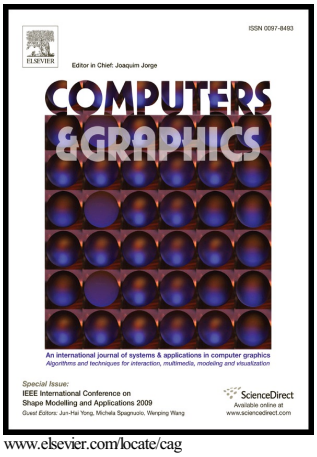
Real-Time Edge-Aware Weighted Median Filtering on the GPU

Hanli Zhao, Dandan Gao, Ming Wang, Zhigeng Pan



www.elsevier.com/locate/cag

PII: S0097-8493(16)30112-1
DOI: http://dx.doi.org/10.1016/j.cag.2016.09.003
Reference: CAG2749

To appear in: *Computers and Graphics*

Received date: 13 June 2016
Revised date: 10 September 2016
Accepted date: 19 September 2016

Cite this article as: Hanli Zhao, Dandan Gao, Ming Wang and Zhigeng Pan Real-Time Edge-Aware Weighted Median Filtering on the GPU, *Computers an Graphics,* http://dx.doi.org/10.1016/j.cag.2016.09.003

Real-Time Edge-Aware Weighted Median Filtering on the GPU

Hanli Zhao[a], Dandan Gao[a], Ming Wang[a], Zhigeng Pan[b,*]

[a]*Intelligent Information Systems Institute, Wenzhou University, Wenzhou 325035, China*
[b]*Digital Media and HCI Research Center, Hangzhou Normal University, Hangzhou 311121, China*

Abstract

Edge-preserving weighted median filtering is a fundamental operator in a great variety of image processing and computer graphics applications. This paper presents a novel real-time weighted median filter which effectively smoothes out high-frequency details while preserving major edges. We propose an edge-aware data structure, the 4-dimensional bilateral grid, by extending the 3-dimensional bilateral grid with an additional range dimension. The edge-aware weights and the weighted median values are computed in the 4-dimensional space. The proposed algorithm is highly parallel, enabling a real-time GPU-based implementation. Experimental results show that our filter can be run in real-time on modern GPUs. A variety of edge-aware applications, including detail enhancement, inverse halftoning, JPEG artifact removal, image stylization, photographic look transfer and image colorization, are demonstrated to verify the feasibility of the proposed weighted median filter.

Keywords: median filter, bilateral filter, bilateral grid, GPU

1. Introduction

Edge-preserving smoothing filters aim at reducing image noises while preserving meaningful structures. The research on developing new effective edge-preserving smoothing filters has been conducted for decades. The median filter [21] has been proven to be more effective than a linear smoothing filter in preserving edges. Each neighbor pixel contributes to an equal weight to the filtered central pixel in the standard median filter. Therefore, the standard median filter cannot preserve high-contrast boundaries well. The bilateral filter [18] is a widely used edge-preserving filter in the image processing and computer graphics community, and many improvements have been made in recent years. However, due to the nature of weighted mean of neighboring pixels, noises also contribute to the output value of the filtered pixel. On the contrary, the weighted median filter (WMF) directly selects the neighbor pixel with a weighted median value to substitute the original value of the filtered pixel. When the edge-aware weight is employed, the weighted median filter has the ability to preserve salient edges. As a consequence, weighted median filtering can produce better edge-preserving smoothing results than bilateral filtering in many cases.

In recent years, many algorithms have been proposed in order to improve the filtering efficiency. Zhang et al. [31] proposed a few efficient schemes to achieve a linear computation complexity with regard to the kernel size. However, their algorithm is designed for fast CPU serial implementation but not suitable for GPU parallelization. Ma et al. [13] proposed a constant time weighted median filtering algorithm by elevating the computation in the 3-dimensional (3D) space. The constant time guided filter [4] is employed to filter the data in the 3D volume with edge-preservation. Similarly, Yang et al. [26] formulated both the median and bilateral filtering as a unified constant-time cost volume aggregation framework. Their algorithm implements the bilateral filter with recursive schemes and incorporates their bilateral filter into the constant time edge-preserving weighted median smoothing in the 3D space. We observe that there is room to further improve the performance of weighted median filtering by exploiting the computation on even higher dimension.

Driven by the insatiable market demand for real-time processing, the programmable GPU has evolved into a highly parallel manycore processor with great computational power. Currently, the GPU has been worked as a co-processor of the CPU to address problems that can be expressed as data-parallel computations [14]. In particularly, a raster image contains regularly aligned pixel matrix and the processing on each pixel is inherently highly parallel. In this paper, we would like to transfer the major workload of the filtering on the GPU by exploiting the parallelism of the algorithm.

In this paper, we elevate the computation domain into a 4-dimensional (4D) space to even improve the performance of weighted median filtering. This is achieved by extending the use of the data structure of bilateral grid [2]. The original bilateral grid is used to approximate the bilateral filtering in a 3D space. It is highly efficient that the time complexity is linear with respect to the computation grid. In our work, we adapt the original bilateral grid as a new edge-aware 4D bilateral grid and scatter the pixel data of the input image into our 4D bilateral grid. As a result, the edge-aware weights and the weighted median values are computed in the 4D grid domain. Extensive experimental results and discussions are presented at the end of the paper to demonstrate the effectiveness of our GPU-based implementation. We show our new filter benefits to a

*Corresponding author. E-mail: zgpan@cad.zju.edu.cn

number of edge-aware applications in the context of detail enhancement, inverse halftoning, JPEG artifact removal, image stylization, photographic look transfer and image colorization.

This paper significantly extends our preliminary conference version which has been published in Edutainment '16 [32]. In this paper, we further provide detailed performance comparison and analysis with state-of-the-art methods, and implement a number of edge-aware image processing applications based on our new filter. This paper has the following contributions:

- By incorporating the 3D bilateral grid with an additional range dimension, a novel edge-preserving weighted median filter with the edge-aware 4D bilateral grid is developed.
- A real-time GPU-based weighted median filter is implemented using CUDA, allowing for various efficient edge-aware image processing.

