Turkish Journal of Science & Technology
Volume 10 (2), 23-27, 2015

GPU Accelerated Image Noise Removal Approach Based on Subpixel Anisotropic Diffusion

Yanhui Guo[1] and Wei Wei[2]

[1] Department of Computer Science, University of Illinois Springfield, Springfield, IL 62703.
[2] Samsung Research America, Inc. Mountain View, CA 94043.

(Received: 24.12.2014; Accepted: 10.04.2015)

Abstract

Denoising is the critical step in digital imaging. This paper improved an image removal algorithm based on subpixel anisotropic diffusion and GPU accelerate technique. The subpixel difference of an image is explored into the diffusion equation (DE). The DE is solved numerically in the subpixel domain and accelerated by using GPU processing. A GPU implementation of the algorithm is presented to alleviate the time costs. The noise is removed using the diffusion procedure. The experimental results show that the proposed algorithm achieved better performance with same denoising results with less time consumptions the GPU acceleration improved the algorithm more suitable for real time application.

Keywords*:* Diffusion equation, subpixel, image noise removal, blocky effect, GPGPU, OpenCL

1. Introduction

Noise occurs frequently in imaging systems such as image acquisition and transmission process, and it is necessary to remove noise and restore the corruption in image [1]. Image denoising is still a challenging problem for the dilemma removing noise and introducing new artifacts and blurring [1].

Image denosing technique has been applied to real-time application like clinical medical imaging and robot navigation. To fully benefit from this real-time capability, image processing methods also need to be implemented in real-time, which is also crucial for real-time applications. Many methods were proposed to remove the noise in the real time. A class of multichannel filters, vector rank M-type K-nearest neighbor (VRMKNNF) filters were proposed to obtain the balance between noise suppression, and edge and fine detail preservation [2]. They combined RM-estimators with different influence functions. An adaptive non parametric approach determined the functional form of the density probability of noise from data into the sliding filtering window. The filters were implemented on the DSP demonstrating that their potential ability to a real-time solution to quality video transmission. The paper [3] analyzed the relationship between noise variance and gray scale value in real-time X-ray images. Then an adaptive local noise reduction filter was proposed to filter the noise in image.

Merging technology of general purpose graphics processing unit (GPGPU) application brings the possibility of real-time processing [5,6,7,10,11,12], such as image denosing. The paper [4] presented a modified version of the NL-means method for real-time denoising of ultrasound images. The NL-means method incorporates an ultrasound dedicated noise model, and is implemented using a GPU. The paper [5] incorporated both temporal information and image information into the NLMeans algorithm for denoising. The NLMeans algorithm was mapped onto parallel computing architectures and implemented using the GPU. The denoising filter can process video sequences images in real-time on a mid-range GPU. The paper [6] performed the multidimensional adaptive filtering using GPUs in echocardiography images. Filtering was done using multiple kernels implemented in OpenCL

(open computing language) working on multiple subsets of the data. The paper [7,10,12] employed particle swarm optimization (PSO) for solving the importation band selection problem in hyperspectral image processing area. Due to the nature of PSO as an evolutionary algorithm, large amount of computation is needed to search the optimal. A parallel algorithm running on GPU is proposed and make the algorithm could return real-time results.)

Modern graphic cards have high computing and programming capabilities for a reasonable monetary cost. The latest generation of AMD graphics card has 2046 Stream Processors that can work in parallel with wide band memory. The OpenCL framework allows to program graphics cards to solve intensive computation problems with a high-level language. The affordable GPU push scientists to utilize the general purpose programming on graphic processor unit for the attempt of real-time computation problems.

A few noise removal algorithms have been proposed based on anisotropic diffusion [8]. Previously, we extended the traditional anisotropic diffusion filtering method by using subpixel technique. Because of the high computation consumption, it is hard to be employed in the real time image noise removal.

In this paper, we improve the subpixel anisotropic diffusion denoising algorithm [9] by acceleration using a GPU. The subpixel difference is defined in the image and employed to solve the diffusion equation for noise removal. The diffusion equation is solved numerically using an iterative approach. The noise is removed after the diffusion procedure is finished. The subpixel anisotropic diffusion method is a computational intense algorithm. In order to increase the computation speed, we employ GPU to accelerate the algorithm, because it has good characteristics to be ported on GPU: every pixel computation needs a regular analysis of same type of zones over the image input in each iteration. The proposed method was tested on a variety of images, and the experimental results show it can improve the computation speed and achieve better noise removal performance at the same time.

