdot i_\tau(x + u, y + v), \tag{5a} \]

where

\[ w_{(x, y)}(u, v) = \frac{S_{(x, y)}(u, v)}{\sum_{(x, y) \in \Omega(x, y)} S_{(x, y)}(u, v)}, \tag{5b} \]

\[ \sum_{(x, y) \in \Omega(x, y)} w_{(x, y)}(u, v) = 1. \tag{5c} \]

\[ p_{(x, y)}(u, v) = c((x, y), (x + u, y + v)), \tag{5d} \]

\[ q_{(x, y)}(u, v) = s(i_\tau(x, y), i_\tau(x + u, y + v)), \tag{5e} \]

\[ S_{(x, y)}(u, v) = p_{(x, y)}(u, v) \cdot q_{(x, y)}(u, v). \tag{5f} \]

The BF (as shown in (5a)) is a typical adaptive smoothing filter, and can evidently also be viewed as a nonlinear diffusion equation [33]. We rewrite (5a) as:

\[ i_{\tau+1}(x, y) - i_\tau(x, y) = \sum_{(u, v) \in \Omega(x, y)} w_{(x, y)}(u, v) i_\tau(x + u, y + v) - i_\tau(x, y) \]

\[ = \sum_{(u, v) \in \Omega(x, y), (x, y)} w_{(x, y)}(u, v) i_\tau(x + u, y + v). \tag{6} \]

Equation (6) can be expressed in a generalized and continuous form as (7).

\[ \frac{dl}{d\tau} = A(I), \tag{7} \]

where \( I \) is the vector expression of \( i_\tau(x, y) \), and \( A \) is a nonlinear function of \( I \) derived from \( \tau \). As a result, we find that the original bilateral filtering can be viewed as a single nonlinear diffusion step. When \( \tau \) is big enough, the filtering results will converge to a stable value under the convergent condition.

### 3. Multigrid Bilateral Filter

#### 3.1 Bilateral Filtering Approximated By Nonlinear Diffusion

As show in Table 1 (the size of image to be processed is 552×480), when using BF to filter an image, the processing time is decreased as the kernel size of the filter reduced. What’s more, the processing time of a BF with a 3×3 kernel size is at least 20 times faster than that with a 18×18 kernel size. Thus, it is an easy task and low computational complexity is required when smoothing an image by applying a
smallest kernel 3x3, even when applied several times. We can modify (5a) with the smallest size kernel version as,

\[ i_{r+1}(x, y) = \sum_{(x, y) \in G} w_{(x, y)}(u, v) \cdot i_r(x + u, y + v). \]  

(8)

Yang mentioned in [34] that a BF with a large kernel size on the original resolution corresponding to a small kernel version on the lower resolution. And as show in Fig. 2, we can infer that, when the parameters of the BF are chosen appropriately, a bilateral filtering process using a large kernel size on the original resolution can be approximated using (8) by applying a smaller kernel on the lower resolution many times on the premise of visual acceptance. The desired filtering result is actually an intermediate status of the diffusion process. The diffusion equation of (8) has the same form as (7). The nature of (7) is to solve a nonlinear equation,

\[ A(I) = 0, \]  

(9)

which can be implemented using iterative techniques. Solving (9) required an iteration process, and each iterative step is equivalent to a one step filtering process as in (5a). The stable solution of (9) is the status that the value of \(|i_{r+1}(x, y) - i_r(x, y)|\) is smaller or equal to a specified and efficiently small value \(\varepsilon\), i.e. \(\forall \varepsilon > 0\)

\[ |i_{r+1}(x, y) - i_r(x, y)| \leq \varepsilon. \]  

(10)

In fact, the desired filtering version is a visually satisfactory solution \(I_a\) \((1 \leq a < \infty)\) of (9) under the iterative condition that \(\tau = a\). The accelerating scheme is to be designed for accelerating the process \("I_0 \rightarrow I_a\"\).

To accelerate the iteration process, in this study, we employ the MG scheme [35], which here we briefly introduce.

3.2 The Multigrid Scheme

MG techniques [35],[36] have been widely used to expedite relaxation processes in image processing [37]. MG methods can be used to provide numerical solutions to linear and nonlinear iterative problems of image processing. For linear problems of the form \(A(u) = f\), we utilize \(v\) to denote an approximation to the exact solution \(u\), and \(e\) \((e = u - v)\) to denote the compensation. Here, we define the residual as \(r = f - A(v) = A(u - v) = A(e)\).

In fact, the residual \(r\) consists of two components, a high frequency part and a low frequency part. The high frequency part of the residual is a local phenomenon, which comes from the inter-coupling between some adjacent mesh-grid points. The low frequency part is a global phenomenon, which comes from the boundary point. As the traditional relaxation method is a local effect, it can smooth the high frequency oscillations quickly, but the method does not work well for the low frequency part. Fortunately, the low frequency part can perform high frequency characteristics at coarse grid which can be eliminated efficiently by a local relaxation at coarse grid. While the new generated low frequency error can be transmitted to a coarser-grid repeatedly, until it reaches the coarsest-grid. We have all solutions back and can combine them from the high level to the lower level in turn. Finally, we can obtain a relatively accurate solution in the finest-grid.