The remainder of this paper is organized as follows. Section 2 briefly reviews some of previous work. Next, Section 3 describes our algorithm in detail. Experimental results and discussions are presented in Section 4 while edge-aware applications are elaborated in Section 5. Finally, we conclude the paper in the last section.

2. Related Work

A variety of edge-preserving image smoothing filters have been proposed for decades. In this section, we briefly review some related work.

The unweighted median filter was first introduced by Tukey [19]. Then Hang et al. [5] presented a $O(r)$ median filtering algorithm based on storing and updating the gray level histogram of the picture elements in the window. Weiss [21] introduced a vectorable $O(\log r)$ median filtering algorithm based on distributed histograms. Cline et al. [3] presented a constant time 8-bit median filter implementation based on a separable argument while Perreault and Hébert [16] proposed a constant time median filter by reducing the update of the column and kernel histograms as well as the computing of the median. Kass and Solomon [6] computed accurate derivatives and integrals of locally-weighted histograms over large neighborhoods and enables a constant time isotropic median filter. These unweighted median filtering algorithms are well suited for CPU implementation and thus cannot be parallelized using GPU. In addition, the median filter may fail to preserve high-contrast image boundaries.

There have been some weighted median filters by taking into account high-contrast edges. A comprehensive tutorial paper was written by Yin et al. [27]. Rhemann et al. [17] employed the GPU-based weighted median filter as a post-processing step to remove streak-like artifacts in the disparity map while preserving the object boundaries. Bilateral filtering weights are used for the median filter weights. Zhang et al. [31] proposed a joint-histogram representation, median tracking and a new data structure that enables fast data access. Although their approach reduces computation complexity to $O(r)$, the acceleration schemes are highly serial and cannot be implemented on the GPU. The algorithm of Yang et al. [26] computes the range distance for all cell with the same spatial coordinate for each pixel and formulates the weighted median filter as a cost aggregation problem followed by a winner-take-all selection. Yang et al. further employed the recursive bilateral filtering [25] for constant time edge-preserving median weights. Unfortunately, the GPU parallelization is not involved in their algorithm. Ma et al. [13] developed the first constant time algorithm for the previously time-consuming weighted median filter. All cells of the 3D volume is initialized as zero, and each image pixel is scattered into corresponding cell based on its spatial coordinate and range coordinate. The constant time guided filter [4] is adopted to compute the edge-aware median weights. Ma et al. implemented their weighted median filter using CUDA to improve the performance. We observe that the performance of the weighted median filter can be further optimized.

Some image edge-preserving filtering algorithms related to our work are also reviewed here. Zhang et al. [30] proposed a color scheme replacement method to replace each segmented region by a desired color style and then performed boundary smoothing and edge overlaying to produce more artistic effects. Xu et al. proposed to apply a global L0 gradient minimization [23] for effective edge-preserving image smoothing and further introduced a relative total variation measure [24] for the removal of textures from meaningful structures. Li et al. [12] solved an edge-aware global optimization problem based on a Gaussian pyramid and an explicit mixed-domain. Zang et al. [28] used a space-filling curve as the reduced domain for efficient structure-aware smoothing. Recently Zang et al. [29] designed a texture smoothing technique based on a guided adaptive empirical mode decomposition. Wang et al. [20] proposed least-squares images as a basis for their edge-preserving image smoothing method.

3. Our Algorithm

First of all, let us introduce the definition of weighted median filter. The weighted median filter is a nonlinear rank order based filter that smoothes an image while preserving major edges. For a box filter window $N(\mathbf{p})$ centered at the pixel $\mathbf{p}$ in an image I, a non-negative weight $w(\mathbf{p},\mathbf{q})$ is assigned to each neighbor pixel $\mathbf{q} \in N(\mathbf{p})$. Let M be the filtered output, the weighted median of $N(\mathbf{p})$ is defined as the value $M(\mathbf{p})$ minimizing the following sum expression [27]:

$$M(\mathbf{p}) = \arg\min_{b} \sum_{\mathbf{q}\in N(\mathbf{p})} w(\mathbf{p},\mathbf{q})|I(\mathbf{q}) - b|, \tag{1}$$

where $b \in [0, 255]$ which contains a total of 256 integral numbers for an input 8-bit digital image. If the value of $w(\mathbf{p},\mathbf{q})$ corresponds to the weight of an edge-preserving smoothing filter, the weighted median filter can produce an edge-preserving smoothing output. In particularly, the bilateral filter weight [18] is commonly used to compute the edge-aware weight:

$$w(\mathbf{p},\mathbf{q}) \quad = \quad G(\mathbf{p}-\mathbf{q},\sigma_1)G(I(\mathbf{p}) - I(\mathbf{q}),\sigma_2) \tag{2}$$

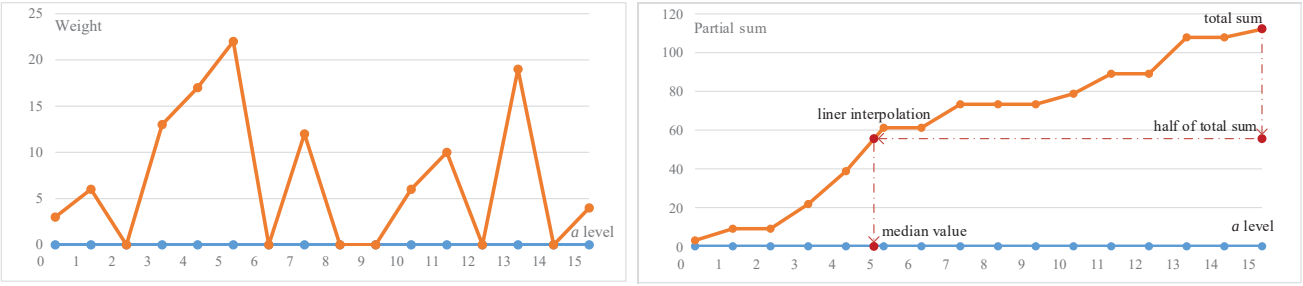


Figure 1: Illustration of median computing on a pixel: (left) sliced Γ weights based on the a level (right) the corresponding partial sum of weights and the interpolated median value.

$$G(\mathbf{x},\sigma) = \exp(-\frac{\|\mathbf{x}\|^2}{\sigma^2}), \tag{3}$$

where G is the univariate Gaussian function of the variance parameter σ.

