DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Internal Research Reports

ISSN

Number 30 | February 2013

A Region Growing Segmentation Algorithm for GPUs

Patrick Nigri Happ

Raul Queiroz Feitosa

Cristiana Bentes

Ricardo Farias

PUC RIO

Internal Research Reports

Number 30 | February 2013

A Region Growing Segmentation Algorithm for GPUs

Patrick Nigri Happ

Raul Queiroz Feitosa

Cristiana Bentes

Ricardo Farias

CREDITS

Publisher:
MAXWELL / LAMBDA-DEE
Sistema Maxwell / Laboratório de Automação de Museus, Bibliotecas Digitais e Arquivos
http://www.maxwell.vrac.puc-rio.br/

Organizers:
Alexandre Street de Aguiar
Delberis Araújo Lima

Cover:
Ana Cristina Costa Ribeiro

A Region Growing Segmentation Algorithm for GPUs

Patrick Nigri Happ, Raul Queiroz Feitosa, Cristiana Bentes, Ricardo Farias

Abstract—**This paper proposes a parallel region growing image segmentation algorithm for Graphics Processing Units (GPU). It is inspired in a sequential algorithm widely used by the Geographic Object Based Image Analysis (GEOBIA) community. Initially, all image pixels are considered as seeds or primitive segments. Fine grained parallel threads assigned to individual pixels merge adjacent segments iteratively following a criterion, which seeks to minimize the average heterogeneity of image segments. Beyond spectral features the merging criterion considers morphological features, which can be efficiently computed in the underlying GPU architecture. Two algorithms using different merging criteria are proposed and tested. An experimental analysis upon five different test images has shown that the parallel algorithm may run more than 19 times faster than its sequential counterpart.**

Index Terms— **Image Segmentation, Parallel Processing, Graphics Processing Unit**

I. Introduction

With the increasing availability of orbital sensors of very high spatial resolution (VHR), a group of image interpretation techniques that came to be called Geographic Object Based Image Analysis, or simply GEOBIA, is gaining importance worldwide. The first and most important step of this methodology is image segmentation. Among the methods proposed in the last four decades [1] for image segmentation, algorithms based on the region growing technique have been the most widely used by the GEOBIA community [2]. However, in practice, the high computational cost of this technique [3] allied to the volume of data to be processed in the VHR images are still demanding more efficient solutions.

The progress of the Very Large Scale Integration Technology in the last few years, made parallel computer organizations commercially available at affordable prices, so that parallel processing became the main alternative to accelerate computationally intensive applications such as image segmentation [4],[5].

Particularly, Graphics Processing Units (GPUs), that is a ubiquitous component of modern computers, offer an excellent cost-to-performance ratio. Modern GPUs use massive parallelism to provide impressive floating-point capability and to significantly improve the application performance.

Despite some previous efforts in the use of GPUs for accelerating image segmentation as in [6] and [7], the works geared to GEOBIA applications are scarce. This paper contributes to fill this gap by proposing two parallel algorithms for region growing segmentation on GPUs. They are based on different heuristics for merging adjacent segments. Their computational performances are evaluated experimentally on two distinct GPUs, and the outcome of the segmentation compared to the sequential output.

The organization of this paper is as follows. Section II briefly describes the basic architecture of a GPU. The sequential segmentation algorithm which derives this proposal is introduced in Section III. Section IV describes the parallel algorithms proposed in this paper and its variants. The experimental evaluation that aimed to assess the computational performance of parallel algorithms is reported in Section V. The article concludes summarizing the main conclusions of the work and indicating future directions.

II. GPU Architecture

This work focuses on the NVIDIA GPUs, widely available on the market. A NVIDIA GPU can be defined as a set of multithreaded Streaming Multiprocessors (SMs), each consisting of a set of Scalar Processor (SPs), the GPU cores. The memory in the GPU is organized as: a large global memory with high latency; a very fast, low latency on-chip shared memory to each SM; and a private local memory to each thread. Data communication between the GPU and the CPU is conducted via the PCIe bus. The CPU and the GPU have separate memory spaces, referred to as host memory and device memory, and the GPU-CPU transfer time is limited by the speed of the PCIe bus.

The NVIDIA programming model is CUDA (Compute Unified Device Architecture). CUDA is a C-based development environment that allows the programmer to define special C functions, called kernels, which are executed

Manuscript received December 29, 2012. This work has been supported by CNPq, and by the FP7 Program as part of the TOLOMEO Project.

P. N. Happ is Research Assistant with the Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro R. Marquês de São Vicente, 225, Gávea, Rio de Janeiro, RJ - Brazil - 22453-900 (phone: +55 3527 1626; fax: +55 3527 1232 (e-mail: patrick@ele.puc-rio.br).

R. Q. Feitosa is with the Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, and with the Department of Computer System Engineering, State University of Rio de Janeiro; (e-mail: raul@ele.puc-rio.br).

C. Bentes, is with the Department of Computer System Engineering, State University of Rio de Janeiro; (e-mail: cris@eng.uerj.br).

