Image Anal Stereol 2015;34:101-110
Original Research Paper

doi: 10.5566/ias.1225

FLOWING BILATERAL FILTER: DEFINITION AND IMPLEMENTATIONS

MAXIME MOREAUD✉ AND FRANCOIS COKELAER
IFP Energies nouvelles, B.P. 3, 69390 Solaize, France
e-mail: maxime.moreaud@ifpen.fr; francois.cokelaer@ifpen.fr

(Received September 22, 2014; revised April 22, 2015; accepted April 27, 2015)

ABSTRACT

The bilateral filter plays a key role in image processing applications due to its intuitive parameterization and its high quality filter result, smoothing homogeneous regions while preserving the edges of the objects. Considering the image as a topological relief, seeing pixel intensities as peaks and valleys, we introduce a way to control the tonal weighting coefficients, the flowing bilateral filter, reducing "halo" artifacts typically produced by the regular bilateral filter around a large peak surrounded by two valleys of lower values. In this paper we propose to investigate exact and approximated versions of CPU and parallel GPU (Graphical Processing Unit) based implementations of the regular and flowing bilateral filter using the NVidia CUDA API. Fast implementations of these filters are important for the processing of large 3D volumes up to several GB acquired by x-ray or electron tomography.

Keywords: adaptive filter, GPU, 3D image processing.

INTRODUCTION

Filtering operation is a critical image processing operation which performs noise attenuation allowing to do further advanced tasks in better conditions (*e.g.*, segmentation or analysis, …). The bilateral filter formulated by Tomasi and Manduchi (1998), whose beginnings can be found in Smith and Brady (1997), belongs to the class of non-iterative and locally adaptive image filters (Barash, 2002). Besides regular spatial weighting (directly linked to the distance from the center of the filter's observation window), the main idea of the bilateral filter is to insert an additional weighting factor based on a tone (or photometric, grey level) distance. Thus, a spatially close pixel, but far in terms of tone value, would have a low contribution in a result value.

The research community devoted large efforts during the past decade to reduce the bilateral filter complexity induced by the computation of the tonal weighting factor, mainly by using non-exact versions of this filter. Straightforwardly, we can consider, as for the Gaussian filter, that a k-D kernel can be decomposed in a succession of k 1-D kernels, each result of a 1-D filter being the next filter's input image (Pham and Vliet, 2005). Note that, unlike the Gaussian filter, the result of this filter is only an approximation of the bilateral filter (the same remark could be done for all non-separable filters, such as median filter approximation (Narendra, 1981)). Later, we will refer to this filter as the separable bilateral filter. Paris and Durand (2006) proposed an original formulation of the bilateral filter seeing it as a convolution in a spatio-tonal space of dimension k+1. This method is composed of three steps. Firstly, conversion of the image in the space of higher order dimension. Secondly, filtering using standard convolution in this space. Thirdly, conversion by interpolation of the result in the initial space. Note that a specific data structure, the "bilateral grid" was recently developed (Chen *et al.*, 2007) combining the latter method with a GPU compliant architecture. Weiss (2006) proposed a O(log(r)) per pixel implementation (with 'r' corresponding to the spatial width of the filter) but limited to constant weighting functions (box-filter). His approach uses sliding windows histogram computation, also known as an efficient way to speed up median filters (Huang, 1981). Recently Porikli, (2008) and Yang *et al.*, (2009), improved this method by using image integral histogram and succeeded in lowering the complexity to O(1) per pixel in the case of constant or polynomial weighting functions (Taylor development of Gaussian kernel). Note that these methods are efficient for 8 bit images but remain computationally intensive for 24 or 32 bits images leading to a huge memory requirement for large images due to local histogram computation. Recently, Chaudhury *et al.*, (2011) used the O(1) algorithm of Porikli, (2008) but with trigonometric range kernels thus yielding a better approximation.

A lot of efforts have been brought to speed up the bilateral filter at algorithmic levels, sometimes by using an elaborated data structure or a rough version of the filter. But, recent advances of massively parallel computation hardware opens new perspectives: especially for processing of large 3D data volumes, up to several GigaBytes.

Despite of the large number of papers on the acceleration of the 2D bilateral filter on GPU, only a few recent works can be found on the 3D implementation. Bethel, (2012) proposed an NVidia CUDA based 3D exact implementation of the bilateral filter together with a study of the impact of the GPU configuration parameters. However, these tests are for a fixed and relatively small 3D image (256x256x120 voxels). In Banterle *et al.*, (2012), an approximated version of the bilateral filter is proposed by using subsampling, benefiting from GPU fast cache texture fetches. This implementation gives a good trade-off between computation time and quality of the filtering result.

In this paper, our contributions are :

- Seeing images as peaks and valleys (Salembier and Serra, 1994; Serra and Soille, 1994**)**, we derive a new bilateral filter formulation: the flowing bilateral filter. Additionally to the well-known spatial and tonal attenuation coefficient, a topological approach of the image allows suppression of "halo[1]" artifacts around a large peak surrounded by two valleys of different values.
- Study of several bilateral filter implementations, *i.e.*, regular (as in (Tomasi and Manduchi, 1998) but with Tukey's biweight function (Durand and Dorsey, 2002)) and separable versions are proposed on massively parallel architecture using the NVidia CUDA API considering the processing of quite large 3D volumes acquired by X-ray or electron tomography.