This paper is organized as follows. In Section 2, the subpixel anisotropic diffusion (SAD) is introduced and GPGPU implementation of the SAD method is given. The experimental results are provided and discussed in Section 3. Finally, the paper is concluded in Section 4.

2. Proposed method

Subpixel anisotropic diffusion: The subpixel algorithm considers the pixels between the integer grids. The subpixel intensity values $I_{i+h_S,j}$, $I_{i-h_S,j}$, $I_{i,j+h_S}$ and $I_{i,j-h_S}$ are computed using a linear interpolation based on the intensity of pixels $P_{i,j}$ at the coordinates (i, j) and its neighbors. The diffusion equation is rewritten using the subpixel difference $\nabla_S I$ as [9]:

$$I_{i,j}^{t+1} = I_{i,j}^t - \lambda_S \nabla_{Sx} g_{Sx\,i,j}^t - \lambda_S \nabla_{Sy} g_{Sy\,i,j}^t \quad (1)$$

$$\nabla_{Sx} g_{Sx\,i,j}^t = \frac{g_{Sx\,i+h_S,j}^t + g_{Sx\,i-h_S,j}^t - 2g_{Sx\,i,j}^t}{h_S^2} \quad (2)$$

$$\nabla_{Sy} g_{Sy\,i,j}^t = \frac{g_{Sy\,i,j+h_S}^t + g_{Sy\,i,j-h_S}^t - 2g_{Sy\,i,j}^t}{h_S^2} \quad (3)$$

$$g_{Sx} = c(|\nabla_S I|^2)\nabla_{Sx} I \quad (4)$$

$$g_{Sy} = c(|\nabla_S I|^2)\nabla_{Sy} I \quad (5)$$

$$g_{Sx\,i,j}^t = c(|\nabla_S I_{i,j}^t|^2)\nabla_{Sx} I_{i,j}^t \quad (6)$$

$$g_{Sy\,i,j}^t = c(|\nabla_S I_{i,j}^t|^2)\nabla_{Sy} I_{i,j}^t \quad (7)$$

$$\nabla_S I_{i,j}^t = \nabla_{Sx} I_{i,j}^t + \nabla_{Sy} I_{i,j}^t = \frac{I_{i+h_s,j}^t + I_{i-h_s,j}^t + I_{i,j+h_s}^t + I_{i,j-h_s}^t - 4I_{i,j}^t}{h_s^2} \quad (8)$$

$$\nabla_{Sx} I_{i,j}^t = \frac{I_{i+h_s,j}^t + I_{i-h_s,j}^t - 2I_{i,j}^t}{h_s^2} \quad (9)$$

$$\nabla_{Sy} I_{i,j}^t = \frac{I_{i,j+h_s}^t + I_{i,j-h_s}^t - 2I_{i,j}^t}{h_s^2} \quad (10)$$

where $\nabla_S(\bullet)$ is the subpixl difference function, $\nabla_{Sx}(\bullet)$ and $\nabla_{Sy}(\bullet)$ are the partial differences along *x* and *y* directions. h_s is the step size, which can be a fractional or integer number. $c(\bullet)$ is a coefficient function.

2.2 GPU implementation

The GPU based computation is an emerging

technology for these years. Due to the high complexity of the SAD algorithm and the recent use of GPUs for massively parallel computation, we decided to use the AMD HD 6850, OpenCL version 1.2 to run our GPU based algorithm. The large computational burden is mainly due to the computation of convolution for each pixel which is used to calculate the subpixel step.

Through OpenCL, we can define functions, called kernels that are executed in parallel by *N* different OpenCL threads. These threads can be grouped into blocks. The blocks can be categorized into 1D, 2D or 3D abstract shape. As the filtered images are 2D images, we chose 2D blocks for our implementation. The size of the blocks is an important parameter to computational time. Empirically, the fastest results were achieved partitioning each block into a 16x16 set of threads. OpenCL architecture has fast local memory called local memory. The access to the local memory is faster than the GPU global memory. For the HD 6850 there are 16,384 bytes/block which are simultaneously used by shared memory and registers. Because of this limitation, we should select the most important data to copy into shared memory.

To implement the GPU-based implementation, the first thing to be considered is memory transfer. As memory transfer from host to device is expensive, the best implementation is transferring the whole data to GPU before start of computation and transferring back after filtering to save memory cycles. In the filtering stage, different pixels can be simultaneously filtered as there is no dependency between any two pixels. In our implementation, we have divided the processing into three parts according the algorithm steps in above section. Figure 1 is a flowchart for the proposed method, and it shows the processing on the GPU including data transfer.

