GPU Edge Preserving Salt and Pepper Image Denoising

C. Spampinato
Department of Informatics and Telecommunication Engineering
University of Catania - Viale Andrea Doria, 6 - 95125 - Catania
e-mail: cspampin@diit.unict.it

***Abstract*— In this paper a two-phase filter for removing salt-and-pepper noise is described. In the first phase, an adaptive median filter is used to identify pixels, which are likely to be affected by noise (noise candidates). In the second phase, the noisy pixels are restored according to a regularization method, which contains a data-fidelity term reflecting the impulse noise characteristics, to be applied only to the identified (by the first phase algorithm) noise candidates. The algorithm was firstly sequentially implemented obtaining very encouraging results in terms of PSNR and MAE and then an implementation of the same algorithm was performed using GPU programming with CUDA, thus increasing the time performance of about 50% with respect to the sequential algorithm.**

***Keywords*— Impulsive noise, Edge-preserving regularization, GPU Image Processing**

I. Introduction

In the last decades, the image-processing field has became more interesting, sustained by the continuous improvements in electrical and computer engineering. The increasing of the computing (processing) power has allowed the researchers to extend the number of applications in this field. As is known, a typical image machine vision system consists of three linked building blocks [1] that perform different tasks. An important step of the lowest level block is the noise removal, since it highly influences the performance of the overall machine vision system.

Therefore, the need for an efficient and effective image restoration method is high and it has grown with the massive production of digital images and movies, often grabbed in poor conditions. A typical noise that affects digital images is *Salt and Pepper noise* [2], which is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware or transmission in a noisy channel [3]. This kind of impulsive noise sets the corrupted pixel value to the maximum or the minimum of the pixels variation range (0 or 255 for an 8-bit image). During the last fifteen years, a large number of methods have been proposed to treat "Salt and Pepper Noise" (more in general impulse noise) from digital images (e.g. [3], [4], [5], [6]) and many others are oriented for image details preserving as the filters reviewed in [7]. Most of these methods are order statistic filters that use the rankorder information of an appropriate set of noisy input pixels and usually consist of a general framework of rank selection filters. The median filter is the most popular nonlinear filter for removing impulse noise, because of its good denoising power [3], its computational efficiency [8]. Application of median filtering to an image, however, requires some caution because it tends to remove image details while removing noise. Moreover, the performance of median filtering is unsatisfactory in suppressing signal-dependent noise [9] and when the noise percentage is quite high. To achieve a good compromise between the image-detail preservation and the noise reduction an impulse detector must be used before filtering. The filtering is then selectively applied to regions where there impulse noise is detected. As far as we know, one the most effective algorithm for edge preserving in salt and pepper denoising is the one proposed by Nikolova [6] that applies a variational method for image details preserving that is based on a data-fidelity term related to the impulse noise. Based on this approach Chan *et al.* in [10] proposed a powerful filter, where the noisy pixel detection is carried out by using an adaptive median filter, that is able to remove salt and pepper noise as high as 90%. In this work the performance of this powerful filter is analyzed with various images. Let us recall that, more precisely, the filter consists of the following phases: 1) the noise candidates are first identified by a detector based on Adaptive Median Filter and then 2) these noise candidates are selectively restored using an objective function with a data-fidelity term and an edge-preserving regularization term. Finally, the algorithm has been implemented with CUDA, which allows us to increase time performance of about 50% with respect to the sequential version of the algorithm. The outline of the paper is as follows: in the next section the overall architecture of the denoising filter is outlined. Sect. III shows the used variational method for noisy pixels restoration. Section IV shows the performance of the sequential implementation of the described filter, whereas sect. V describes the CUDA implementation of the proposed algorithm. Finally, the last section points out the conclusions and the future work.

II. Two Steps Edge Preserving Filter Architecture

The described filter is a two-phase algorithm, indeed, it consists of an Adaptive Median Filter classifier for noisy pixels identification and of a variational method [6] for restoring all the pixels that have been identified as *noisy pixels* by the first block. More in detail the two stages of the algorithm are:

- *Noisy Pixels Identification by using Adaptive Median Filter* - Let us denote by $\hat{y}$ the map obtained by the adaptive median filter classifier that has an one in correspondence of the position of the noisy pixels, whereas it has a 0 in correspondence of the uncorrupted pixels. Hence the set of noisy pixels (where the restoration algorithm has to be applied) consists of the overall pixels of the original

image y whose values in the $\hat{y}$ map are equal to 1. Hence the set of noisy pixels is defined as follows:

$$N = \{(i,j) \in \mathrm{A} : \hat{y}_{i,j} = 1\}$$

The set of all uncorrupted pixels is $N^c = N/A$, where A is the set of all pixels and N is the set of the noisy pixels.