According to the definition of weighted median filter, the output $M(\mathbf{p})$ can be calculated after a simple sorting. Firstly, sort the weighted values inside $N(\mathbf{p})$. Secondly, add up the weights from the beginning of the sorted set until the sum being equal to half of the total sum of weights. Finally, the value of $M(\mathbf{p})$ corresponds to the pixel value with the last weight added. Ma et al. [13] and Yang et al. [26] have proposed their respective constant time implementations of the weighted median filter by elevating the computation domain into 3D volumes. In our approach, we aim to compute the weighted median filter in an even higher dimension to obtain more performance gain. In the following, we will describe our GPU-based edge-preserving weighted median filter in detail.

In this paper, we extend the original bilateral grid to define a new 4D data structure for our weighted median filtering. The bilateral grid [2] is a GPU-based data structure for efficient bilateral filtering. The original bilateral grid is defined as a 3D array, where the first two dimensions (x, y) correspond to 2D spatial coordinates in the image while the third dimension z corresponds to the image intensity range. We include an additional range dimension a into our 4D bilateral grid. Instead of the homogeneous values in the original bilateral grid, only intensity values are stored in our new grid.

In general, our new 4D bilateral grid is used in three steps. First, we create a 4D grid from the input image. Then, the data inside the grid are weighted and smoothed. Finally, the weighted median values are computed by grid slicing and accumulating.

3.1. Grid Creation

Given an input image I and an edge image E normalized to $[0, 1]$, the image width W, the image height H, the sampling rate of x and y axes s_s, the sampling rate of z axis s_r and the sampling rate of a axis s_a, the 4D bilateral grid contains $(W \times H)/(s_s^2 \times s_r \times s_a)$ grid cells. We construct Γ as follows:

1. For all grid cells (x, y, z, a), we perform the initialization:

$$\Gamma(x, y, z, a) = 0. \tag{4}$$

2. For each pixel in the image I at position (x, y), we perform the accumulation:

$$\Gamma([\frac{x}{s_s}], [\frac{y}{s_s}], [\frac{E(x,y)}{s_r}], [\frac{I(x,y)}{s_a}]) += 1, \tag{5}$$

where $[\cdot]$ denotes the closest-integer operator. Note that we only count the number of image pixels corresponding to each grid cell during the construction of our 4D bilateral grid. This is different from Chen et al.'s original bilateral grid [2] which accumulate both the image intensity and the number of pixels using homogeneous coordinates. Since each image pixel is scattered to the 4D grid once, the image data are sparsely coded in the grid.

3.2. Grid Smoothing

The grid smoothing is performed to compute aggregated weights in each a level. Ma et al. [13] employed the guided filter [4] to smooth each slice of their 3D histogram volume while Yang et al. [26] suggested to employ constant time bilateral filters in their method. Benefiting from the edge-aware bilateral grid, a separable Gaussian smoothing is applied to the x, y and z axes of the grid respectively. The a axis remains unfiltered. It is well known that the tails of the Gaussian kernel fall of quickly. In order to obtain the efficiency gain, the truncated Gaussian convolution is employed to produce an approximate result [15]. Since neighboring pixels with quite different edge values are scattered to the grid cells far away, each cell will not take into consideration remote cells when performing the local Gaussian smoothing. Therefore, the separable Gaussian smoothing on the grid can effectively smooth out noisy signals without smoothing across meaningful edges.

3.3. Grid Slicing

We illustrate the process of median computing in Fig. 1. The finding of the median value for each image pixel (x, y) requires the accumulation of partial sums of weights. The partial sum of weights for each pixel is calculated by adding up each slice based on the a level:

$$S(x,y,a) = \begin{cases} \Gamma(\frac{x}{s_s}, \frac{y}{s_s}, \frac{E(x,y)}{s_r}, a), & \text{if } a = 0; \\ S(x,y,a-1) + \Gamma(\frac{x}{s_s}, \frac{y}{s_s}, \frac{E(x,y)}{s_r}, a), & \text{else.} \end{cases} \tag{6}$$

Let $A = [1/s_a]$ be the number of a levels, then the value of a ranges from 0 to $A - 1$ with a total of A integral numbers.

Figure 2: Various smoothing effects with increasing sampling rates of s_a (left to right): input, $s_a = 1/256$, $s_a = 1/128$, $s_a = 1/64$, $s_a = 1/32$, $s_a = 1/16$, $s_a = 1/8$. We set $s_s = 16$, $s_r = 1/16$ for all images. The input image was downloaded from `http://mrl.nyu.edu/projects/image-analogies`.

Consequently, the partial sum $S(x, y, A-1)$ with the maximum a level stores the total sum of weights. We can find the weighted median intensity $\overline{a}$ from the total sum: $S(x, y, \overline{a}) = S(x, y, A-1)/2$. The values of x/s_s, y/s_s, $E(x, y)/s_r$ and $\overline{a}$ are usually not integral numbers and thus the slicing and interpolation in the bilateral grid are required.