METHODS

We first recall the definition of the regular bilateral filter. Then, we give the exact definition of the flowing bilateral filter. Finally, the definition of the separable version of the flowing bilateral filter is given.

REGULAR BILATERAL FILTER

Let *I* be an image defined on its spatial domain *D*. Let *f* and *g* be two even functions having their maximum in $x = 0$, decreasing from $x = 0$ and parameterized by σ_f and σ_g for *f* and *g* respectively. *f* and *g* are typically Gaussian but can take other forms like the fast decreasing and truncated Tukey's biweight function ($f(x) = g(x) = 0.5(1-(x/\sigma)^2)^2$ if $|x| < \sigma$, 0 otherwise). For a pixel location *p* in *D*, the result of the bilateral filter is given by $I'(p)$ (Eq. 1):

$$\begin{aligned} I'(p) &= \frac{1}{W}\sum_{p' \in D} f_{\sigma_f}(\|p-p'\|)\, g_{\sigma_g}(|I(p)-I(p')|)\, I(p) \\ \text{with } W &= \sum_{p' \in D} f_{\sigma_f}(\|p-p'\|)\, g_{\sigma_g}(|I(p)-I(p')|) \end{aligned} \quad . \qquad (1)$$

The effect of this filter is illustrated on a 1D profile (Fig. 1). The bilateral filter can be seen as signal convolution with the function *f* weighted by the function *g*. A pixel present in the observation window will, therefore, be strongly taken into account in the convolution points at low distance from the current point (standard convolution) and close in intensity of the current point (action of function *g*), but these two aspects are taken into account independently "*f* does not see the intensity and *g* does not see the distance".

FLOWING BILATERAL FILTER

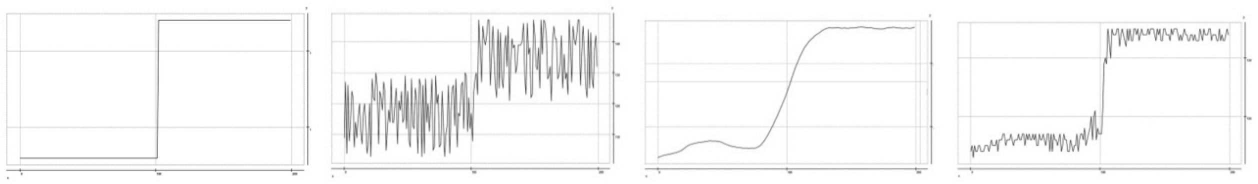
Here, an image is seen as a topological relief. We consider the case where two valleys of different values are surrounding a large peak (*see* Fig. 2 for an illustration). As we have seen before, the bilateral filter, with an adequate parameterization, can keep intact the strong transition. However, in this particular case, we can observe an overflow of the valleys around the peak, the lower value valley into the higher value valley and reciprocally. This case leads to the creation of "halo" in the filtered images and can be avoided if $f \times g$ is strictly decreasing. This kind of function can be designed by imposing the decrease of the function g by adding a comparison while computing filter's tonal weight : if d and d' are distances from the central pixel, with d>d', then g(d) must be smaller than g(d'). This criterion can be formulated as a morphological reconstruction operation (Serra, 1988; Vincent, 1993) in the weighting function space. Grayscale morphological reconstruction $\rho_Y(X)$ of Y from X is obtained by iterating grayscale geodesic dilation of X "under" Y until stability is reached (Vincent, 1993):

$$\rho_Y(X) = \bigvee_{n \geq 1} \delta_Y^{(n)}(X) \text{ with } \delta_Y^{(1)}(X) = (X \oplus B) \wedge Y$$

and

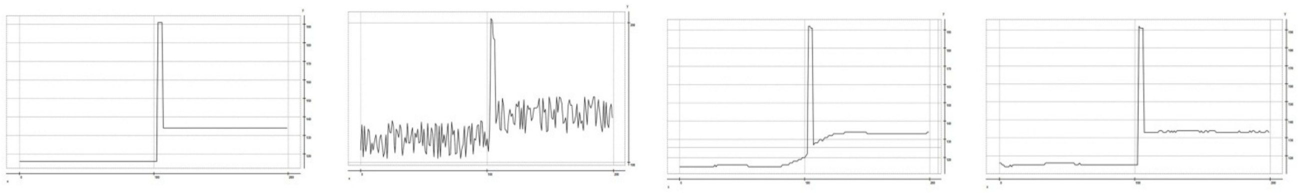
$$\delta_Y^{(n)}(X) = \delta_Y^{(1)} \circ \delta_Y^{(1)} \circ \ldots \circ \delta_Y^{(1)}(X) \text{ (n times).}$$

[1]we mean by "halo" overflows with dark or light intensities not present on the initial image and caused by a filter process.



a. b. c. d.

Fig. 1. *a. Initial 1D profile; b. 1D profile with additional Gaussian noise; c. After application of a spatial filter (Tukey function σ = 15); d After application of a bilateral filter (Tukey functions σ_f = 15 et σ_g = 10).*



a. b. c. d.

