34th Annual International Conference of the IEEE EMBS
San Diego, California USA, 28 August - 1 September, 2012

3D Nonlinear Complex-Diffusion Filter on GPU

Pedro Rodrigues[1], Pedro Serranho[2], and Rui Bernardes[3]

***Abstract*— The ramp preserving 2D nonlinear complex-diffusion filter introduced by Gilboa *et al.* (2004) was extended to 3D (Maduro *et al.*, 2012). We propose a graphical processing unit implementation of the 3D filter for an overall faster processing in order to be used in a clinical setting.**

We perform a search for the best diffusion parameters (the number of iterations and spread of the diffusivity) for the 2D and 3D filters and compare their results resorting to synthetic spectral-domain optical coherence tomography volumetric data and several quantitative metrics.

Execution time improvement of our implementation versus a single-core approach is also presented, showing that it allows for a full 3D volume to be processed under 7.5 seconds.

I. INTRODUCTION

Image and volume filtering frequently represent the initial steps in most image processing systems and often play a crucial role for the respective systems.

While the majority of medical imaging systems are still based on 2D modalities, *e.g.*, color fundus photography, 3D imaging systems are now taking their place in the daily clinical practice, *e.g.*, OCT (optical coherence tomography), ultrasound, and computer tomography systems.

The increasingly larger sets of data per patient and scan do represent, therefore, an added challenge due to the increase in processing time associated to the larger amount of data, in the one hand, and due to the increase in complexity of algorithms, on the other hand. In consequence, data processing times may render these complex procedures difficult to apply in the analysis of a large number of patients (even in a research environment) and clearly making it difficult (if not impossible) to use in the clinical practice.

With the recent introduction of GPU (graphical processing unit) systems that brought into the daily practice the power of massive parallel computing, a window of opportunities emerged opening the door for more complex and advanced processing methods to be applied resorting to not exclusive computing facilities.

*This work was supported by the Fundação para a Ciência e a Tecnologia under the research projects PTDC/SAU-ENB/111139/2009 and PTDC/SAU-BEB/103151/2008 and by the COMPETE program (FCOMP-01-0124-FEDER-015712 and FCOMP-01-0124-FEDER-010930, respectively)

[1]P. Rodrigues is with Centre of New Technologies for Medicine, Association for Innovation and Biomedical Research on Light and Image Coimbra, Portugal prodrigues at aibili.pt

[2]P. Serranho is with Mathematics Section, Department of Science and Technology, Open University, Lisbon, Portugal and Institute of Biomedical Research in Light and Image, Faculty of Medicine, University of Coimbra, Coimbra, Portugal pserranho at uab.pt

[3]R. Bernardes is with Centre of New Technologies for Medicine, Association for Innovation and Biomedical Research on Light and Image Coimbra, Portugal and Institute of Biomedical Research in Light and Image, Faculty of Medicine, University of Coimbra, Coimbra, Portugal rcb at aibili.pt

In this way, OCT volumetric scans from the human ocular fundus (with data range between between 0 and 1) of 512x1024x128 or 200x1024x200 voxels (x, y, and z directions, respectively) are filtered taking the advantage of a CUDA (compute unified device architecture) implementation for GPU processing.

CUDA is a parallel computing architecture developed by NVIDIA Corporation (Santa Clara, CA, USA). In CUDA C the code runs on the host engine (with one or more CPUs) and on one or more devices (with CUDA-enabled GPUs) concurrently [1]. In turn, GPUs can run thousands of threads in parallel, which makes them a very powerful platform for computationally demanding algorithms that can be solved using massive parallelism such as the one addressed in this paper [1], [2].

The complex-diffusion process for denoising 2D images was proposed by Gilboa *et al.* [3]. Salinas and Fernández [4] optimized this process for 2D OCT scans (B-scans — xy plane) of the ocular fundus and compared the results with the nonlinear Perona-Malik diffusion filter [5]. However, by restricting the diffusion to 2D, one is ignoring possibly relevant information on adjacent B-scans.

In this work, the filter was extended from 2D to 3D [6] and implemented in CUDA for computing performance.

II. NONLINEAR COMPLEX-DIFFUSION FILTER

The general nonlinear diffusion equation is

$$\frac{\partial s}{\partial t} = \nabla \cdot (D \nabla s) \tag{1}$$

where $\nabla\cdot$ and ∇ are the divergence and gradient operators, respectively, D is the coefficient of diffusion, t denotes time and s is the image/volume.