Our 4D bilateral grid can be viewed as a sequence of 3D bilateral grids based on a levels. Given the 4D bilateral grid Γ and the edge image E, we slice the grid at $(x/s_s, y/s_s, E(x, y)/s_r, a)$ for each level a. The grid slicing operation is actually a simple trilinear interpolation which is implemented with a linear interpolation of two bilinear interpolations from the 8 neighboring grid cells. Note that each cell of the grid only stores a scalar value for a grayscale image. Therefore, our slicing does not need to divide Γ with homogeneous values which is required in Chen et al.'s bilateral grid [2]. Since we cannot find the exact value of $\overline{a}$, we estimate it by slicing S with a linear interpolation followed by a scaling:

$$M(x, y) = s_a \cdot \left(\lfloor \overline{a} \rfloor + \frac{S(x, y, \overline{a}) - S(x, y, \lfloor \overline{a} \rfloor)}{S(x, y, \lfloor \overline{a} \rfloor + 1) - S(x, y, \lfloor \overline{a} \rfloor)} \right), \quad (7)$$

where $\lfloor x \rfloor$ denotes the largest integer not greater than x. For an 8-bit digital image, the maximum value of A is 256 and the minimum sampling rate of a is $s_a = 1/256$. If we use the minimum sampling rate of a, no interpolation is required and the accessed result is the exact weighted median value. If a coarser sampling rate of a is used, the linear interpolation will produce approximate median values. Note that higher order interpolations [2, 26] can be used to produce a better fitting result but may need more computations.

3.4. GPU Implementation

In this section, we present the implementation details on the GPU parallelization which enables a real-time weighted median filtering.

Chen et al. [2] have implemented their bilateral grid using the vertex shader and the pixel shader in the graphics rendering pipeline. Currently, NVIDIA CUDA [14] provides highly parallel supports for arbitrary data scattering and data collection using GPUs. CUDA C is a standard C programming language with some ornamentations to allow the programmer to specify

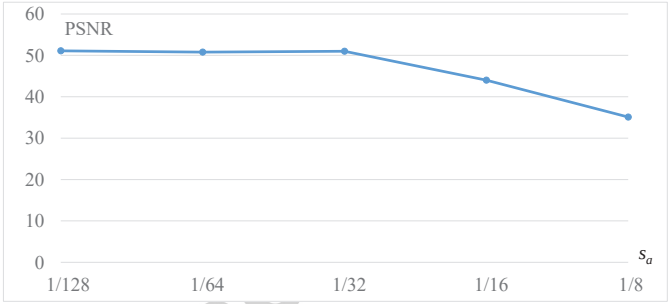


Figure 3: PSNR statistics of Fig. 2 as compared with the result of $s_a = 1/256$.

Figure 4: Different smoothing effects with increasing s_r: input, $s_r = 1/16$, $s_r = 1/4$. We set $s_s = 16$, $s_a = 1/64$ for the images. The input image was obtained from samples of [14].

which code should run on the GPU. In our work, we have implemented the proposed weighted median filter using the easy-to-use yet powerful CUDA.

The 4D bilateral grid is stored as a 2D texture by tiling both of z levels and a levels across the texture plane. Since our bilateral grid does not contain homogeneous values, we only use a 32-bit floating point for each grid cell. Therefore, a grid in general requires $(4 \times W \times H)/(s_s^2 \times s_r \times s_a)$ bytes of video memory in this format. For the values of $s_s = 16$, $s_r = 1/16$ and $s_a = 1/16$, our 4D bilateral grid requires about 4 megabyte of video memory per megapixel. If a moderate GPU do not have enough video memory to process large images, one can store the data only for each active a level in the video memory and use the main memory as the out-of-core buffer.

For the grid creation, we launch CUDA *threads* based on the numbers of x and y levels of the grid. As a consequence, we can take advantage of the CUDA *shared memory* for the collection of in-cell pixel numbers. The efficient *atomic addition* on the shared memory is used for accumulating the number of pixels into each grid cell.

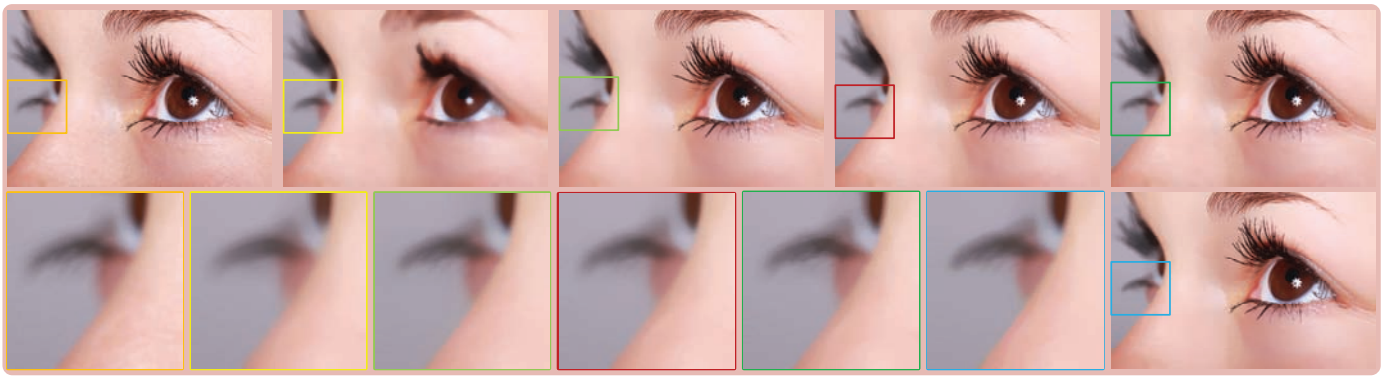


Figure 5: The comparison of smoothing with different filters (left to right): input, the standard median filter, Chen et al's bilateral grid filter, the Ma et al.'s weighted median filter, Zhang et al.'s 100+ times faster weighted median filter, and (the rightmost in the bottom row) our weighted median filter. The filtering effects on the low-contrast nose are zoomed in for better visualization. The input image was downloaded from `http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds`.