After moving all the projection data to GPU memory, separate kernel calls are launched to execute these three steps. The kernel calls are issued in a non-blocking manner but they are dependent on finishing of the previous kernel. The whole algorithm can be summarized as follows:

Step 1: Load the input image into the host memory and transfer the data to the device side

Step 2: Calculate subpixel difference of $\nabla_S I^t$ at time step *t*

This kernel is the implementation of the step 1. As there are 8 different masks for convolution representing 8 different derivative directions. In this way, we have to implement the kernel 8 times with 8 different input masks. The mask size is of 3x3. The approach to implement convolution in parallel is to load a block of the image into a shared memory array, do a point-wise multiplication of a filter-size portion of the block, and then write this sum into the output image in device memory. Each thread block processes one block in the image. Each thread generates a single output pixel.

For any reasonable filter kernel size, the pixels at the edge of the shared memory array will depend on pixels not in shared memory. Around the image block within a thread block, there is an apron of pixels of the width of the kernel radius that is required in order to filter the image block. Thus, each thread block must load the pixels to be filtered and the apron pixels into shared memory. Note: The apron of one block overlaps with adjacent blocks. The aprons of the blocks on the edges of the image extend outside the image – these pixels can either be clamped to the intensity of pixels at the image edge, or they can be set to zero.

a. Set the block size equals to best suggested number of 16x16 and global size with the number of threads equal to the number of pixels of input image.
b. Load the pixel value of each pixel from global memory to local memory of each thread.
c. Add apron pixels to four sides of each block.
d. Calculate the point-wise multiplication and store the sum into the global memory.

Step 3: Compute g_{Sx}^t and g_{Sy}^t

This step is calculating the equation (4) and (5). The nature of the computation is suitable for spread the work of each pixel to each thread

a. Set the global size with the number of threads equal to the number of pixels of input image.
b. Calculate the diffusion coefficient by

computation the transfer function, multiplication with rate parameter and subpixel matrix.

Step 4: Calculate subpixel differences $\nabla_{Sx} g_{Sx}^t$ and $\nabla_{Sy} g_{Sy}^t$.

This step is calculating the subpixel of g_{Sx} and g_{Sy}. It also uses the parallel implementation of convolution. The detailed steps are the same with Step (2).

Step 5: Update the image according to the Eq. (1)

This step is spread the update calculation of each pixel to each thread.

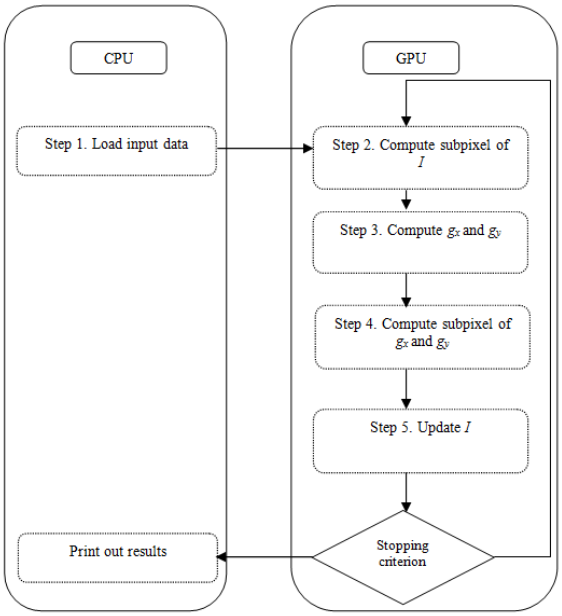

Fig. 1 Flow chart of the GPU implementation

3. Experimental Results

In this section, we present the results obtained by applying our proposed novel subpixel difference model for image denoising. We test the proposed method on same images in [8] having Gaussian noise with different standard deviations. The denoising performance is evaluated quantitatively using signal-to-noise ratio (SNR). The parameters are taken as the same parameters in [8].

We start with a serial CPU implementation. Then we parallelize the implementation by using OpenCL GPU. The CPU machine used in the experiments is an Intel i7-3820 3.60 GHz with Hyper thread and 16 GB of memory. The GPU is AMD HD 6850 that has 960 stream processors with 1 GB memory. The core clock is 775 MHz.

The "Lena" image is used to demonstrate the performance in this experiment. We scale it to different size to explore the computation different between CPU and GPU parallel implementation. The iteration number of our proposed method for this test is 100. 5 repeated runs are made to obtain the average running time for each test. In Table 1, different image size are tested to get the corresponding CPU and GPU running time. Analysis about the relationship between running time (as well as speedup) and the size of the image was conducted. As seen from Figure 2, the speedup of the GPU implementation has been affected a lot by the size of the images. Figure 2 shows the speedup of GPU, and the accelerations were increased as the size of the image increased. This is because when the size of the image is increased, it directly leads to the increase of utilization of GPU cores, which greatly enhance the speedup performance of GPU.