Fig. 2. *a. Initial 1D profile composed by a large peak surrounded by two valleys; b. 1D profile with additional Gaussian noise; c. After application of a bilateral filter (f and g, Tukey function, with σf=20 and σg=20); d. After application of the flowing bilateral filter (same parameters).*

This new filter can be written as:

$$I'(p)=\frac{1}{W}\sum_{p'\in D} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I}(p')\,I(p)$$
$$\text{with} \qquad . \tag{2}$$
$$W=\sum_{p'\in D} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I}(p')$$

Within this formulation, the weighted function depends on the location of the central pixel (Eq. 3). This new filter enables the suppression of “halo”, however, morphological opening by reconstruction results in a significant additional time computing overhead especially for 3D processing.

$$\begin{aligned} &G_{\sigma_g,p,I}(p')=\rho_{B_{\sigma_g,p,I}}\left(A_{\sigma_g,p}\right) \text{ with} \\ &A_{\sigma_g,p}(x)=g_{\sigma_g}(0) \text{ if } x=p, \text{ else } 0; \text{ and} \quad . \\ &B_{\sigma_g,p,I}(x)=g_{\sigma_g}\left(\|I(p)-I(x)\|\right) \end{aligned} \tag{3}$$

SEPARABLE FLOWING BILATERAL FILTER

In order to reduce the complexity and memory usage of such a filter, it is possible to design an approximated version by means of a separated kernel (Pham and Vliet, 2005). The separable bilateral filter can be written as Eq. 4:

$$\begin{aligned} &I'(p)=\frac{1}{W_y}\sum_{p'\in Dy(p)} f_{\sigma_f}\left(\|p-p'\|\right)g_{\sigma_g}\left(\|I_x(p)-I_x(p')\|\right)I_x(p), \\ &I_x(p)=\frac{1}{W_x}\sum_{p'\in Dx(p)} f_{\sigma_f}\left(\|p-p'\|\right)g_{\sigma_g}\left(\|I(p)-I(p')\|\right)I(p) \\ &\text{with} \\ &W_x=\sum_{p'\in Dx(p)} f_{\sigma_f}\left(\|p-p'\|\right)g_{\sigma_g}\left(\|I(p)-I(p')\|\right), \quad . \\ &W_y=\sum_{p'\in Dy(p)} f_{\sigma_f}\left(\|p-p'\|\right)g_{\sigma_g}\left(\|I_x(p)-I_x(p')\|\right) \\ &\text{and} \\ &D_x(a)=p\in D\,|\,p_y=a_y \text{ et } D_y(a)=p\in D\,|\,p_x=a_x \end{aligned} \tag{4}$$

The formulation of the separable flowing bilateral filter (*see* an illustration Fig. 4) can be written with a morphological reconstruction as Eq. 5. This formulation can lead to a practical and efficient implementation. Indeed, a morphological reconstruction can be implemented as a simple floating point comparison by a neighborhood element (*see* Algo. 1).

$$\begin{aligned} &I'(p)=\frac{1}{W_y}\sum_{p'\in Dy(p)} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I_x}\left(\|I_x(p)-I_x(p')\|\right)I_x(p), \\ &I_x(p)=\frac{1}{W_x}\sum_{p'\in Dx(p)} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I}\left(\|I(p)-I(p')\|\right)I(p) \\ &\text{with} \\ &W_x=\sum_{p'\in Dx(p)} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I}\left(\|I(p)-I(p')\|\right), \\ &W_y=\sum_{p'\in Dy(p)} f_{\sigma_f}\left(\|p-p'\|\right)G_{\sigma_g,p,I_x}\left(\|I_x(p)-I_x(p')\|\right), \\ &\text{and} \quad . \\ &G_{\sigma_g,p,I}(p')=\rho_{B_{\sigma_g,p,I}}\left(A_{\sigma_g,p}\right) \text{ with} \\ &A_{\sigma_g,p}(x)=g_{\sigma_g}(0) \text{ if } x=p, \text{ else } 0 \text{ and} \\ &B_{\sigma_g,p,I}(x)=g_{\sigma_g}\left(\|I(p)-I(x)\|\right), \\ &D_x(a)=p\in D\,|\,p_y=a_y \text{ et } D_y(a)=p\in D\,|\,p_x=a_x \end{aligned} \tag{5}$$

```
Function Horizontal_flowing_BFilter()
BEGIN
FOREACH p in I
 fRc = 1; // flowing tonal coeff. init
 sum_c = 0.0; // neighborhood coeff. sum
 FOREACH p' in Dx
  Sc = GetSpatialCoeff(p');
  Rc = GetTonalCoeff(p');
  IF( Rc <= fRc) // flowing bilateral filter
  fRc = Rc
  ENDIF
  c = fRc*Sc;
  sum_c += c;
  tot = c*I(p');
 ENDFOREACH
 O(p) = tot/sum;
END FOREACH
```

Algo. 1. *Pseudo code of separable flowing bilateral filter considering the horizontal direction.*

GPU ARCHITECTURE AND CUDA PROGRAMMING MODEL

Even if GPU were originally designed to perform graphic oriented applications, such as renderings and textures mappings, their native parallel architecture (Single Instruction Multiple Data) led the scientific community to bring speed up to highly computational demanding applications. For our study, we used the computed unified device architecture (CUDA) (NVIDIA CUDA C, 2012) developed by NVIDIA for the implementation of regular and separable bilateral filters. In order to operate the native GPU capability to perform parallel work, the NVidia CUDA API enables to run thousands of threads in parallel by launching a batch of threads called *warps* on the GPU's Streaming Multiprocessors (*SMs*) via a function called *kernel function*. A *kernel function* can be seen as a specialized template function on the index of the threads and blocks launched on the *SM*s. GPUs own different types of memory which are on and off chip. The DRam or global memory is the main GPU memory enabling *inter alia*, reading and writing from the host machine (with a high latency and via the PCI Express), and providing memory pointers to *kernel* functions in order to perform data processing. *Shared memory* and *registers* are on chip memories providing a low latency and high bandwidth. Note that *shared memory* is statically allocated by the programmer and that an allocated buffer is shared by all the threads of a block whereas registers are handled automatically by the driver and are related to each of the threads individually.