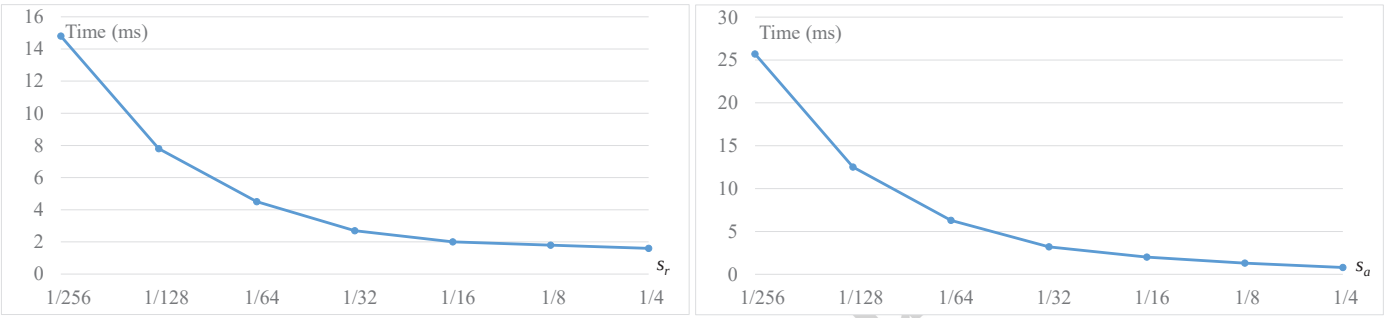


Figure 6: Statistics of running times for a 1-megapixel grayscale image with (left) different values of s_s and (right) different values of s_a. We set $s_s = 16$, $s_r = 1/16$ and $s_a = 1/16$ by default.

During the grid smoothing, the sampling rates s_s and s_r correspond to the variances of the Gaussian kernels as suggested in [15]. The separable Gaussian filter is applied to x, y and z axes respectively with a truncated 5-tap 1D Gaussian kernel. Again the shared memory is used to avoid duplicated loading of the grid data among different threads in each CUDA *block*. Since the maximum number of overlapping cells is 5 for the 5-tap Gaussian kernel, the shared memory can save the loading time efficiently.

Finally, the weighted median for each pixel is computed with two loops. The first loop accumulates the partial sums S for each a level. For the trilinear interpolation used in computing S, we perform two bilinear interpolations for the two nearest z levels and linear interpolate between them. The second loop finds the two consecutive a levels based on the half of the total sum of S and then linear interpolate between them to output the median intensity. As a consequence, the computed median intensity is the weighted median filtering result.

There are two schemes to filter a color image. We can first transform the input RGB color space into CIE-Lab color space and perform the proposed weighted median filtering on the luminance channel and then transform it back to RGB color space. We can also filter each RGB color channel separately. The second scheme can produce clearer color boundaries in some cases where two adjacent regions with similar intensities contain quite different colors.

4. Results and Discussions

We have implemented our novel weighted median filter using highly parallel NVIDIA CUDA kernels. In order to verify the efficiency of the proposed algorithm, we tested it on a PC with a 3.20GHz Intel Core i5-3470 CPU (8 GB main memory) and an NVIDIA GeForce GTX Titan X GPU (3072 streaming processors and 12 GB graphics memory).

Fig. 2 show a various smoothing effects with gradually increasing sampling rates of s_a. An 8-bit luminance channel contains maximum 256 intensities and thus the minimum sampling rate is $1/256$. With the minimum sampling rate, we obtain the exact median result as shown in the second image from left of Fig. 2. With the gradually coarser sampling rate, linear interpolation is used to approximate the median value. We give the corresponding quantified PSNR (peek signal-to-noise ratio) values for these approximate results as compared with the exact one in Fig. 3. The PSNR values are over 40dB with $s_a \leq 1/16$ which means that the approximate results are quite similar to the exact one. From Fig. 3, we can see that the approximation error is gradually increased with the increase of sampling rate. For the rightmost image, unwanted quantization artifacts are visible but the PSNR value is still over 35dB. Therefore, even with $s_a = 1/8$ the approximate result can still be tolerable in some applications.

In order to verify the capability of edge-preservation corresponding to s_r, we show different smoothing effects with differ-

Table 1: Performance comparison of our method with Zhang et al.'s 100+ times faster WMF [31] and Ma et al's constant time WMF [13]. The spatial kernel size $s_s = 16$ and the range sampling rate $s_r = 1/16$ for all methods. The C++ code of 100+ times faster WMF is provided by the authors while the constant time WMF with guided aggregation [4] is coded using CUDA by ourselves.

Method	Image size (in pixel)						
	512×512	768×768	1024×1024	1280×1280	1536×1536	1792×1792	2048×2048
100+ times faster WMF (CPU, 1 channel)	189ms	377ms	730ms	950ms	1401ms	1773ms	2517ms
Constant time WMF (GPU, 1 channel)	286ms	519ms	741ms	1037ms	1361ms	1727ms	2122ms
Our WMF (GPU, 1 channel, $s_a = 1/256$)	12.4ms	19.0ms	28.3ms	39.8ms	54.4ms	70.7ms	94.9ms
Our WMF (GPU, 1 channel, $s_a = 1/16$)	0.9ms	1.4ms	2.0ms	2.7ms	3.8ms	4.6ms	5.6ms
Our WMF (GPU, 3 channels, $s_a = 1/16$)	2.1ms	3.6ms	5.7ms	7.8ms	10.6ms	13.2ms	16.3ms
Our WMF (CPU, 1 channel, $s_a = 1/16$)	414ms	945ms	1664ms	2598ms	3745ms	5125ms	6662ms

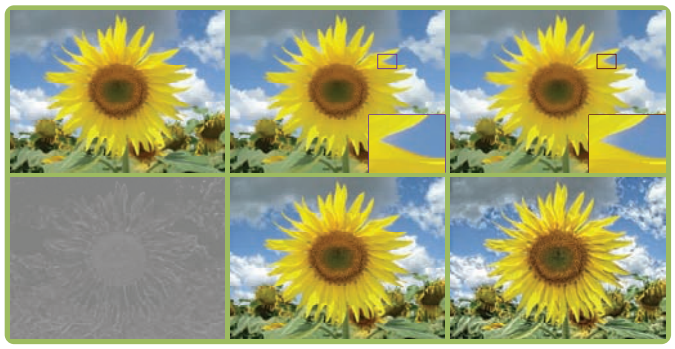


Figure 7: (Top) input, base layer with RGB separate filtering, and base layer with luminance-channel filtering; (bottom) detail layer for the top-right base layer and corresponding progressive levels of detail enhancement. The input image was downloaded from `http://kaiminghe.com`.