In brief, MG is a recursive application of a two-grid process. The iterative method (such as Gauss-Seidel or Jacobi relaxation) can be applied to solve the fine-grid problem. These iterations have the property that, after relaxation, the noise will be smoothed.

Since the coarse grid is sparser than the fine grid, the computation is less expensive on the coarse grid. The fine grid residual \(r\) is computed and restricted to the coarse grid \(r_1 = P_1(r)\). \(P_1\) is a restriction operator to dampen the low frequency part of the residual, and \(e\) can be determined by solving the coarse grid residual equation \(A_1(e) = r_1\). Then we can perform prolongation to transmit \(e\) back to the fine grid to correct the fine grid approximation \(v\), i.e. \(v \leftarrow v + P_1(e)\), where \(P_1\) is prolongation operator. By recursively solving the coarse grid equation with this two-grid process, a MG algorithm is formed. This process is called the MG V-cycle (as shown in Fig. 3), as the algorithm starts with an initial estimate, telescopes down to the coarsest grid, and then returns to the finest grid[38].

3.3 Technical Details

We rewrite (9) with index \(n\) for distinguishing the fine and coarse grid levels, as in (11). In most MG based image processing practices, the scale of \(I^{(n)}\) is twice of that of \(I^{(n+1)}\).

\[ A^{(n)}(u^{(n)}) = 0. \]  

(11)

From the view of idiomatic expression in image processing problems, we use \(I^{(n)}\) and \(I^{(n+1)}\) instead of \(I^h\) and \(I^h\) to represent the fine and coarse grid components, respectively. In this paper, we set \(n \leq 4\), because \(n = 4\) represents the coarsest grid, and \(n = 0\) represents the finest grid.

In this paper, \(A(\cdot)\) is nonlinear and \(A(u - v) \neq A(u) - A(v)\), such that the residual of nonlinear \(A(\cdot)\) is as follows:

\[ A(u) - A(v) = r. \]  

(12)

There are two basic approaches for nonlinear MG. The first is to apply MG for the solution of the Jacobian system where a linearization scheme (such as the Newtons method) is employed at each iteration. The second approach is to apply MG directly to the nonlinear problem by employing the so called full approximation scheme (FAS) [39]. In the FAS, a nonlinear iteration is applied to smooth the error. The full equation is solved on the coarse grid, after which the coarse-grid error is extracted from the solution. This correction is then interpolated and applied to the fine grid approximation. In this study, we adopt Full MG (FMG) [38]
and a FAS scheme (Fig. 3, in our work we adopt a modified FAS with the final V-cycle omitted) to accelerate the form of (9) for every coarse level, and the whole system input is \( I^{(4)} \).

For input \( I^{(4)} \), we apply only a single relaxation for solving (13), which is equivalent to single step filtering.

\[
A^{(4)}(u^{(4)}) = 0.
\]  

(13)

With \( I^{(4)} \) denoting the result of (13), and, if the low frequency error in the coarsest grid is small, the result does not need compensation, \( I^{(4)} \) can be approximated by one step filtering \( \tilde{I}^{(4)} = I^{(4)} \otimes \kappa \), where \( \kappa \) is a nonlinear kernel of the filter represented in (8). \( \tilde{I}^{(4)} \) is prolonged to level-3 and filtered using (8), such that we obtained \( I^{(3)} \) as the initial status of the next V-cycle.

Our FMG scheme consists of several V-cycles (there are three V-cycles that need to be implemented in this study, see Fig. 3). Let \( I^{(n)} \) be an initial status of one V-cycle \( (0 < n < 4) \). It can also be viewed as an approximation to the original fine-grid problem (11).

The coarse-grid version of (12) is

\[
A^{(n+1)}(I^{(n+1)} + e^{(n+1)}) - A^{(n+1)}(I^{(n+1)}) = r^{(n+1)},
\]

(14)

Here \( A^{(n+1)} = (A^{(n)})_1 \) and \( I^{(n+1)} = (I^{(n)})_1 \).

The coarse-grid residual is chosen simply to be the restriction of the fine-grid residual

\[
r^{(n+1)} = (r^{(n)})_1 = (-A^{(n)}(I^{(n)}))_1.
\]

(15)

Making these substitutions in the coarse-grid residual equation yields

\[
A^{(n+1)}(I^{(n+1)} + e^{(n+1)}) - (A^{(n)}(I^{(n)}))_1 = A^{(n+1)}(I^{(n+1)}) - (A^{(n)}(I^{(n)}))_1.
\]

(16)

Assuming we can find \( u^{(n+1)} \), the coarse-grid error can be extracted from the solution by \( e^{(n+1)} = u^{(n+1)} - I^{(n+1)} \), and can be prolonged up to the fine grid and used to correct the fine grid approximation,

\[
I^{(0)} \leftarrow I^{(0)} + (e^{(n+1)})_1; \quad I^{(0)} \leftarrow I^{(0)} \otimes \kappa.
\]  

(17)