Through the discretization in time of (1) one gets

$$s^{n+1} = s^n + \Delta t \left[\nabla \cdot (D^n \nabla s^n)\right] \tag{2}$$

with Δt a discrete time step and n the iteration number.

The complex diffusivity proposed by Gilboa *et al.* [3], [7] for a ramp preserving process is defined as

$$D = \frac{e^{i\theta}}{1 + \left(\frac{\mathrm{Im}(s)}{\kappa\theta}\right)^2} \tag{3}$$

where $\mathrm{Im}(.)$ denotes the imaginary part, θ is a phase angle close to zero and κ is a constant parameter that modulates the spread of the coefficient of diffusion.

Additionally, a Gaussian filter (with 3x3x3 voxel filter and a standard deviation $\sigma = 1$) is applied to s^n before computing the diffusivity [4].

978-1-4577-1787-1/12/$26.00 ©2012 IEEE

III. IMPLEMENTATION

The algorithm implementation can be divided in an environment setup step, that only runs once, and two loop stages that are called iteratively. The first loop stage filters $\mathrm{Im}(s)$ (kernel K1) and stage two performs the remaining arithmetic operations, namely the computation of $\nabla \cdot (D\nabla s)$ (kernels K2-4) and the update of s (K5).

A. Setup Step

The setup overhead plays an important role for the performance of the algorithm. Device and host memory are allocated, thread block sizes and grid sizes for some of the kernels are defined, configuration of cache sizes is performed, and streams for efficient latency hiding are created.

The Gaussian filter is created and copied to the GPU's constant memory, a low latency read-only memory, to be accessed from all threads in the grid.

In this step we also compute the SM (streaming multiprocessor) occupancy for the kernels K1-4 in order to find flexible thread block dimensions and suitable grids.

Besides occupancy, other restrictions are checked as well, namely the maximum grid sizes, maximum threads per block, and total number of registers allocated per thread block. These restrictions are checked in order to ensure that a given SM has enough resources to be assigned to at least one thread block at a time.

One should note that the number of registers and the formulae for the occupancy varies in between CUDA architectures and are compilation dependent. These factors need to be accounted for in order to achieve a flexible GPU-independent process.

In addition, CUDA-enabled GPUs allow two configurations in terms of L1 cache and shared memory sizes, which is a low latency memory in the same on-chip location as L1 and like L1 is shared by the threads within a thread block. This allows one to switch between maximizing L1 or shared memory for each kernel [8]. Hence, we maximize L1 cache for all kernels in order to reduce cache misses and consequently time-wise expensive global memory fetches. Since, in this process, none of the kernels make use of shared memory.

B. Loop

One cannot guarantee that available global memory is enough to cover the amount of memory required for the loop stage. Consequently, in the setup stage, we have to account for the need to partition the inputs and outputs to be processed separately according to the size of the 3D imaging data and the size of the GPU's global memory.

Some GPUs provide support for the host to asynchronously schedule copies between the host and the device, as well as kernel calls and allow it to execute them simultaneously using CUDA streams — a single stream is a sequence of instructions issued by the host that execute in order, yet different streams may execute their instructions out of order, with respect to one another, or concurrently [8].

At each loop stage the input and output volumes are partitioned into a number of subsets of slices and the last subset is zero-padded for performance. Each subset is processed by a different stream and each stream is responsible to copy a subset of slices to global memory, launch the kernels, and copy the results back to host memory.

When scheduling asynchronous calls, one has to consider that most GPUs do not support concurrent data transfers. Thus, one must not schedule all calls of one stream followed by all operations of another stream and so forth. This would result in no optimization at all (in comparison with a single stream process) and would most likely increase the process overall execution time. Specifically, the host–device copy of the second stream would not take place until the end of the device–host copy transaction of the first stream. Nonetheless, by scheduling the host–device copies and kernel calls for all streams before all device–host copies, one can have kernel execution and copies of different streams happening simultaneously accomplishing good kernel execution hiding. The first device–host copy would have to wait for the last host–device copy for GPUs without support for concurrent copies, however, kernel execution time would still be partially or even completely masked by copy latency.

Before the end of each loop stage, one has to synchronize all streams, *i.e.*, wait for the streams to have finished copying their results to RAM.

C. Kernels

Optimizations were made that consist in using the **#pragma unroll** directive and minimizing use of registers.

In each loop iteration different types of kernel are called. We can classify them into two categories.

1) Spatial Convolution: Regarding the convolution kernel K1, we created a 2D grid composed by 2D thread blocks (x and y dimensions), where each tread computes an array of pixels in z direction with length equal to the number of slices in the subset. Grid and thread block dimensions are, as mentioned before, computed in the setup stage based on occupancy and resource restrictions.