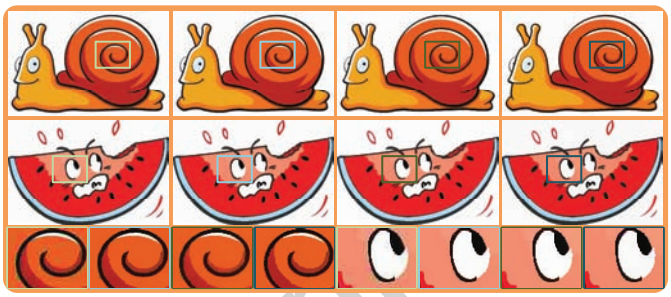


Figure 8: Comparison of removal of JPEG Cartoon artifacts (from left to right): input, 100+ times faster WMF [31], constant time WMF [13], and our WMF. The selected regions are zoomed in for better illustration. We can see that all filters produce comparable artifact removal results. The input images were downloaded from `http://kaiminghe.com`.

Figure 9: An example of inverse halftoning (from left to right): input image ©Hergé, our result, and the result of [8] provided by the authors.

ent sampling rates of s_r in Fig. 4. More details are preserved with smaller s_r, whereas more edges are smoothed out with large s_r. When the value of s_r is too large, some meaningful edges are also smoothed.

Fig. 5 shows the comparison of our weighted median filter with the standard median filter, Chen et al's bilateral grid filter [2], Ma et al.'s weighted median filter [13] and Zhang et al.'s 100+ times faster weighted median filter [31]. All the methods use the same filtering width $s_s = 16$ and the bilateral grid filter and our filter use the same range sampling rate $s_r = 1/16$. In addition, we use $s_a = 1/64$ in this test. Ma et al.'s weighted median filter employs the guided image filter [4] for computing the edge-preserving weights in this comparison. The standard median filter cannot preserve edges well while the bilateral grid filter fail to preserve the edge on the nose. See the different filtering effects in the low-contrast nose in Fig. 5, the nose boundary is well preserved with our method.

In order to test the performance impacts of range sampling rates, we give the statistics of running times in Fig. 6 for a 1-megapixel grayscale image with different values of the parameters s_r and s_a. The values of s_r and s_a control the number of values in the two range dimensions of the 4D grid, respectively. From the figure we can see that the performance gradually increases with the increase of the two parameters. It should be noted that larger values of s_a will result in approximate median values as illustrated in Fig. 3.

In Table 1 we show the performance comparison of our method with Zhang et al.'s 100+ times faster WMF [31] and Ma et al's constant time WMF [13]. The C++ code of 100+ times faster WMF is provided by the authors while the constant time WMF with guided aggregation [4] is coded using CUDA by ourselves. If we set constant values for all parameters (i.e., s_s, s_r and s_a), the performance gradually decreases with respect of the image size. As shown in the table, our algorithm with $s_a = 1/16$ only requires 5.6ms even for a high-resolution 8-bit image with 2048×2048 pixels. Real-time performance can be achieved for high-resolution images with our new WMF. Our GPU code is substantially optimized and the times are much better than the times reported in our preliminary conference version [32]. In comparison, Zhang et al.'s 100+ times faster WMF [31] and Ma et al.'s constant time WMF [13] can only achieve near real-time performance for low-resolution images. In addition to the CUDA implementation on the GPU, we have also implemented our method using C++ on the CPU. From Table 1 we can see that our WMF running on the GPU outperforms our WMF running on the CPU by 2∼3 orders of magnitude speedup with our experimental environment. Note that the 100+ times

Figure 10: Vivid image stylization effects based on our weighted median filter: (left) input images, (right) stylized images. The input desert image was downloaded from `http://www.cse.cuhk.edu.hk/~leojia/projects/L0smoothing`. The two input flower images were downloaded from `http://maverick.inria.fr/Publications/2007/OBBT07`. The three input portrait images were downloaded from `http://www.cs.umsl.edu/~kang`.

Figure 11: Photographic look transfer (left to right): the model image, input images and the corresponding transferred images. The two input hill images were downloaded from `http://people.csail.mit.edu/soonmin/photolook`. The input shark image was downloaded from `http://groups.csail.mit.edu/graphics/bilagrid`. Compared to [2]

faster WMF cannot be parallelized on the GPU directly. As a consequence, the constant time WMF is faster than the 100+ times faster WMF for high-resolution images on our test PC because of the parallelization power of the GPU. Yang et al. [26] did not show the performance of their WMF in their paper and the CPU performance of 83ms per megapixel was reported for their median filter with the intensity downsampling rate of 16. In summary, our algorithm outperforms these state-of-the-art WMF algorithms in terms of speed.

Grid smoothing and slicing have a major drawback of not suitable for handling many values. The 3 color channels may introduce millions of different values regarding their different combinations, let alone the modification to extend the method to 4D with even more segments for separation. Therefore, we reduce to hundreds of values with the channel separation scheme to improve the performance. We have recorded the running times for 3 color channels in Table 1. We can see that the filtering on grayscale images is roughly 3 times as fast as the filtering on color images with the same values of parameters. The slight difference between the RGB separate filtering and the luminance-channel filtering is illustrated in Fig. 7 (top).

5. Edge-Aware Applications

We have implemented a variety of edge-aware image processing applications based on our new filter, including detail enhancement, JPEG artifacts removal, inverse halftoning, image stylization, photographic look transfer and image colorization.

Detail enhancement is very important for better visualization of image details. As shown in Fig. 7, the sunflower is decomposed into a base layer and a detail layer. The base layer contains large structural information while the detail layer has small detail information. The base layer is produced by smoothing the input image with our edge-aware filter and the detail layer is the difference between the base layer and the input image. As a result, the details of the sunflower are progressively boosted as in Fig. 7 (bottom).