- *Variational Method for noisy pixels restoration* - The problem of image restoration for edge preserving is an inverse problem solved by using regularization. In this work a variational approach, where the restored image u is the solution of the following optimization problem restricted to the set of the noisy pixels N.

$$\min_{u \in N} F(u) = \alpha \int R(u) + \beta \int D(u,d) \qquad (1)$$

Where d is the image corrupted by the noise, $D(u,d)$ is the data-fidelity term that is related to the kind of noise and provides a measure of the dissimilarity between d and the output image u, whereas $R(u)$ is a regularization term that uses a-priori knowledge for enforcing the solution, β and α are the regularization parameters that balance the effects of both mentioned terms.

III. Variational Method for Noisy Pixels Restoration

As previously described, once the noisy pixels have been identified, the restoration step for the edge preserving is then carried out by using a functional in the form described in formula 1. Despite of all the functionals for edge preserving proposed during the last fifteen years, the one used in this work is:

$$F_d|_N(u) = \sum_{(i,j) \in N} [|u_{i,j} - d_{i,j}| + \frac{\beta}{2}(S_1 + S_2)] \qquad (1)$$

where

$$S_1 = \sum_{(m,n) \in V_{i,j} \cap N} 2 \cdot \varphi(u_{i,j} - d_{m,n})$$
$$S_2 = \sum_{(m,n) \in V_{i,j} \cap N} \varphi(u_{i,j} - u_{m,n})$$

where N represents the noisy pixels set and $V_{i,j}$ is the set of the four closest neighbors of the pixel with coordinates (i,j).

The considered functional is given by the sum of two terms: *a data fidelity term* proposed by Nikolova in [6], which represents the deviation from a data image y, which may be marred by noise, and the *regularization term* that incorporates the variation of a function that penalizes oscillations and irregularities, although does not remove high level discontinuities. The last term is responsible for the edge preserving. Both methods, according to Nikolova, use the data fidelity term $|u - d|$. Generally an iterative method, related to the percentage of noise, is used for the functional minimization, [11], so that the convergence rate depends on the image smoothness. This minimization algorithm works on the residuals $z = u - y$ of the functional (1) and it is following shown.

Algorithm for functional minimization

- Initialize $z_{ij}^{(0)} = 0$ for each $(ij) \in A$;
- At each iteration k, calculate, for each $(ij) \in A$,

$$\xi_{i,j}^{(k)} = \beta \sum_{(m,n) \in V_{i,j}} \dot{\varphi}\left(y_{i,j} - z_{i,j} - y_{m,n}\right)$$

where $z_{m,n}$, for $(m,n) \in Vi,j$, are the latest updates and $\dot{\varphi}$ is the derivative of φ that we choose equal to $|t|^{\alpha}$.

- If $\xi_{i,j}^{(k)} = 1$, set $z_{i,j} = 0$. Otherwise, solve for $z_{i,j}^{(k)}$ in the nonlinear equation:

$$\beta \sum_{(m,n) \in V_{i,j}} \dot{\varphi}\left(z_{i,j}^{(k)} + y_{i,j} - z_{m,n} - y_{m,n}\right) = sign\left(\xi_{i,j}^{(k)}\right)$$

The updating of $z_{i,k}^{(k)}$ was done according to [11], i.e. that $z^{(k)}$ converges to $\hat{z} = \hat{u} - y$, where the restored image $\hat{u}$ minimizes F_y in (1). By choosing $\phi(t) = |t|^{\alpha}$, the nonlinear equation (1) can be solved by Newton's method with quadratic convergence by using a suitable initial guess derived in [12].

IV. Experimental Results for the Sequential Algorithm

The algorithm was firstly implemented sequentially in MATLAB 7.4.0 and tested on a PC with a CPU Intel Core 2 Duo T7700 with 2GB of RAM and with a graphics board GeForce 8600M GS 256MB. Among the commonly tested 256-by-256 8-bit gray-scale images, the one with homogeneous region (Lena, Pepper) and the ones with high activity (Bridge, Baboon) were selected for the simulations. In the simulations, images were corrupted by "salt" (with value 255) and "pepper" (with value 0) noise with equal probability. Also a wide range of noise levels varied from 10% to 90% with increments of 20% was tested. The images used for evaluation of the algorithm performance are shown in fig. 1.