Filtering \( \otimes \kappa \) is used to smooth the noise produced by prolongation of (17). The coarser grid updating process is similar until attaining the coarsest grid.

\[
I^{(n+1)} \leftarrow I^{(n+1)} + (e^{(n+2)})_1; \quad I^{(n+1)} \leftarrow I^{(n+1)} \otimes \kappa.
\]  

(18)

At the coarsest grid, \( e^{(4)} \) can approximately be obtained by one step filtering,

\[
e^{(4)} = I^{(4)} \otimes \kappa - I^{(4)}.
\]  

(19)

The structure shown in (Fig. 3) is not original FMG, but a modified version where a V-cycle is omitted for time savings. The final result \( I^{(0)} \) is corrected by

\[
I^{(0)} \leftarrow I^{(0)} + (e^{(1)})_1; \quad I^{(0)} \leftarrow I^{(0)} \otimes \kappa.
\]  

(20)

Here, we provide the pseudo-code of a single V-cycle process as follows. Considering the ease of understanding involved, we also give the corresponding simulated functional exhibition (as shown in Fig. 4).

---

**Algorithm 1: pseudo-code of one V-cycle**

**input:** \( I^{(m)} \), \( (0 \leq n \leq N - 1) \);

**output:** \( I^{(n)} \);

**initial status:** \( m_1 = n, \ m_2 = N \).

For \( (m_1 < N) \)

restriction: \( I^{(m+1)} = (I^{(m_1)})_1; \)

\( m_1 = m_1 + 1; \)

end

relaxation: \( \tilde{I}^{(N)} \leftarrow I^{(N)} \otimes \kappa; \)

For \( (m_2 \geq n) \)

obtain compensation: \( e^{(m_2)} = I^{(m_2)} \otimes \kappa - \tilde{I}^{(m_2)}; \)

prolongation: \( I^{(m_1)} \leftarrow I^{(m_1)} + e^{(m_1-1)}; \)

compensation: \( \tilde{I}^{(m_1)} \leftarrow \tilde{I}^{(m_1)} + e^{(m_2-1)}; \)

filtering: \( \tilde{I}^{(m_1)} \leftarrow \tilde{I}^{(m_1)} \otimes \kappa; \)

\( m_2 = m_2 - 1; \)

end

**Restriction operator:** \( P_1 \)

The full-weighting restriction operator produces new coarse-grid points. Every value of the new coarse-grid point \( v_1(i, j) \) is just the weighted average of the values at the corresponding fine-grid point \( v(2i, 2j) \) and its eight nearest neighbours. The equation can be formulated as:

\[
v_1(i, j) = \frac{1}{16} (v(2i - 1, 2j - 1) + v(2i - 1, 2j + 1) + v(2i + 1, 2j - 1) + v(2i + 1, 2j + 1) + v(2i, 2j + 1) + v(2i, 2j - 1) + v(2i + 2, 2j) + 8v(2i, 2j)).
\]

(21)
Prolongation operator: $P$

In order to eliminate the artefact noise produced by restriction, the easiest and most natural method is to use a linear prolongation operator. The linear prolongation operator can be defined by $v = P_i(v_i)$ with the components of $v$ given as follows:

$$v(2i, 2j) = v_1(i, j),$$
$$v(2i+1, 2j) = \frac{1}{2}(v_1(i, j) + v_1(i+1, j)),$$
$$v(2i, 2j+1) = \frac{1}{2}(v_1(i, j) + v_1(i, j+1)),$$
$$v(2i+1, 2j+1) = \frac{1}{4}(v_1(i, j) + v_1(i+1, j) + v_1(i, j+1) + v_1(i+1, j+1)).$$

Relaxation operator

For every coarse grid, we employ a relaxation operator using a bilateral filtering with a $3 \times 3$ kernel. In order to speed up the relaxation process, we borrow the idea of 3DBF [28] to build look-up tables (LUTs), and thereby convert the BF from nonlinear to linear. In the following, we will give a brief introduction to this process.

For a given image $i(x)$, $i(x) \in \{0, 1, 2, \cdots, N-1\}$, where $N$ is the total number of grayscale values (in general, $N = 256$). With $i(x) = k$, we can define

$$W_k(\xi) = s(i(x), i(\xi)) = s(k, i(\xi)),$$

and

$$I_k(\xi) = s(i(x), i(\xi)) \cdot i(\xi) = W_k(\xi) \cdot i(\xi).$$

Then we rewrite the formulas for the BF as:

$$h^B(x) = k_B^{-1}(x) \int_{\xi \in \Omega(x)} c(\xi, x) \cdot s(i(\xi), i(x)) d\xi,$$

where

$$k_B(x) = \int_{\xi \in \Omega(x)} c(\xi, x) \cdot W_k(\xi) d\xi.$$

The values of $W_k(\xi)$ and $I_k(\xi)$ are determined by $(i(\xi), i(x)) = (\zeta, k)$ (where $\zeta \in [0, 255], k \in [0, 255]$) and $(\sigma_c, \sigma_r)$. Since $(\sigma_c, \sigma_r)$ are fixed values in a fixed coarse level, we can build two LUTs to hold the values of $W_k(\xi)$ and $I_k(\xi)$ at every level. When we need to compute the intensity value at $(\xi, x)$ in the filtered image, we can look-up the corresponding values of $W_k(\xi)$ and $I_k(\xi)$ in the LUTs. Because the size of BF we use in this study is less than 256, the value of $c(\xi, x)$ in (2) can also be known by recourse to the LUTs. The usage of LUTs converts bilateral filtering from nonlinear to linear, which can accelerate the processing procedure of the BF to a great degree.