As we carried over our experiments with an NVidia Quadro 4000 of capability 2.0 we will limit our description to the Fermi architecture. Regarding the relation between GPU architecture and CUDA API, there are three levels of parallelism expressed and they represent three levels of granularity:

- The smaller execution level is the *warp* of threads. In our case the SMs run a batch of 32 threads simultaneously.
- The second level of parallelism is the block, whose size is chosen by the programmer and contains a maximum of 1024 threads for the Fermi architecture. Note that a block can only be executed by one SM however, one SM can execute several blocks.
- The last level of granularity is the grid (more precisely the grid of threads blocks), often determined by the mapping of threads on data desired by the programmer (*e.g.*, it can be convenient to make one thread treating one voxel).

In order to reach a high arithmetic peak for a given applications some basic strategies are recommended (NVIDIA CUDA Best, 2012) and will be experimented in ours implementations:

- An algorithm should be written to exhibit parallelism.
- Minimize data transfer between CPU and GPU because of their penalizing latency.
- Ensure coalesced read from global memory.
- Avoid conditional branching (*e.g.*, "if" statements).
- Maximize SM occupancy rate, *i.e.*, give the multiprocessors a large number of blocks to process.

GPU CODE OPTIMIZATION

Spatial coefficient

The first optimization realized for all the implementations is the pre-computation of the spatial attenuation coefficients on the CPU. As these coefficients don't change during program execution, we load them on the GPU in an on-chip buffer memory of 64kB called *constant memory* which is accessible by all the threads with high bandwidth and small latency.

Memory acces

Our first experiment was designed to show the impact of non-strided access when working with multi-dimensional arrays allocated as a linear memory block. Even if misaligned global memory fetches issues were resolved since the introduction of Fermi architecture (due to the additional L1 cache of 128 bytes in each SMs), Fig. 3 reveals that column and depth (Y and Z directions) fetches penalize global computation time. The specification of the Quadro 4000 GPU card announces a bandwidth peak of 89.6 Go/sec, this serves as a reference to evaluate the speed of our algorithm. The slowdown observed for Y and Z passes is due to the GPU driver fetching mechanism in the global memory where fetches are performed via 32, 64 or 128 bytes transactions aligned with their size. In the case of a 3D image processing algorithms, we need to consider the three directions and to perform a local scan around each voxel. For the regular bilateral filter, we consider a 3D neighborhood window, thus a convenient memory fetching optimization is to bind image data memory portion to a 3D texture cache in order to speed up spatial locality access.

For the separable bilateral filter, even if we can't avoid the reading and writing of the data from the global memory, we can benefit of this first read to load an on-chip and fast buffer of shared memory, and then to perform memory fetches into this buffer for the neighborhood scan. Note that the memory requirement consists of allocating enough memory for the input and output image on the Dram of the GPU (*e.g.*, this represents 1GByte for a 512x512x512 image).

Performance vs occupancy

This first implementation was designed to maximize *SMs* occupancy, *i.e.*, one thread computes one result image voxel. Starting from this first implementation, we decided to investigate the vectorization capabilities of the GPU by treating simultaneously N_BATCH of rows, columns or data vectors in the depth direction thereby introducing Instruction Level Parallelism (ILP) via loop unrolling directive (Volkov, 2010). Note that vectorization is lowering the number of threads and blocks launched on the *SMs* thus tending to reduce occupancy while rising registers usage. Hence the principle is to allocate and to load N_BATCH.

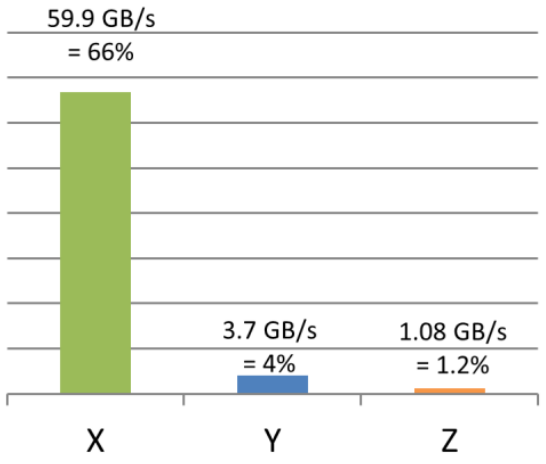


Fig. 3. *Global memory bandwidth ratio with theoretical hardware bandwidth peak considering the three filtering directions (X, Y, Z) through a 512x512x512 – 32 bits data volume and with bilateral filter's fetching pattern.*