Restoration performance is quantitatively measured by the peak signal-to-noise ratio (PSNR) and the mean absolute error (MAE) defined as follows:

$$PSNR = 10 \cdot \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i,j} (r_{i,j} - x_{i,j})^2}$$
$$MAE = \frac{1}{MN} \sum_{i,j} (r_{i,j} - x_{i,j})^2$$

where $r_{i,j}$ and $x_{i,j}$ denote, respectively, the pixel values of the restored image and the original image. Moreover, CPU-TIME (in seconds) was used to test time performance. Tables 1, 2, 3 and 4 shows the PSNR, MAE and CPU-TIME, respectively, for Lena, Peppers, Bridge and Baboon images reported at varying of noise level (from 10% to 90% at step of 20%) and with α set to 1.15. This value was experimentally identified, in fact, a lower value allowed us to remove noise but not staircases effect. With a higher value, instead, the noise could not be fully removed. Hence the selection of α was a trade-off between noise suppression and detail preservation as shown in [6]. In our tests we set $\phi(t) = |t|^{1.15}$

Fig. 1. Images used for denoising algorithm testing

and β was tuned to give the best result in terms of PSNR.

% Noise	PSNR	MAE	CPU-TIME (sec)
10%	41.90	0.46	834.89
30%	36.07	1.57	2342.126
50%	33.05	2.90	2945.23
70%	29.70	4.98	5890.11
90%	23.57	11.61	9356.24

Table 1. PSNR, MAE and CPU-TIME for different noise levels for *Lena Image*

% Noise	PSNR	MAE	CPU-TIME (sec)
10%	42.07	0.71	845.32
30%	37.15	1.23	2361.11
50%	34.96	3.11	3287.19
70%	28.09	5.01	5911.40
90%	22.76	12.36	10297.86

Table 2. PSNR, MAE and CPU-TIME for different noise levels for *Peppers Image*

% Noise	PSNR	MAE	CPU-TIME (sec)
10%	36.41	2.39	919.06
30%	32.42	3.94	3149.84
50%	29.33	5.49	5671.03
70%	25.74	8.44	7276.10
90%	21.17	15.71	11833.78

Table 3. PSNR, MAE and CPU-TIME for different noise levels for *Bridge Image*

To better appreciate the performance in denoising of the filter, fig. 2 shows the output images for Bridge and Baboon when they are affected by 70% of noise.

V. GPU Implementation with CUDA

The optimization of the original algorithm was led by the code profiling according to the Amdahl law [13]. The

% Noise	PSNR	MAE	CPU-TIME (sec)
10%	34.74	3.11	912.141
30%	31.82	4.48	4010.81
50%	29.16	6.29	6025.90
70%	23.91	8.23	8112.34
90%	19.58	17.41	12368.41

Table 4. PSNR, MAE and CPU-TIME for different noise levels for *Baboon Image*

parallelization of the code was divided in two steps:

- Adaptive Median filter optimization;
- Variational Method optimization.

The first step was optimized using a matricial implementation of the adaptive median filter instead of a sequential one. At this step no GPU parallelization was performed since some function in the code didn't use local variables. The parallelization was made entirely for the variational method. The matricial implementation allowed us to achieve better performance for identifying noisy pixels, which means that, if for instance the execution of the AMF (in its sequential form) usually takes about 159 seconds for a 256-256 image, by introducing the matricial implementation it takes about 25 seconds.
The optimization of the variational method was done by taking into account both CPU and GPU parallelization method. More in detail, the code has been re-written, in a deterministic way and without field effects, to solve $z_{i,j}^{(k)}$ in the following nonlinear equation:

$$\beta \sum_{(m,n)\in V_{i,j}} \dot{\varphi}\left(z_{i,j}^{(k)} + y_{i,j} - z_{m,n} - y_{m,n}\right) = sign\left(\xi_{i,j}^{(k)}\right)$$

The processing of the set of noisy pixels in the above equation was performed by the simil-BLAS operations (*gfor/gend)* of Jacket (CUDA for MATLAB). Moreover, the computation of the above differential equation was also CPU-parallelized with *parfor* (parallel toolbox MATLAB). The evaluation of the performance was carried out on Lena Image (256x256) at varying of the percentage of affecting noise and by comparing the sequential implementation, the CPU-Parallelization and the GPU-Parallelization.