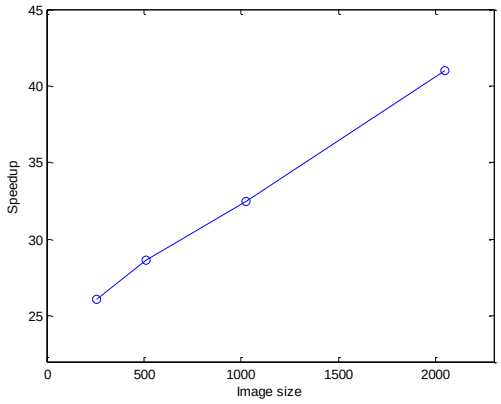

Fig. 2 Speedup of the GPU parallel implementation

The experimental results show that the proposed algorithm achieved better performance with same denoising results and less time costs. For the images with large size, our proposed method demonstrates the superiority to the old method without GPU algorithm. The GPU acceleration improved the algorithm more suitable

for real time application.

Tab. 1 Average running time of GPU parallel implementation

Image size	256x256	512x512	1024x1024	2048x2048
CPU time (s)	85.442	181.512	375.162	780.438
GPU time (s)	3.279	6.337	11.551	19.02

4. Conclusions

A denoising technique is improved using the subpixel anisotropic diffusion and GPU technology. The subpixel difference is defined, and the diffusion equation is defined using the subpixel difference. The noise is removed using diffusion procedure which is accelerated using GPU. A variety of images having noise with different levels are employed to test the performance of the proposed method. The experimental results show that the proposed algorithm yields same performance with the previous subpixel method with less computing time. The GPU acceleration improved the proposed algorithm more suitable for real time application in image processing

5. References

1. A. Buades, B. Coll and J. M. Morel (2005). A review of image denoising algorithms, with a new one, Multiscale Modeling & Simulation, vol.4, pp. 490-530.
2. Ponomaryov, Volodymyr I.; Gallegos-Funes, Francisco J.; Rosales-Silva, Alberto. (2005) Real-Time Color Imaging Based on RM-Filters for Impulsive Noise Reduction, Journal of Imaging Science and Technology, Volume 49, Number 3, pp. 205-219
3. Han Shi, Jiaxin Shao, Dong Du, Baohua Chang, and Huayong Cao. (2011) "Noise reduction of the real-time X-ray image based on modified adaptive local noise reduction filter", the 4th International Congress on Image and Signal Processing, Vol. 4, pp. 1945–1949.
4. Fernanda Palhano Xavier de Fontes, Guillermo Andrade Barroso, Pierrick Coupe, and Pierre Hellier. (2011) Real time ultrasound image denoising. Journal of Real-Time Image Processing, Vol. 6, No. 1, pp. 15-22.
5. Bart Goossens, Hiêp Luong, Jan Aelterman, Aleksandra Pižurica, and Wilfried Philips. (2010) A GPU-Accelerated Real-Time NLMeans Algorithm for Denoising Color Video Sequences", Advanced Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science, Vol. 6475, pp 46-57.
6. M. Broxvall, K. Emilsson, and P. Thunberg. (2012) Fast GPU Based Adaptive Filtering of 4D Echocardiography. IEEE Transactions on Medical Imaging, Vol. 31 , No. 6, pp. 1165 – 1172.
7. Wei Wei, Qian Du, and Nicolas H. Younan. (2012) Fast supervised hyperspectral band selection using graphics processing units. Journal of Applied Remote Sensing, vol. 6.
8. P. Perona and J. Malik. (1990) Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.12, pp. 629-639.
9. Yanhui Guo and Heng-Da Cheng. (2012) Image noise removal approach based on subpixel anisotropic diffusion. J. Electron. Imaging. Vol. 21, No. 3.
10. W. Wei, Q. Du, and N. H. Younan. (2012) Optimized spectral transformation for detection and classification of buried radioactive materials. IEEE Transactions on Nuclear Science, vol. 59.
11. Q. Du, W. Wei, D. May, and N. H. Younan. (2010) Noise-adjusted principal component analysis for buried radioactive target detection and classification. IEEE Transactions on Nuclear Science, vol. 57.
12. W. Wei, Q. Du, and N. H. Younan. (2010) Particle swarm optimization based spectral transformation for radioactive material detection and classification. Proceedings of IEEE Conference on Computational Intelligence for Measurement Systems and Applications, Taranto, Italy.