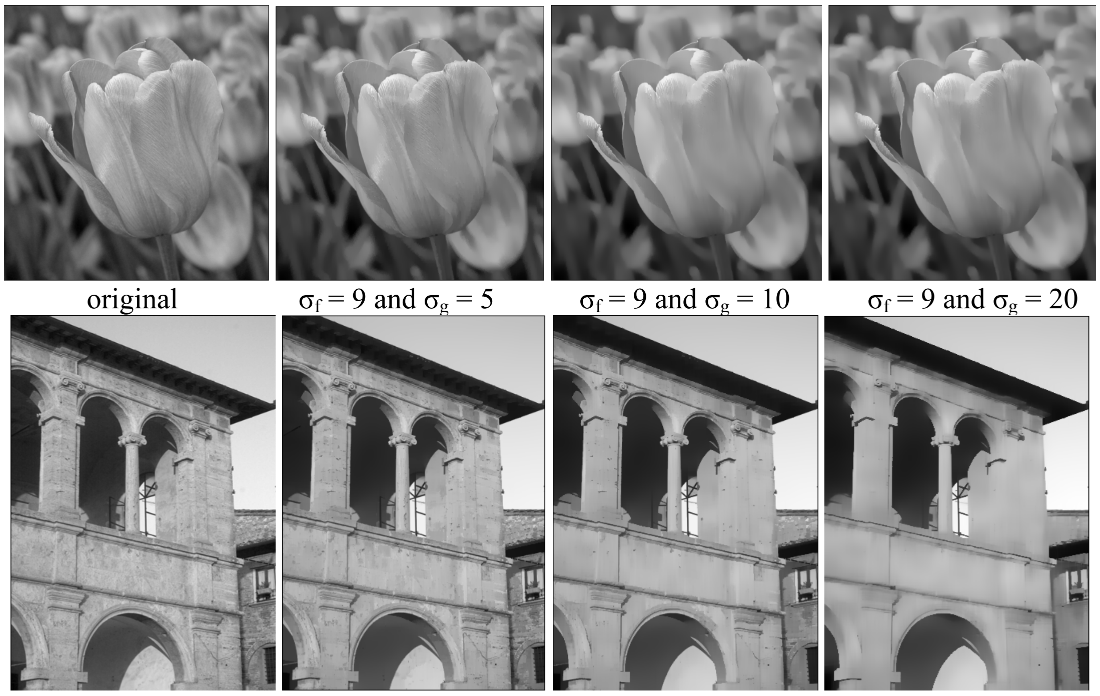

original
$\sigma_f = 9$ and $\sigma_g = 5$
$\sigma_f = 9$ and $\sigma_g = 10$
$\sigma_f = 9$ and $\sigma_g = 20$
original
$\sigma_f = 9$ and $\sigma_g = 5$
$\sigma_f = 9$ and $\sigma_g = 10$
$\sigma_f = 9$ and $\sigma_g = 20$