JPEG artifacts removal aims to smooth out unwanted artifacts near edges produced after the frequency quantization of JPEG compression. The JPEG format is a frequently used image compression format to reduce the storage consumption. It is required to remove JPEG artifacts. We simply filter the JPEG image using our edge-aware filter. Fig. 8 compares our artifact removal results with the results of 100+ times faster WMF [31] and constant time WMF [13]. As show in Fig. 8, all filters can

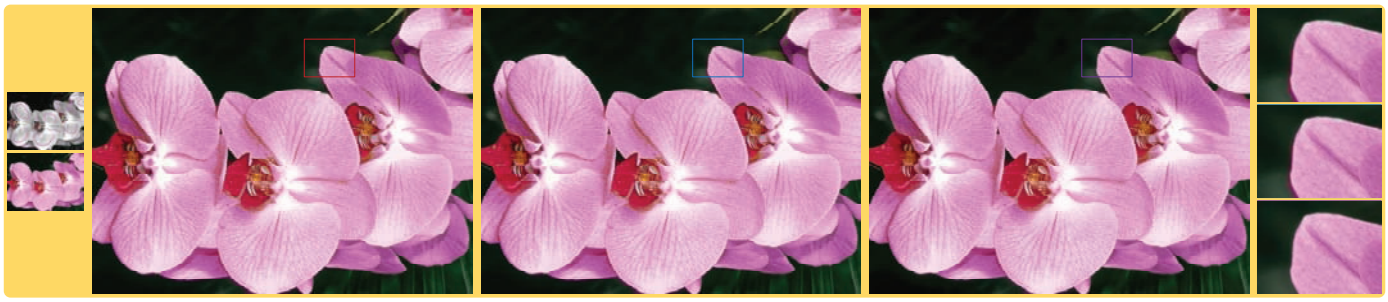


Figure 12: Image colorization for a 1600×1200 image (left to right): subsampled 320×240 grayscale image and corresponding colorized image, nearest neighbor upsampling, bilinear upsampling, and our upsampling. Detail insets are given on the rightmost and our result avoids the spill of colors over edges. The input image was downloaded from `http://www.weimeikeai.com/weimeizhuomian`.

effectively smooth out compression artifacts while preserving meaningful edges.

Inverse halftoning refers to remove background dots from a halfton image while preserving the main information of the image. Background stipple dots can be viewed as noisy details and we remove these dots with our new filter. We show different reconstruction results in Fig. 9 for our method and an excellent inverse halftoning method [8]. The experiment shows that our simple method can also produce an acceptable inverse halftoning effect.

Image stylization produces cartoon-like images from photographs with less amount of colors and dark edge lines. We first perform our edge-preserving smoothing filter to generate a piecewise smoothing image and then draw dark lines onto salient edges. The coherent lines are rendered based on the coherent local structure tensor [9, 10] in this paper. Soft luminance quantization is employed to produce cartoon-like effects [22, 34]. As shown in Fig. 10, our approach is able to convert photographs to vivid stylized results.

Photographic look transfer tries to transfer the photographic style of a model image to an input image [1]. In this paper, we follow the pipeline of Chen et al. [2]. First, decompose the image into two layers using our edge-aware filter. Second, perform the histogram transfer for these two layers separately. Finally, compose the two transferred layers into a final photographic transferred image. From Fig. 11, we can see that our method effectively transfers the photographic look from a model photograph.

Image colorization is a computer-assisted technique which propagate user-specified colors to the whole grayscale image. Many colorization algorithms [11, 33] require to solve large linear systems which are usually time consuming. Kopf et al. proposed to colorize a downsampled image [11] and then upsample it with a joint bilateral upsampling method [7]. In our approach, we first colorize a downsampled image using a nonlocal optimization method [33]. Then a high-resolution image is obtained with a simple bilinear upsampling. Finally, our edge-aware filter is applied to the upsampled color image with the guidance of the input high-resolution grayscale image. Note that the lightness distribution of the output color image remains the same as the input grayscale image and only the two chrominance channels are updated. Fig. 12 shows our colorization result. We can see that our approach effectively avoids the spill of colors over edges in the image.

6. Conclusions and Future Work

In this paper, we have presented a novel real-time weighted median filtering algorithm by substantially exploit the parallelism of modern GPUs. A new edge-aware 4D bilateral grid is defined by elevating the computation domain of the traditional 3D bilateral grid into an even higher dimension. The use of 4D bilateral grid makes the computation complexity of weighted median filtering be less relevant to the size of input image but depend on the coarsely sampled bilateral grid. Experimental results have shown that our new algorithm can be run efficiently in real-time. In addition, the edge-aware applications have demonstrated the feasibility of the proposed algorithm.

When small sampling rates are used to perform the filtering operation, large memory and computational costs are required because of the large resolution of the bilateral grid. In the future, we would like to reduce the required memory by exploring a compact representation of the bilateral grid. Another future direction is to exploit more edge-aware image processing and computer graphics applications related to the weighted median filtering. As studied by Ma et al. [13], the weighted median filtering can be used for disparity refinement in local stereo matching as well as the depth image upsampling. We plan to incorporate our filter into these applications in the near future.

Acknowledgements

We would like our anonymous reviewers for their helpful comments which definitely improve this paper. This work was supported by the Zhejiang Provincial Natural Science Foundation of China (Grant No. LY15F020019), the National Natural Science Foundation of China (Grant No. 61332017), the Open Project Program of the State Key Lab of CAD&CG, Zhejiang University (Grant No. A1610), and the Science and Technology Planning Project of Wenzhou, China (Grant No. G20150019). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GeForce GTX Titan X GPU used for this research.

References

[1] S. Bae, S. Paris, F. Durand. Two-scale tone management for photographic look. ACM Transactions on Graphics, 2006, 25(3), pp. 637-645.

[2] J. Chen, S. Paris, F. Durand. Real-time edge-aware image processing with the bilateral grid. ACM Transactions on Graphics, 2007, 26(3), article no. 103.