Choosing the parameters

In this paper, each relaxation is equivalent to single step filtering by (8). It is well know that a BF has two parameters to be adjusted. For a given image, it is a challenging task to choose $5 \times 2$ parameters. In order to simplify this process, numerous experiments are conducted. On the basis of abundant tests and observations, we find that the number of parameters which need to be chosen can be reduced to 3.

From C. Tomasi and R. Manduchi’s work [9], we know that the domain filter (as shown in (2)) acts as a standard Gaussian filter, while range filtering (as shown in (3)) can preserve the edges. The experimental results show that the range filter has a greater contribution to the processing results than the domain filter. In order to simplify the process, we define $\sigma_r = \sigma_c$.

For each scale, there are 3 parameters needed to be adjusted. For convenience, we defined the parameter in N-scale as $\alpha_{5-N}$. By analyzing the experimental results, we
find the functions and relationships of these 5 parameters are as follows.

1. $\alpha_1$ and $\alpha_2$ are the resource of blocking artifacts. The value of these can not be over large.

2. $\alpha_4$ and $\alpha_5$ are the resource of ringing and ghosting artifacts. Their values also can not be over large.

3. $\alpha_1$ should not be larger than $\alpha_2$. To simplify, we choose $\alpha_2 = 2\alpha_1$.

4. $\alpha_4$ should not be larger than $0.5 \times \alpha_5$, or we will haveghosting artifacts. To simplify, we set $\alpha_5 = 2\alpha_4$.

5. The value of $\alpha_3$ is independent of the remaining parameters. Its value can not be too large or too small. There will be ghosting artifacts under a large value, and blocking artifacts under a small value (here, we set $\alpha_3$ between 10 and 40).

Hence, the number of user-specified parameters in our method is 3, and the relationships of these 5 parameter is $\alpha_5 = 2\alpha_4$, $\alpha_2 = 2\alpha_1$, $10 \leq \alpha_3 \leq 40$.

3.4 Differences from Multiscale Based Methods

As mentioned before, most of the fastest versions of the BF are based on downsampling or multiscale ideas [12], [22], [23], [28], [29]. Although the MGBF is similar to the above methods in some ways, they are different in theoretical and fundamental aspects.

For the MGBF, the accelerating scheme focuses on solving nonlinear equations and improving convergent speed (or the reduction in the number of iterations). Multiscale based methods pay more attention to the number of points required for computing, where downsampling may be the most direct and effective way. For the MGBF, we analyze the characteristics of the residuals, and utilize fine-grid and coarse-grid schemes to eliminate different parts of residuals (or high and low frequency errors). While, for multiscale based methods, the filtering process in different scales aims at eliminating the errors produced by downsampling and interpolation.

4. Reconsideration of the Relationship Between the BF and Nonlinear Diffusion

For adaptive filtering, a bridge is built between bilateral filtering and nonlinear diffusion in [33]. It is easy to find that single incidence of bilateral filtering is equivalent to a one step nonlinear diffusion process. In this study, we assume that single incidence bilateral filtering with a large kernel can be approximated by a version using a $3 \times 3$ kernel size by applying the small version many times on the premise of visual acceptance. In other words, single incidence bilateral filtering with a large kernel can be approximated by a nonlinear diffusion process with a certain length of time. Accelerating the bilateral filtering is equivalent to reducing the time of the diffusion process. Therefore, a link between fast bilateral filtering and nonlinear diffusion is built with the MG scheme, which focuses more attention on numerical calculus (as shown in Fig. 5).

5. Experiments

In this section, we give our experimental results to show that our method can accelerate the process of bilateral filtering without influencing its visual effect.

To show the advantage of the new method in speeding up the BF, we compare the processing time and the visual quality of results (using the peak signal-to-noise ratio, PSNR) for our MGBF method, RTBF (Real-Time o(1) Bilateral Filtering [28]) and 3DBF (A fast approximation of the bilateral filter using a signal processing approach [22]).