Fig. 4. *Separable flowing bilateral filter results on standard tests images.*

RESULTS

QUALITATIVE AND QUANTITATIVE ANALYSIS

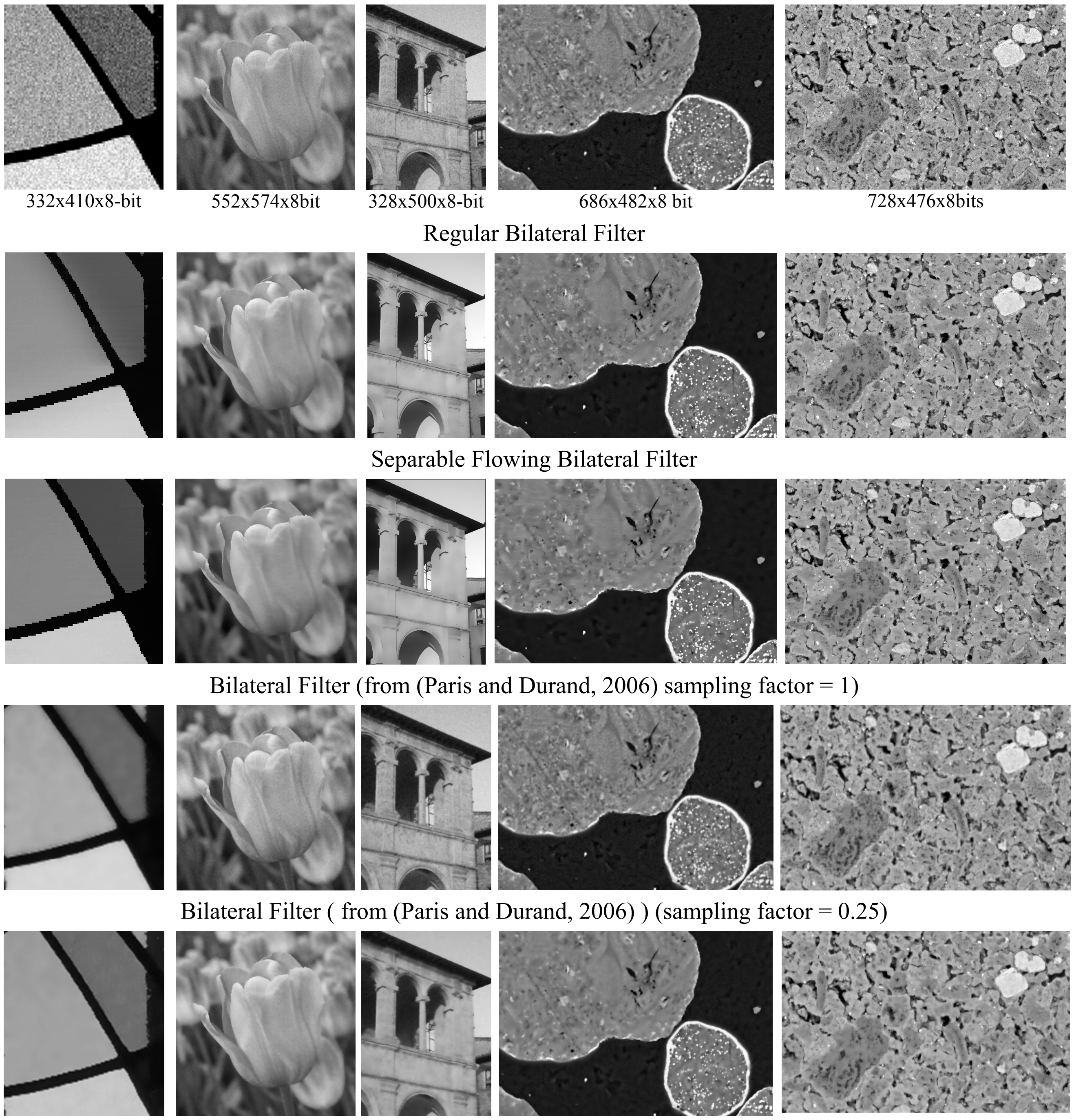

332x410x8-bit
552x574x8bit
328x500x8-bit
686x482x8 bit
728x476x8bits
Regular Bilateral Filter
Separable Flowing Bilateral Filter
Bilateral Filter (from (Paris and Durand, 2006) sampling factor = 1)
Bilateral Filter (from (Paris and Durand, 2006)) (sampling factor = 0.25)

Fig. 5. *Qualitative comparisons on 2D images. For each images, spatial and tonal parameters are equivalent.*

Table 1. *Timings comparisons between Regular, Separable Flowing and (Paris and Durand, 2006) bilateral filter implementations on standard test images.*

"Stained Glass"	**"Tulip"**	**"House Corner"**	**"Catalyst 1"**	**"Catalyst 2"**
Regular Bilateral Filter (CPU / GPU implementation)				
5.24s / 0.1s	12.62s / 0.34s	6.51s / 0.18s	13.14s / 0.1s	13.77s / 0.12s
Flowing Bilateral Filter (CPU implementation)				
24.29s	56.75s	29.77s	60.32s	66.89s
Separable Bilateral Filter (CPU implementation)				
0.336s	0.812s	0.42s	0.845s	0.885s
Separable Flowing Bilateral Filter (CPU / GPU implementation)				
0.35s / 0.01s	0.83s / 0.03s	0.43s / 0.01s	0.87s / 0.02s	0.926s / 0.02s
Bilateral Filter (from (Paris and Durand, 2006) (CPU , subsampling = 1)				
0.6s	1.1s	0.6s	1.4s	1.4s
Bilateral Filter (from (Paris and Durand, 2006) (CPU , subsampling = 0.25)				
12.2s	47.7s	14.8s	63.4s	68.8s

Table 2. *Quantitative comparisons between the proposed implementations of the flowing bilateral filter and other discussed implementations on various 2D images.*

"Stained Glass"			**"Tulip"**			**"House Corner"**		
PSNR	**UIQ**	**SSIM**	**PSNR**	**UIQ**	**SSIM**	**PSNR**	**UIQ**	**SSIM**
Noise								
27.74	0.99	0.99	27.5	0.962	0.963	25.11	0.964	0.965
Regular Bilateral Filter								
31.62	0.996	0.996	33.3	0.992	0.992	28.56	0.987	0.987
Flowing Bilateral Filter								
31.74	0.996	0.996	**33.35**	0.992	0.992	**28.627**	0.987	0.987
Separable Bilateral Filter								
31.6	0.993	0.993	33.77	0.993	0.993	27.99	0.985	0.985
Separable Flowing Bilateral Filter								
32.27	0.997	0.997	34.24	0.994	0.994	**28.627**	0.987	0.987
Bilateral Filter (Paris and Durand. 2006) subsampling = 0.25								
30.23	0.995	0.995	**34.31**	0.994	0.994	28.09	0.985	0.985

"Catalyst 1"			**"Catalyst 2"**		
PSNR	**UIQ**	**SSIM**	**PSNR**	**UIQ**	**SSIM**
Noise					
27.79	0.983	0.983	27.76	0.969	0.969
Regular Bilateral Filter					
31.19	0.992	0.992	28.45	0.97	0.971
Flowing Bilateral Filter					
31.23	0.992	0.992	**28.49**	0.97	0.971
Separable Bilateral Filter					
30.41	0.991	0.991	27.15	0.959	0.96
Separable Flowing Bilateral Filter					
31.6	0.993	0.993	**28.37**	0.97	0.971
Bilateral Filter (Paris and Durand. 2006) subsampling = 0.25					
30.79	0.991	0.991	27.35	0.96	0.961

For these tests, we used a noise removal application. For each image we added a Poisson noise (5%) and performed an analysis of the filtered images generated by five different implementations: the regular bilateral filter, separable bilateral filter, flowing bilateral filter, separable flowing bilateral filter and (Paris and Durand 2006) implementation with two different sampling factors.