[3] D. Cline, K. B. White, P. K. Egbert. Fast 8-bit median filtering based on separability. In Proceedings of IEEE International Conference on Image Processing, Washington, IEEE Computer Society, 2007, 5, pp. 281-284.

[4] K. He, J. Sun, X. Tang. Guided image filtering. In Proceedings of European Conference on Computer Vision, Berlin, Springer-Verlag, 2010, pp. 1-14.

[5] T. Huang, G. Yang, G. Tang. A fast two-dimensional median filtering algorithm. IEEE Transactions on Acoustics, Speech and Signal Processing, 1979, 27(1), pp. 13-18.

[6] M. Kass, J. Solomon. Smoothed local histogram filters. ACM Transactions on Graphics, 2010, 29(4), article no. 100.

[7] J. Kopf, M.F. Cohen, D. Lischinski, M. Uyttendaele. Joint bilaterial upsampling. ACM Transactions on Graphics, 2007, 26(3): 96:1-96:5.

[8] J. Kopf, D. Lischinski. Digital reconstruction of halftoned color comics. ACM Transactions on Graphics, 2012, 31(6): 140:1-140:10.

[9] J.E. Kyprianidis, J. Döllner. Image abstraction by structure adaptive filtering. In Proceedings of EG UK Theory and Practice of Computer Graphics, 2008, pp. 51-58.

[10] J.E. Kyprianidis, H. Kang. Image and video abstraction by coherence-enhancing filtering. Computer Graphics Forum, 2011, 30(2), pp. 593-602.

[11] A. Levin, D. Lischinski, Y. Weiss. Colorization using optimization. ACM Transactions on Graphics, 2004, 23(3): 689-694.

[12] X.-Y. Li, Y. Gu, S.-M. Hu, R. R. Martin. Mixed-domain edge-aware image manipulation. IEEE Transactions on Image Processing, 2013, 22(5): 1915-1925.

[13] Z. Ma, K. He, Y. Wei, J. Sun, E. Wu. Constant time weighted median filtering for stereo matching and beyond. In Proceedings of IEEE International Conference on Computer Vision, Washington, IEEE Computer Society, 2013, pp. 49-56.

[14] NVIDIA Corporation. https://developer.nvidia.com/cuda-toolkit. CUDA programming guide. 2015, version 7.5.

[15] S. Paris, F. Durand. A fast approximation of the bilateral filter using a signal processing approach. International Journal of Computer Vision, 2009, 81, pp. 24-52.

[16] S. Perreault, P. Hébert. Median filtering in constant time. IEEE Transactions on Image Processing, 2007, 16, pp. 2389-2394.

[17] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, Washington, IEEE Computer Society, 2011, pp. 504-511.

[18] C. Tomasi, R. Manduchi. Bilateral filtering for gray and color images. In Proceedings of International Conference on Computer Vision, Washington, IEEE Computer Society, 1998, pp. 839-846.

[19] J. W. Tukey. Exploratory data analysis. Reading, MA. Addison-Wesley, 1977.

[20] H. Wang, J. Cao, X. Liu, J. Wang, T. Fan, J. Hu. Least-squares images for edge-preserving smoothing. Computational Visual Media, 2015, 1(1): 27-35.

[21] B. Weiss. Fast median and bilateral filtering. ACM Transactions on Graphics, 2006, 25, pp. 519-526.

[22] H. Winnemöller, S.C. Olsen, B. Gooch. Real-time video abstraction. ACM Transactions on Graphics, 2006, 25(3): 1221-1226.

[23] L. Xu, C. Lu, Y. Xu, J. Jia. Image smoothing via L0 gradient minimization. ACM Transactions on Graphics, 2011, 30(6): 174:1-174:12.

[24] L. Xu, Q. Yan, Y. Xia, J. Jia. Structure extraction from texture via relative total variation. ACM Transactions on Graphics, 2012, 31(6): 139:1-139:10.

[25] Q. Yang. Recursive bilateral filtering. In Proceedings of European Conference on Computer Vision, Berlin, Springer-Verlag, 2012, pp. 399-413.

[26] Q. Yang, N. Ahuja, K.H. Tan. Costant time median and bilateral filtering. International Journal of Computer Vision, 2015, 112, pp. 307-318.

[27] L. Yin, R. Yang, M. Gabbouj, Y. Neuvo. Weighted median filters: A tutorial. IEEE Transactions on Circuits Systems-II: Analog and Digital Signal Processing, 1996, 43(3), pp. 157-192.

[28] Y. Zang, H. Huang, L. Zhang. Efficient structure-aware image smoothing by local extrema on space-filling curve. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(9): 1253-1265.

[29] Y. Zang, H. Huang, L. Zhang. Guided adaptive image smoothing via directional anisotropic structure measurement. IEEE Transactions on Visualization and Computer Graphics, 2015, 21(9): 1015-1027.

[30] S.-H. Zhang, X.-Y. Li, S.-M. Hu, R. R. Martin. Online video stream abstraction and stylization. IEEE Transactions on Multimedia, 2011, 13(6): 1286-1294.

[31] Q. Zhang, L. Xu, J. Jia. 100+ times faster weighted median filter (wmf). In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Washington, IEEE Computer Society, 2014, pp. 2830-2837.

[32] H. Zhao, D. Gao, M. Wang, Z. Pan. Real-time weighted median filtering with the edge-aware 4d bilateral grid. In Proceedings of the 10th International Conference on E-Learning and Games (Edutainment '16), 2016, Berlin, Springer, LNCS 9654, pp. 125-135.

[33] H. Zhao, G. Nie, X. Li, X. Jin, Z. Pan. Structure-aware nonlocal optimization framework for image colorization. Journal of Computer Science and Technology, 2015, 30(3): 478-488.

[34] H. Zhao, X. Jin, J. Shen, X. Mao, J. Feng. Real-time feature-aware video abstraction. The Visual Computer, 2008, 24(7-9), pp. 727-734.