Fig. 2. Outputs of the proposed algorithm for Bridge and Baboon images affected by 70% of noise.

The obtained results are shown in table 5.

%	Sequential	CPU	GPU
10%	605.354	434.878	370.153
30%	2542.871	1851.267	1376.632
50%	3676.252	2736.652	1931.140
70%	4867.286	3455.712	2676.852
90%	7421.046	5268.942	3784.733

Table 5. Performance (in sec) comparison for solving the differential equation in the variational method for *Lena Image*

Therefore, CUDA implementation allowed to increase time performance of about 50%.

VI. Conclusions and Future Work

In this paper, a two steps filter aiming at well preserving image details is proposed. It represents a very promising filter for removing salt-and-pepper noise in the image denoising field and its performance in reducing noise and in preserving image details are good, so that the idea to use it for image encryption is taking place. Experimental results show that this algorithm outperforms existing median-based or soft computing approaches both in the noisy pixels identification phase and in the restoring phase. Even at a very high noise level (90%), the texture, the image details and the edges are not smeared. An important aspect is the efficiency of the algorithm, indeed, the performance of the proposed filter in terms of $CPU - TIME$ is quite low even using the GPU implementation. In fact, the GPU implementation allowed us to reduce the time taken for denoising of about 50%, but this is still not enough, especially if it should be used for encryption purposes. In order to improve the performance, distributed genetic algorithms on Grid Computing have been already implemented for the restoration phase but they need a deeper evaluation, even if early results are encouraging since a reduction of 10 times (1500 sec) for restoring Baboon image when it is corrupted with 90% of noise was achieved. Moreover, we are planning to implement with CUDA the algorithm proposed by the author in [14], which outperforms the one here described. Future works will also aim to reduce more the processing time by using message passing interface (MPI) architecture.

VII. Acknowledgments

The work was carried out under the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission - Capacities Area - Research Infrastructures.

References

[1] E. Davies, *Machine Vision: Theory, Algorithms, Practicalities, Third Edition*. Elsevier, 2005.

[2] R. Gonzales and R. Woods, *Digital image processing*. Prentice Hall, 2002.

[3] A. Bovik, *Handbook of Image and Video Processing*. Academic Press, 2000.

[4] L. Bar, N. Sochen, and N. Kiryati, "Image deblurring in the presence of salt-and-pepper noise," in *Scale Space and PDE Methods in Computer Vision* (R. Kimmel, N. Sochen, and J. Weickert, eds.), vol. 3459 of *Lecture Notes in Computer Science*, pp. 107–118, Springer Berlin / Heidelberg, 2005.

[5] F. Li and J. Fan, "Salt and pepper noise removal by adaptive median filter and minimal surface inpainting," in *CISP '09. 2nd International Congress on Image and Signal Processing, 2009*, pp. 1 –5, oct. 2009.

[6] M. Nikolova, "A variational approach to remove outliers and impulse noise," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, pp. 99–120, 2004.

[7] J. Astola and P. Kuosmanen, *Fundamentals of Nonlinear Digital Filtering*. Boca Raton, CRC, 1997.

[8] T. S. Huang, G. J. Yang, and G. Y. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 1, pp. 13–18, 1979.

[9] P. Yang and O. Basir, "Adaptive weighted median filter using local entropy for ultrasonic image denoising," in *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the 3rd International Symposium on*, vol. 2, pp. 799–803 Vol.2, Sept. 2003.

[10] R. Chan, C. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization," *IEEE Trans. on Image Processing*, vol. 14, pp. 1479–1485, October 2005.

[11] C. R. Vogel and M. E. Oman, "Fast, robust total variation-based reconstruction of noisy, blurred images," *IEEE Transactions on Image Processing*, vol. 7, no. 6, pp. 813–824, 1998.

[12] J.-F. Cai, R. H. Chan, and C. Fiore, "Minimization of a detail-preserving regularization functional for impulse noise removal," *J. Math. Imaging Vis.*, vol. 29, no. 1, pp. 79–91, 2007.

[13] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[14] A. Faro, D. Giordano, G. Scarciofalo, and C. Spampinato, "Bayesian networks for edge preserving salt and pepper image denoising," in *IPTA08*, pp. 1–5, 2008.