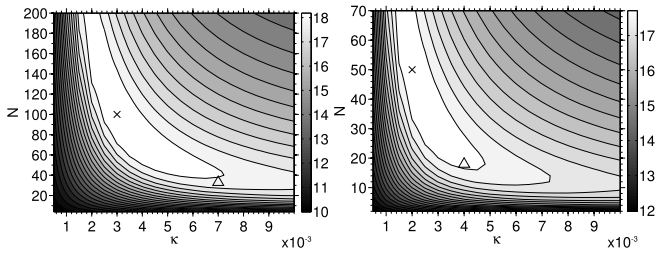
In convolution operations data close to the boundaries (*e.g.*, voxels of the first slice) require manipulation so that there is no data loss from iteration to iteration. To that intent and to minimize warp divergence, we extended data volumes through replication of the outermost voxels prior to kernel launch.

2) Algorithm Specific: Each kernel (K2, K3,and K4) computes $\nabla \cdot (D\nabla s)$ approximation in on direction (x, y, and z, respectively) for the given subset.

3) Element-Wise Arithmetic: As to K5, it uses a 1D grid with 1D thread blocks with hard-coded sizes It is responsible for simple element-wise arithmetic operations over complex volumes. Each thread computes and writes the results for the indexes that are located in the thread's index on the grid plus multiples of the number of threads on the grid.

IV. RESULTS

Serranho *et al.* [9] proposed a way of generating synthetic noise-free OCT volumes as well as OCT speckle noise to



(a) (b)

Fig. 1. S/MSE isolines for several (κ, N) pairs for (a) the 2D and (b) the 3D filters. The crosses earmark the best pair for each method and the triangles earmark the chosen parameters for the computation of the metrics in Table I.

use as a system for evaluation of image processing methods. These computer generated synthetic retinas will be used here to evaluate the 3D despeckling filter and to compare it to the 2D process, thus providing the same ground truth for both filters.

To filter the OCT scans of the ocular fundus, a $\theta = \pi/30$ [3], a $\Delta t = 0.24$ seconds for the 2D filter, and a $\Delta t = 0.16$ seconds for the 3D filter were used [10]. As for κ and the number of iterations N, we optimize them for both filters. As such, the signal-to-mean-square-error (S/MSE) is computed

$$S/MSE = 10\log_{10}\left[\frac{\sum_{x,y,z}(s-\mu)^2}{\sum_{x,y,z}(\hat{s}-s)^2}\right] \tag{4}$$

with $\hat{s}$ the filtered volume and s the synthetic volume The results are shown in Fig. 1 for the 2D and the 3D processes using 8 synthetic retinas.

The maximum of the S/MSE does occur for (κ, N) of $(0.003, 100)$ for the 2D and $(0.002, 50)$ for the 3D.

Additionally, other metrics were computed for comparison between the original and the filtered versions with both processes. These were the signal-to-noise ratio, computed as