We implement the RTBF and 3DBF algorithms in C. For RTBF and MGBF, we test two versions with different parameters, indicated by RTBF1, RTBF2, MGBF1 and MGBF2, respectively. Therefore, we have five testing experiments: RTBF1, RTBF2, MGBF1, MGBF2 and 3DBF. The five group parameters are shown as follows:

RTBF1:
RangeScale=7; SpatialDownsampleRate=3;
SpatialFilter=0; SpatialSigma=0.03;
RangeSigma=0.05;

RTBF2:
RangeScale=4; SpatialDownsampleRate=3;
SpatialFilter=0; SpatialSigma=0.03;
RangeSigma=0.12;

MGBF1: $\alpha_1$ to $\alpha_5 = 0.125$ 0.25 10 7.5 15;
MGBF2: $\alpha_1$ to $\alpha_5 = 0.05$ 0.1 5 5 10;

3DBF:
SpatialSigma=16.0;
RangeSigma=0.05;

Readers can refer to [22], [28] for details concerning parameter choice.
5.1 Speed Evaluation

To test our accelerating scheme, we design three experiments. Figure 6 shows nine natural images and one synthetic image. We apply RTBF1, RTBF2, 3DBF and MGBF1 to these data, and record the corresponding run-times, as shown in Table 2. For MGBF, the running time is independent of the parameters, and we choose only MGBF1 for speed evaluation. From inspection of Table 2, we find that our method can achieve a better record for accelerating the filtering process than the other two fast approaches.

To find the relationship between runtime and the size of the image, we conduct another experiment: testing images with different sizes. Table 3 reveals the time consumption with different image sizes by RTBF1, 3DBF and MGBF1. To facilitate visual comparison, we provide the corresponding curve graphs of Table 3 in Fig. 8.

![Fig. 6](image1.png) Test images employed in the experiments (different images with different sizes).

![Fig. 7](image2.png) Test image.

![Fig. 8](image3.png) Relationship between the run time and the size of the test image. (A) Image sizes from 0.047 to 2.562M. (B) A local region of (A) with sizes from 0.072 to 0.167M.

Table 2 Time consumption of different images for the different methods employed.

<table>
<thead>
<tr>
<th>Image(No)</th>
<th>Runtime(ms)</th>
<th>RTBF1</th>
<th>RTBF2</th>
<th>3DBF</th>
<th>MGBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>flower(1)</td>
<td>91.97</td>
<td>63.88</td>
<td>86.89</td>
<td></td>
<td>57.03</td>
</tr>
<tr>
<td>rock(2)</td>
<td>175.54</td>
<td>125.40</td>
<td>163.39</td>
<td>87.18</td>
<td></td>
</tr>
<tr>
<td>building(3)</td>
<td>171.00</td>
<td>121.13</td>
<td>165.10</td>
<td>85.06</td>
<td></td>
</tr>
<tr>
<td>dome(4)</td>
<td>94.74</td>
<td>67.12</td>
<td>90.36</td>
<td>59.70</td>
<td></td>
</tr>
<tr>
<td>dragon(5)</td>
<td>169.36</td>
<td>120.83</td>
<td>170.36</td>
<td>86.33</td>
<td></td>
</tr>
<tr>
<td>housecorner(6)</td>
<td>112.13</td>
<td>79.47</td>
<td>105.47</td>
<td>64.56</td>
<td></td>
</tr>
<tr>
<td>swamp(7)</td>
<td>117.57</td>
<td>83.01</td>
<td>126.20</td>
<td>66.41</td>
<td></td>
</tr>
<tr>
<td>synthetic(8)</td>
<td>95.31</td>
<td>69.78</td>
<td>85.36</td>
<td>64.08</td>
<td></td>
</tr>
<tr>
<td>temple(9)</td>
<td>146.71</td>
<td>107.08</td>
<td>147.39</td>
<td>77.89</td>
<td></td>
</tr>
<tr>
<td>turtle(10)</td>
<td>166.94</td>
<td>118.06</td>
<td>167.17</td>
<td>85.65</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Relationship between the runtime and the size of the test image.

<table>
<thead>
<tr>
<th>No</th>
<th>Size(M)</th>
<th>Runtime(ms)</th>
<th>RTBF</th>
<th>3DBF</th>
<th>MGBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>180 × 273 = 0.047M</td>
<td>18.43</td>
<td>19.73</td>
<td>33.61</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>200 × 303 = 0.059M</td>
<td>23.66</td>
<td>22.37</td>
<td>36.07</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>222 × 336 = 0.072M</td>
<td>26.93</td>
<td>27.66</td>
<td>36.65</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>246 × 373 = 0.089M</td>
<td>32.62</td>
<td>31.81</td>
<td>39.94</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>273 × 414 = 0.110M</td>
<td>42.74</td>
<td>40.23</td>
<td>41.16</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>303 × 460 = 0.136M</td>
<td>51.40</td>
<td>49.17</td>
<td>44.46</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>336 × 511 = 0.167M</td>
<td>61.81</td>
<td>57.58</td>
<td>44.99</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>373 × 567 = 0.206M</td>
<td>76.23</td>
<td>69.46</td>
<td>51.53</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>414 × 630 = 0.254M</td>
<td>92.47</td>
<td>86.18</td>
<td>58.38</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>460 × 700 = 0.311M</td>
<td>115.79</td>
<td>105.91</td>
<td>64.58</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>511 × 777 = 0.387M</td>
<td>138.25</td>
<td>131.36</td>
<td>73.49</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>567 × 863 = 0.477M</td>
<td>169.93</td>
<td>169.84</td>
<td>83.71</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>630 × 958 = 0.589M</td>
<td>218.87</td>
<td>212.83</td>
<td>95.86</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>699 × 1064 = 0.729M</td>
<td>257.30</td>
<td>264.35</td>
<td>112.09</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>776 × 1182 = 0.895M</td>
<td>313.08</td>
<td>322.54</td>
<td>132.54</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>862 × 1313 = 1.105M</td>
<td>388.15</td>
<td>375.09</td>
<td>155.05</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>957 × 1458 = 1.388M</td>
<td>476.08</td>
<td>439.06</td>
<td>182.72</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1063 × 1620 = 1.681M</td>
<td>586.71</td>
<td>539.85</td>
<td>226.04</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1181 × 1800 = 2.075M</td>
<td>724.54</td>
<td>660.49</td>
<td>266.63</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1312 × 2000 = 2.562M</td>
<td>946.17</td>
<td>813.45</td>
<td>321.98</td>
<td></td>
</tr>
</tbody>
</table>
space for memory (which take about 25ms), and the advantage is not so significant when the size of the image is under 0.136M.