On the “Stained Glass” image, no “halo” effects surrounding the two regions border (darker one for the brighter region and brighter one for the brighter region respectively) can be noticed with the flowing bilateral. As we can see, the regular bilateral filter produces “halo” artifacts. Paris and Durand (2006) strategy does not produce these artifacts but, unfortunately, the noise is remaining strongly present with the test realized with a sampling factor equal to 1. Considering a sampling factor at 0.25, a better result can be obtained but the computing time is then much longer (*see* Table 1). Our approach gives the best compromise between image quality (no “halos” artifacts and strong noise reduction) and computing times. We also compared the results generated by these implementations on standard tests images (“Tulip” and “House Corner”) and on 2D Scanning Electron Microscopy images of catalyst supports ("Catalyst1" and "Catalyst2"). Several criterions namely PSNR, UIQ and SSIM (Wang *et al.*, 2004) are used to compare images before and after noise filtering, the results are summarized in Table 2. Except for “Tulip” whose texture is diagonally oriented, the flowing bilateral filter is producing at least as well or better than all the other filters used for the comparisons (cf. the PSNR values). One can note also that the separable version of the flowing bilateral filter is very interesting (*see* Table 1), furthermore its running time on GPU is a hundred time lower than the Paris and Durand (2006) implementation with a 0.25 sampling factor.

COMPUTING TIME

Experimental results presented below are obtained with an NVidia Quadro 4000 and an Intel Xeon QuadCore 2.8Ghz on a 512x512x512 – 32 bits data volume (*see* Fig. 7). We compared the computation times of optimized versions of the separable flowing bilateral filters with Tukey's biweight functions kernels implemented on GPU and CPU multi-threads. CPU code is implemented using the openMP API. In addition to pre-compute filter’s spatial coefficients, the inner loop is parallelized on all cores for each of the passes of the filter, achieving in our case a quasi linear efficiency compared to a single core implementation. Fig. 6 illustrates the computation times for the different implementations described in the previous section. Firstly, it may be observed that the GPU implementations of the separable bilateral filter (*cf.* GPU SEP TUKEY and GPU FLOWING SEP TUKEY) outperform all the others implementations presented here. Indeed, it only needs a few seconds to treat a half GB data volume and bring it back to the CPU. Despite the use of a 3D texture cache, the regular bilateral filter GPU implementation (cf. GPU BRUTE FORCE) is very slow. We can notice that comparison with the CPU implementation (cf. CPU SEP TUKEY) can give an order of magnitude of the reachable speed-up compared to a GPU implementation. With a half kernel size of 39, an acceleration factor of approximately 20 is obtained. Considering a batch of 4 voxels to be computed at the same time by each thread (*i.e.*, via the loading of 4 rows into *shared memory*) we benefit from *ILP* and observe a 40% speed-up compared to the previously described separable implementation. Moreover, we notice no additional computation time change with the flowing bilateral filter considering the GPU or the CPU implementations.

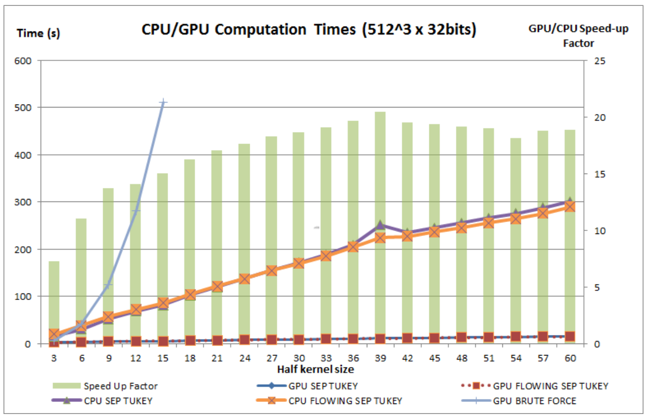


Fig. 6. *Computation times of CPU/GPU implementations of flowing and regular bilateral filters.*

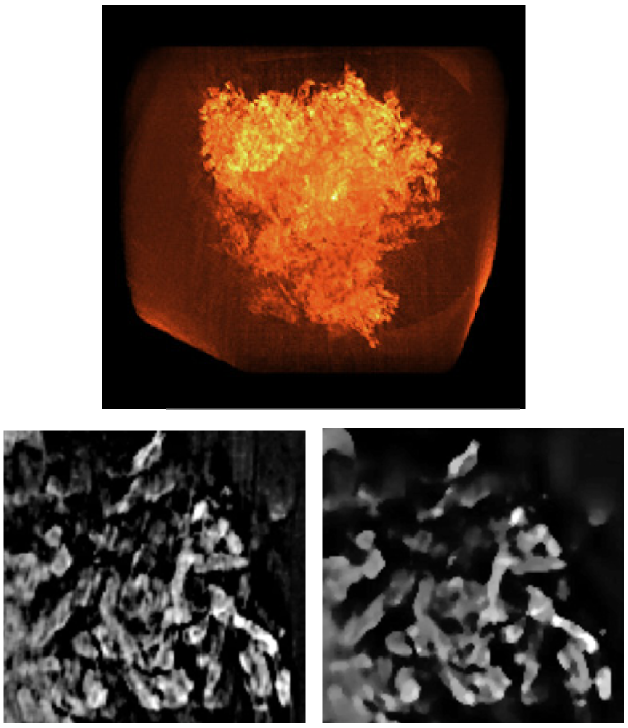


Fig. 7. *3D volume of alumina catalyst (size 512x512x512, resolution 1nm.voxel-1) obtained by electron tomography (Tran et al., 2014). Upper image: 3D observation by volume rendering; Left image: one region of interest (ROI) of one slice of the volume; Right image: same ROI after flowing bilateral filter (parameter spatial 4, intensity 15).*