$$SNR = \frac{\mu}{\sigma} \tag{5}$$

with σ the standard deviation of the data; the effective number of looks

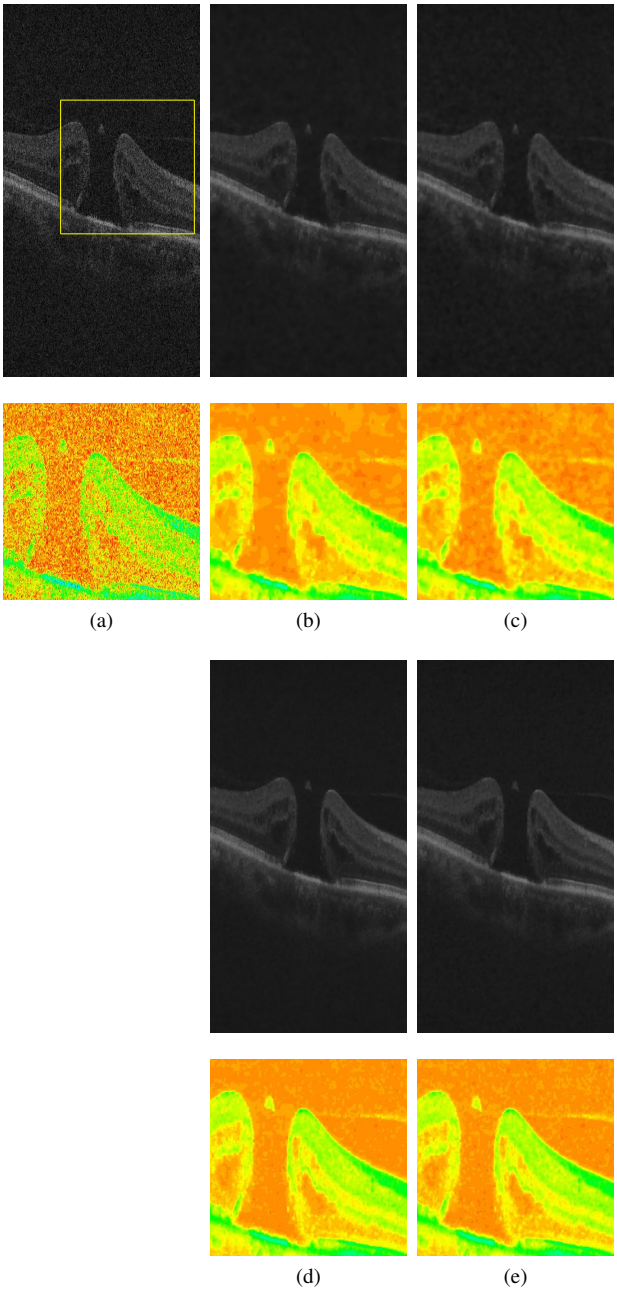
$$ENL = \left(\frac{\mu_h}{\sigma_h}\right)^2 \tag{6}$$

where h stands for homogeneous region; the contrast-to-noise ratio

$$CNR = 10\log\left(\frac{\mu_f - \mu_r}{\sqrt{\sigma_f^2 + \sigma_r^2}}\right) \tag{7}$$

where f is a feature and r denotes background noise reference; and the mean structure similarity index [11], here expanded to 3D,

$$MSSIM = \frac{1}{V}\sum_{x,y,z} SSIM(s, \hat{s}) \tag{8}$$



(a) (b) (c) (d) (e)

Fig. 2. (a) Original OCT B-scan, (b) OCT filtered with the 2D process with $\kappa = 0.003$ and 100 iterations and (c) $\kappa = 0.007$ and 33 iterations, and (d) OCT filtered with the 3D process with $\kappa = 0.002$ and 50 iterations and (e) $\kappa = 0.004$ and 18 iterations. Enlarged details in a pseudo-color space are show for ease of comparison.

with

$$SSIM(s_1, s_2) = \frac{(2a_1a_2 + C_1)(2d_{12} + C_2)}{(a_1^2 + a_2^2 + C_1)(d_1^2 + d_2^2 + C_2)} \tag{9}$$

where V is the number of voxels, $C_1 = (0.01L)^2$, $C_2 = (0.03L)^2$, and L is the maximum intensity scale value of the data (*e.g.*, 255 for 8-bit data). The values a_j, d_j and d_{12} are

TABLE I

COMPARISON TABLE BETWEEN SYNTHETIC NOISY OCT AND THE RESPECTIVE 2D AND 3D FILTERED VOLUMES.

	κ	N	Δt (s)	t (s)	S/MSE	SNR	ENL	CNR	MSSIM
Noisy	-	-	-	-	5.62±1.11	1.63±0.10	6.14± 0.39	4.68±1.57	0.39±0.02
2D Filtered	0.003	100	0.24	24.00	18.60±0.84	1.89±0.16	212.25± 66.94	7.08±1.69	0.96±0.00
	0.007	33	0.24	7.92	18.09±0.85	1.87±0.15	142.75± 33.39	6.85±1.66	0.95±0.01
3D Filtered	0.002	50	0.16	8.00	17.99±0.97	1.88±0.16	322.84±140.17	6.97±1.67	0.96±0.01
	0.004	18	0.16	2.88	17.77±0.98	1.88±0.16	240.21± 82.91	6.86±1.66	0.96±0.01

TABLE II

EXECUTION TIMES (IN SECONDS) FOR THE SINGLE-CORE IMPLEMENTATION ON MATLAB AND THE CUDA C IMPLEMENTATION ON A GPU, AS WELL AS THE AVERAGE PERFORMANCE IMPROVEMENT.