Another test provides further evidence of the advantage of our method for larger image sizes. We perform the test (all images are the same size (3744 x 2160), as shown in Fig. 9) to compare the runtime (Table 4) of 3DBF, RTBF1 and our method MGBF1. Figure 10 shows the histogram indicative of Table. 4 for a more intuitive comparison. From examination of Fig. 10 and Table. 4, we find that our filter is the fastest filter (about three times faster than the others).

5.2 Quality Evaluation

In addition to speed, another important evaluation of a fast BF is the numerical accuracy (or error induced by the ac-

Table 4  Time consumption of different images of the same size.

<table>
<thead>
<tr>
<th>No.</th>
<th>3DBF</th>
<th>RTBF</th>
<th>MGBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4341.99</td>
<td>2799.28</td>
<td>955.03</td>
</tr>
<tr>
<td>2</td>
<td>4253.01</td>
<td>2828.41</td>
<td>907.85</td>
</tr>
<tr>
<td>3</td>
<td>4409.22</td>
<td>2848.84</td>
<td>963.33</td>
</tr>
<tr>
<td>4</td>
<td>4395.38</td>
<td>2859.14</td>
<td>918.29</td>
</tr>
<tr>
<td>5</td>
<td>4417.37</td>
<td>2760.78</td>
<td>943.14</td>
</tr>
<tr>
<td>6</td>
<td>4471.80</td>
<td>2900.08</td>
<td>954.95</td>
</tr>
<tr>
<td>7</td>
<td>4441.40</td>
<td>2881.85</td>
<td>965.41</td>
</tr>
<tr>
<td>8</td>
<td>4421.63</td>
<td>2867.78</td>
<td>930.62</td>
</tr>
<tr>
<td>9</td>
<td>4553.12</td>
<td>2842.69</td>
<td>1037.42</td>
</tr>
<tr>
<td>10</td>
<td>4448.70</td>
<td>2857.33</td>
<td>939.54</td>
</tr>
<tr>
<td>Average</td>
<td>4415.36</td>
<td>2848.22</td>
<td>951.56</td>
</tr>
</tbody>
</table>

Table 5  Quality comparisons with different filters under different ground truths.