CONCLUSION

Seeing an image as peaks and valleys, we introduced the flowing bilateral filter, suppressing "halo" artifacts typically produced by the regular bilateral filter around a large peak surrounded by two valleys. The proposed methodology was to combine a morphological reconstruction in the tonal space in order to ensure the strict decreasing of the tonal weighting function. A separable version of this new filter was also proposed. This version requires only little change from the original approximate separable version of the bilateral filter algorithm. We also proposed GPU implementations of the separable flowing bilateral filter by using the NVidia CUDA API. With this version, the global memory of the GPU in the row, column or depth direction can be preloaded into a fast access buffer and is inducing a great speed up compared to a CPU implementation. Indeed, we can reach an acceleration factor up to 20 compared to a CPU implementation parallelized on 4 cores. Time computing of regular or flowing separable bilateral filters on CPU or GPU are almost identical.

ACKNOWLEDGEMENTS

The part of this paper dealing with implementations on GPU of exact and separable bilateral filter was presented at the 11th European Congress of Stereology and Image Analysis, July 9–12, 2013 in Kaiserslautern, Germany.

REFERENCE

Banterle F, Corsini M, Cignoni P, Scopigno R (2012). A low-memory, straightforward and fast bilateral filter trough subsampling in spatial domain. Comput Graphics forum 31:19–32.

Barash D (2002). Fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. IEEE T Pattern Anal Mach Intell 24:844–7.

Bethel EW (2012). Exploration of optimization options for increasing performance of a GPU implementation of a three-dimensional bilateral filter. Technical Report, LBNL-5406E, Lawrence Berkeley National Laboratory, Berkeley CA, USA, 94720.

Chaudhury KN, Sage D, Unser M (2011). Fast bilateral filtering using trigonometric range kernels. IEEE T Image Process 20:3376–82.

Chen J, Paris S, Durand F (2007). Real-time edge-aware image processing with the bilateral grid. ACM T Graphics 26:103.

Durand F, Dorsey J (2002). Fast bilateral filtering for the display of high-dynamic range images. ACM T Graphics 21: 257–66.

Huang TS (1981). Two-Dimensional Signal Processing II : Transforms and Median Filters. Berlin: Springer-Verlag, 1:209–11.

Narendra PM (1981). A separable median Filter for Image Noise Smoothing. IEEE T Pattern Anal Mach Intell 3:20–9.

NVIDIA C (2012). NVIDIA CUDA C Programming Guide 5.0.

NVIDIA Best (2012). NVIDIA CUDA C Best Practices Guide 5.0.

Paris S, Durand F (2006). A fast approximation of the bilateral filter using a signal processing approach. In Proceedings of the European Conference on Computer Vision, pp. 568–80.

Pham TQ, Vliet LJ (2005). Separable bilateral filtering for fast video preprocessing. In IEEE International Conference on Multimedia and Expo, pp. 1–4.

Porikli F (2008). Constant Time O(1) Bilateral filtering. In IEEE International Conference on Computer Vision and Pattern Recognition 1–8.

Salembier P, Serra J (1994). Mathematical Morphology and its Applications to Signal Processing (Special issue), 38.

Serra J (1988). Image Analysis and Mathematical Morphology, Part II: Theoretical Advances, Academic Press, London.

Serra J, Soille P (1994). Mathematical Morphology and its Applications to Signal Processing, Series "Computational Imaging and Vision".

Smith SM, Brady JM (1997). SUSAN – A new approach to low level image processing. Int J Comput Vision 23: 45–78.

Tomasi C, Manduchi R (1998). Bilateral filtering for gray and color images. In IEEE Proceedings of the Sixth International Conference on Computer Vision 836–46.

Tran V-D, Moreaud M, Thiébaut É, Denis L and Becker J M (2014). Inverse Problem Approach for the Alignment of Electron Tomographic Series, Oil & Gas Science and Technology. IFP Energies nouvelles 69:279–91.

Vincent L (1993). Morphological grayscale reconstruction in image analysis: Applications and efficient algorithms. IEEE T Image Process 2:176–201.

Volkov V (2010). Better Performance at Lower Occupancy. Proceedings of the GPU Technology Conference, GTC 10.

Wang Z, Bovik AC, Sheikh HR, Simoncelli EP (2004). Image quality assessment: From error visibility to structural similarity. IEEE T Image Process 13:600–12.

Weiss B (2006). Fast median and bilateral filtering. ACM T Graphics - Proc of ACM SIGGRAPH 25(3):519–26.

Yang Q, Tan KH, Ahuja N (2009). Real-time O(1) bilateral filtering. In IEEE Conference on Computer Vision and Pattern Recognition 557–64.