N iterations	Single Core	GTX 570	Speedup
1	17.007± 0.686	1.154±0.050	14.74
18	393.336±14.559	7.419±0.323	53.02
50	1074.277±39.791	19.421±0.853	55.32

given by

$$\begin{aligned} a_j &= \sum_k g(k) w_j(k) \\ d_j &= \sqrt{\sum_k g(k) \left(w_j(k) - a_j\right)^2} \\ d_{12} &= \sum_k g(k) \left(w_1(k) - a_1\right) \left(w_2(k) - a_2\right) \end{aligned} \tag{10}$$

with $j = 1, 2$, where w_j represents local 3D windows of the volume s_j in the neighborhood of the voxel (x, y, z) and g is a 3D Gaussian weight function (of integral equal to one and standard deviation 1.5 [11]).

Table I summarizes all metrics for the (κ, N) pairs that result in the highest S/MSE and the pair $(0.004, 18)$, for the 3D algorithm, that was chosen to decrease the total processing time without compromising the quality measures. For the 2D filter the metrics were also computed for $N = 50$, which corresponds to a similar total diffusion time t as $N = 18$ in the 3D filter, and $\kappa = 0.007$ (the best possibility for that N). Fig. 2 shows a B-scan from an actual OCT filtered with the respective parameters.

ENL was computed in the vitreous and CNR was computed in the retina with the vitreous as reference.

Detailed execution times and speedups for our implementation versus an optimized single-core Matlab version are shown in Table II. These times were obtained from 10 runs on each of the 8 OCTs, on a system with a 3.4 gigahertz CPU and a GeForce GTX 570 GPU.

Through the optimization of fundamental parameters, it was possible to achieve similar results at a fraction of the number of iterations that, added to the CUDA implementation, rendered a speedup factor over 140.

V. DISCUSSION

Graphic processing units are now becoming widely available at most desktop computers. Besides traditional applications, *e.g.*, computer graphics rendering, the massive parallel power can be used for scientific computation and is specifically tailored for naturally parallel processes as convolutions, Fourier transforms and the like. In this work we took advantage of this computing power to address the need for the denoising of a medical imaging modality, the optical coherence tomography, a coherent based imaging system. Wave coherent based imaging suffers from a specific noise, speckle noise, that renders a grainy appearance potentially masking important information.

This initial approach will be tested on a recently proposed adaptive filter [12].

ACKNOWLEDGMENT

The authors would like to thank Carl Zeiss Meditec, Inc. (Dublin, CA, USA) and Dr. M. Horne for their support on getting access to OCT data.

REFERENCES

[1] NVIDIA, CUDA C Best Pratices Guide, NVIDIA, May 2011.
[2] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, GPU computing, in Proceedings of the IEEE, vol. 96, no. 5, May 2008, pp. 879-899.
[3] G. Gilboa, N. Sochen, and Y. Zeevi, Image enhancement and denoising by complex diffusion processes, IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 8, pp. 1020-1036, August 2004.
[4] H. Salinas and D. Fernández, Comparison of pde-based nonlinear diffusion approaches for image enhancement and denoising in optical coherence tomography, IEEE Trans. Med. Imaging, vol. 26, no. 6, pp. 761-771, June 2007.
[5] P. Perona and J. Malik, Scale-space and edge detection using anisotropic diffusion, IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 7, pp. 629-639, Jul. 1990.
[6] C. Maduro, P. Serranho, T. Santos, P. Rodrigues, J. Cunha-Vaz, and R. Bernardes, OCT Noise Despeckling Using 3D Nonlinear Complex Diffusion Filter, in: Technologies for Medical Sciences, Lecture Notes in Computational Vision and Biomechanics, R. Jorge and J. Tavares, M. Barbosa, and A. Slade, Eds. Springer, 2012, vol. 1, pp. 141-157.
[7] G. Gilboa. Linear and Nonlinear Diffusions (Perona-Malik). Accessed on May 2012. [Online]. Available: http://visl.technion.ac.il/ gilboa/PDE-filt/diffusions.html
[8] NVIDIA, NVIDIA CUDA C Programming Guide, NVIDIA, June 2011.
[9] P. Serranho, C. Maduro, T. Santos, J. Cunha-Vaz, and R. Bernardes, Synthetic OCT data for image processing performance testing, in Proc. of 18th IEEE International Conference on Image Processing (ICIP), 2011, pp. 409-412.
[10] A. Araújo, S. Barbeiro, and P. Serranho, Stability of finite difference schemes for compex diffusion processes, SIAM J. Numer. Anal. (in press).
[11] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process., vol. 13, no. 4, pp. 1-14, April 2004.
[12] R. Bernardes, C. Maduro, P. Serranho, A. Araújo, S. Barbeiro, and J. Cunha-Vaz, Improved adaptive complex diffusion despeckling filter, Opt. Express, vol. 18, no. 23, pp. 24 048-24 059, November 2010.