<table>
<thead>
<tr>
<th>Methods</th>
<th>(0, 0)</th>
<th>(30, 10)</th>
<th>(30, 3)</th>
<th>(10, 10)</th>
<th>(10, 10)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flower</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>34.71</td>
<td>38.38</td>
<td>36.22</td>
<td>38.46</td>
<td>36.25</td>
<td>36.80</td>
</tr>
<tr>
<td>RTBF2[28]</td>
<td>29.16</td>
<td>34.17</td>
<td>30.59</td>
<td>30.85</td>
<td>29.96</td>
<td>30.95</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>34.57</td>
<td>38.56</td>
<td>36.33</td>
<td>39.61</td>
<td>36.43</td>
<td>37.10</td>
</tr>
<tr>
<td>MGBF1</td>
<td>35.49</td>
<td>37.56</td>
<td>39.61</td>
<td>40.32</td>
<td>39.09</td>
<td>38.42</td>
</tr>
<tr>
<td>MGBF2</td>
<td>38.03</td>
<td>35.59</td>
<td>40.49</td>
<td>43.69</td>
<td>46.68</td>
<td>40.90</td>
</tr>
<tr>
<td>Rock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>36.42</td>
<td>33.02</td>
<td>33.14</td>
<td>39.49</td>
<td>38.00</td>
<td>36.01</td>
</tr>
<tr>
<td>RTBF2[28]</td>
<td>27.87</td>
<td>32.44</td>
<td>30.32</td>
<td>28.87</td>
<td>28.59</td>
<td>29.62</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>36.46</td>
<td>33.42</td>
<td>33.53</td>
<td>42.53</td>
<td>39.27</td>
<td>37.04</td>
</tr>
<tr>
<td>MGBF1</td>
<td>32.87</td>
<td>32.76</td>
<td>32.08</td>
<td>35.08</td>
<td>35.34</td>
<td>33.63</td>
</tr>
<tr>
<td>MGBF2</td>
<td>37.09</td>
<td>31.27</td>
<td>32.71</td>
<td>38.94</td>
<td>40.16</td>
<td>36.03</td>
</tr>
<tr>
<td>Dome</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>35.71</td>
<td>34.88</td>
<td>34.71</td>
<td>38.3</td>
<td>36.92</td>
<td>36.10</td>
</tr>
<tr>
<td>RTBF2[28]</td>
<td>28.46</td>
<td>33.87</td>
<td>31.07</td>
<td>29.44</td>
<td>29.08</td>
<td>30.38</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>34.57</td>
<td>34.35</td>
<td>34.75</td>
<td>33.78</td>
<td>33.24</td>
<td>33.00</td>
</tr>
<tr>
<td>MGBF1</td>
<td>34.17</td>
<td>34.01</td>
<td>35.88</td>
<td>36.62</td>
<td>36.80</td>
<td>35.50</td>
</tr>
<tr>
<td>MGBF2</td>
<td>37.36</td>
<td>32.65</td>
<td>34.72</td>
<td>40.26</td>
<td>41.42</td>
<td>37.28</td>
</tr>
<tr>
<td>Housecorner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>33.40</td>
<td>33.05</td>
<td>31.63</td>
<td>38.01</td>
<td>36.35</td>
<td>34.05</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>34.06</td>
<td>31.07</td>
<td>32.10</td>
<td>39.63</td>
<td>36.51</td>
<td>34.67</td>
</tr>
<tr>
<td>MGBF1</td>
<td>30.36</td>
<td>30.80</td>
<td>33.14</td>
<td>33.03</td>
<td>32.98</td>
<td>32.06</td>
</tr>
<tr>
<td>MGBF2</td>
<td>34.59</td>
<td>28.54</td>
<td>30.99</td>
<td>37.38</td>
<td>38.56</td>
<td>34.01</td>
</tr>
<tr>
<td>Synthetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>37.01</td>
<td>33.52</td>
<td>34.84</td>
<td>37.97</td>
<td>37.52</td>
<td>36.18</td>
</tr>
<tr>
<td>RTBF2[28]</td>
<td>28.97</td>
<td>31.75</td>
<td>31.27</td>
<td>29.34</td>
<td>29.18</td>
<td>30.10</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>33.28</td>
<td>32.68</td>
<td>32.99</td>
<td>39.46</td>
<td>36.21</td>
<td>34.92</td>
</tr>
<tr>
<td>MGBF1</td>
<td>30.00</td>
<td>33.32</td>
<td>35.38</td>
<td>33.72</td>
<td>33.48</td>
<td>33.22</td>
</tr>
<tr>
<td>MGBF2</td>
<td>33.60</td>
<td>30.16</td>
<td>33.03</td>
<td>37.50</td>
<td>38.77</td>
<td>34.61</td>
</tr>
<tr>
<td>Turtle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTBF1[28]</td>
<td>36.03</td>
<td>30.60</td>
<td>33.42</td>
<td>39.38</td>
<td>40.65</td>
<td>36.07</td>
</tr>
<tr>
<td>RTBF2[28]</td>
<td>27.72</td>
<td>32.76</td>
<td>29.92</td>
<td>28.90</td>
<td>28.45</td>
<td>29.55</td>
</tr>
<tr>
<td>3DBF[22]</td>
<td>35.10</td>
<td>32.59</td>
<td>32.99</td>
<td>34.17</td>
<td>34.07</td>
<td>35.81</td>
</tr>
<tr>
<td>MGBF1</td>
<td>31.88</td>
<td>32.20</td>
<td>34.84</td>
<td>34.50</td>
<td>34.59</td>
<td>33.60</td>
</tr>
<tr>
<td>MGBF2</td>
<td>33.55</td>
<td>29.68</td>
<td>31.57</td>
<td>37.68</td>
<td>38.57</td>
<td>34.21</td>
</tr>
</tbody>
</table>
Table 6  Time consumption of different filters for gray-scale image filtering.

<table>
<thead>
<tr>
<th>Method</th>
<th>Times (ms/Mp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Transform (NC)</td>
<td>160</td>
</tr>
<tr>
<td>Domain Transform (RF)</td>
<td>60</td>
</tr>
<tr>
<td>Ours</td>
<td>143</td>
</tr>
</tbody>
</table>

Fig. 11  Filtering results. (a) The original image, (b) filtered image by the Domain Transform filter (RF) with $\sigma_r = 0.4, \sigma_s = 30$, (c) filtered image by the Domain Transform filter (NC) with $\sigma_r = 0.4, \sigma_s = 30$, (d) filtered image by our method with parameters: $\alpha_1$ to $\alpha_5 = 1.5 3 20 5 10$.

Fig. 12  PSNR comparisons between different filters under different ground truths. (a) The parameters of ground truth are $(\sigma_r, \sigma_s) = (30, 10)$, (b) the parameters of ground truth are $(\sigma_r, \sigma_s) = (10, 3)$, (c) the parameters of ground truth are $(\sigma_r, \sigma_s) = (30, 3)$; (d) the average performance of a, b and c.

PSNR = $-10 \log_{10} \left( \frac{1}{|\Omega|} \sum_{x \in \Omega} |I_{MGBF}(x) - I_{BF}(x)|^2 \right)$

(26)

to quantify the error induced by the accelerating approximation. Where $I_{MGBF}$ and $I_{BF}$ represent the filtering results of our MGBF method and the BF respectively. To compare the PSNR of the images filtered by 3DBF, RTBF and our MGBF method, we need to choose a unified ground truth. However, there is, in fact, no unified ground truth for the original BF with different parameters. Therefore, we choose the original input and four results by BF with different parameters as the ground truth (as shown in Table. 5, (0,0), (30,10), (30,3), (10,10) and (10,3) represent the ground truths with different parameters $(\sigma_r, \sigma_s)$). The test images are shown in Fig. 9, and Table. 5 presents the PSNR evaluations of RTBF1, RTBF2, MGBF1, MGBF2 and 3DBF values under different ground truths. The average PSNR evaluations over different ground truths are also provided. Figure 12 presents the graph indication of Table. 5 for a more intuitive comparison.

The results of PSNR given in Table. 5 and Fig. 12 show that our method does not suffer a loss in image quality, and the PSNR values of our method are close to that of 3DBF. Since our method achieves acceleration through approximation, we have chosen to measure the numerical accuracy by comparing our technique with other existing approximations. This comparison pertains only to the numerical quality of the approximation, and readers should bear in mind that numerical differences do not necessarily produce unsatisfying outputs. To balance this numerical aspect, we also provide visual results (shown in Fig. 13) to allow examination the output. Important criteria are then the regularity of the achieved smoothing, artefacts that add visual features which do not exist in the input picture, tone (or colour) faithfulness and so on. From the results and analysis of our experiments, we can conclude that our method accelerates the runtime of the BF without sacrificing the image quality.

5.3 Comparisons with Other Edge-Preserving Filter

Recently, Gastal and Oliveira [40] proposed another edge-preserving filter—Domain Transform filter. The key idea of Gastal’s work is to iteratively and separately apply 1D edge-aware filters. Domain Transform filter is $O(N)$ time complexity which is achieved by integral images or recursive filtering. To show the significance of our method in speeding up the BF, We will compare our filter with Domain Transform filter in this section.

The comparison of running time is listed in Table 6. As shown in Fig. 11, though the domain transform is faster than MGBF, it simply smoothing the image and does not have a good structure-transferring ability.
The BF has been proven to be very useful but time consuming. It is nonlinear and computationally expensive which restricts its real-time applications. The effort of this study is to design a new algorithm to accelerate the filtering process. A fast version of BF (MGBF) is presented in this paper. We use nonlinear diffusion to approximate bilateral filtering with a large kernel, and the desired filtering result is an intermediate status of the overall diffusion process. Acceleration implies that a shorter time should be taken to achieve the desired solution. Iterative linear equation techniques are sufficiently mature to cope with the nonlinear diffusion equation, which can be accelerated using the MG scheme. In this study, we adopt the nonlinear FMG structure. Experimental results with both simulated data sets and real sets show that the new method can achieve almost twice the speed of the state-of-the-art. Barash first proposed that the nature of bilateral filtering resembles that of anisotropic diffusion [33], in which an adaptive filter is used to build a link between bilateral filtering and nonlinear diffusion. But for numerical calculus, they are still two distinct computational methods. In this paper, we present another relationship between bilateral filtering and nonlinear diffusion: that bilateral filtering can be approximated and accelerated by nonlinear diffusion based on the FMG scheme. Figure 5 shows a visual expression and comparison between our work and Barash’s [33]. The new relationship focuses more attention on numerical calculus.

Acknowledgments

The authors would like to thank the support of the National Natural Science Foundation of China (No. 11002082 and No. 61271393), Shenzhen STP (JCYJ20130402145002441) and Huawei Innovation Research Program(IRP-2013-14-17).

6. Conclusion

The BF has been proven to be very useful but time consuming. It is nonlinear and computationally expensive which restricts its real-time applications. The effort of this study is to design a new algorithm to accelerate the filtering process. A fast version of BF (MGBF) is presented in this paper. We use nonlinear diffusion to approximate bilateral filtering with a large kernel, and the desired filtering result is an intermediate status of the overall diffusion process. Acceleration implies that a shorter time should be taken to achieve the desired solution. Iterative linear equation techniques are sufficiently mature to cope with the nonlinear diffusion equation, which can be accelerated using the MG scheme. In this study, we adopt the nonlinear FMG structure. Experimental results with both simulated data sets and real sets show that the new method can achieve almost twice the speed of the state-of-the-art. Barash first proposed that the nature of bilateral filtering resembles that of anisotropic diffusion [33], in which an adaptive filter is used to build a link between bilateral filtering and nonlinear diffusion. But for numerical calculus, they are still two distinct computational methods. In this paper, we present another relationship between bilateral filtering and nonlinear diffusion: that bilateral filtering can be approximated and accelerated by nonlinear diffusion based on the FMG scheme. Figure 5 shows a visual expression and comparison between our work and Barash’s [33]. The new relationship focuses more attention on numerical calculus.


[34] Q. Yang, “Hardware-efficient bilateral filtering for stereo matching,” 2013.