R. Farias is with the Department of Computer System Engineering at COPPE, Federal University of Rio de Janeiro; (e-mail: rfarias@cos.ufrj.br).

in parallel on the GPU by CUDA threads. The threads are organized into a hierarchy defined by a matrix of blocks, each performing a specified number of threads. Threads in the same block can cooperate among themselves using synchronization primitives, shared memory, and global memory. The GPU supports a great number of fine-grain threads.

Thread processing is not independent in the GPU. Threads are processed in groups called warps. Within a warp, all the threads execute the same instruction. If one thread diverges from the others, its execution is serialized. In this case, the warp has to be issued multiple times, one for each group of divergent threads, leading to performance degradation. For harnessing the parallel processing capability of the GPU it is advisable to structure the application so that the warps hold a large number of threads and the load is distributed as evenly as possible among them. Moreover, as access to global memory has high latency, it is important that the threads have fine granularity. More information about NVIDIA's GPUs and the CUDA programming model can be obtained in [8].

III. Sequential Algorithm

The parallel algorithm proposed in this paper and presented in Section IV can be regarded as a parallel implementation of Baatz and Schäpe algorithm [9]. This section presents a brief overview of this algorithm, to enable the reader to make sense of the novelties brought by the present paper.

At the beginning of the segmentation, each image pixel constitutes a seed and represents one segment. The segments are created in an iterative process. At each iteration, all segments are visited following a pseudorandom order, so as to favor a balanced growth of all segments.

When a segment is visited, the global heterogeneity increase, which would result from its fusion with each of its neighbors, is calculated. Then, the best neighbor is determined, i.e., the one whose fusion would produce the lowest heterogeneity increase. Subsequently, one of two decision heuristics is adopted. In the variant known as *best fitting* the visited segment is merged with its best neighbor, without any further concern. In the second variant, known as *local mutual best fitting*, merging occurs only if this condition is mutual, i.e., if the visited segment is also the best neighbor of its best neighbor. Notice that for both heuristics, fusion is additionally conditioned to the amount of global heterogeneity increase: its value should be lower than the square of the *scale* parameter, which determines the maximum allowed heterogeneity increase that can result from merging of two segments. The *scale* parameter indirectly influences the average size of the final segments. The segmentation procedure ends when no further merging can be performed.

As shown in (1), the measure of the heterogeneity increase (*f*), also called *fusion cost*, has a *spectral component* (h_{color}) and a *morphological component* (h_{shape}). The relative importance of each component is determined by a *mixture factor* (w_{color}). Thus, the fusion factor is given by:

$$f = w_{color}.h_{color} + (1 - w_{color}).h_{shape}. \qquad (1)$$

The *spectral component* of the *fusion cost* is given by a linear combination of the standard deviation of the pixels values within the segment in each band. The coefficients of the linear combination are user-defined parameters that determine the relative contribution of each spectral band.

In the original algorithm, the *morphological component* is a function of two features: *Compactness* (*Cmp*) and *Smoothness* (*Smt*), respectively defined in (2) and (3). The former considers the edge length *l* and the object area *n*, and is minimum for circular shapes as it can be seen in

$$Cmp = l / \sqrt{n}. \qquad (2)$$

In contrast, the latter achieves its minimum for rectangular shapes, being formulated in terms of the edge length *l* and the edge length of segment's bounding box *b* as in

$$Smt = l / \sqrt{b}. \qquad (3)$$

A further user defined parameter expresses the relative importance of *Compactness* and *Smoothness*.

IV. Parallel Algorithm

A. General Description

The central idea of the proposed algorithm is to create one thread to carry out the processing related to each image pixel, so as to take full advantage of the GPU fine-grain thread computing capacity. This approach also fosters a better load balancing, since each thread will always deals with the computation of a single pixel.

The algorithm sets up a data structure in the GPU global memory with the information about the pixels and the segments they belong to is stored. The data structure is a vector whose index identifies the pixel.

Each vector entry contains the following fields: a) the identifier of the segment which the pixel belongs b) whether or not the pixel belongs to the edge of its segment c) information necessary for calculating spectral and morphological features of the segment d) the identifier of the best neighbor segment, and e) the *fusion cost* related to its best neighbor.

The pixel called from this point forward in the text as segment maker is the one whose identifier matches the segment's identifier. The information contained in fields c) to e) is only relevant for the segment maker.

In this paper we propose two parallel segmentation algorithms: PBF (Parallel Best Fitting), based on the "*best fitting*" segmentation; and PLMBF (Parallel Local Mutual Best Fitting), derived from the "*local mutual best fitting*" heuristic. Both versions are performed by six kernels executed in parallel in the GPU (see Fig. 3) and described in the following. A more detailed description is available at [10].

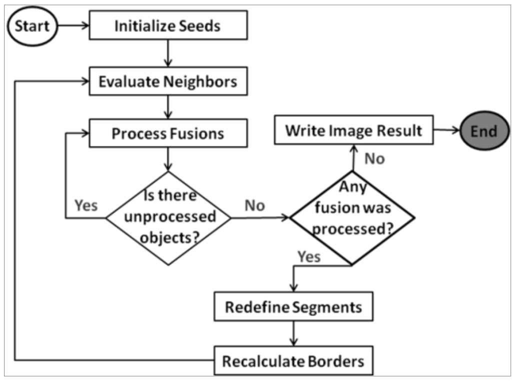

Fig. 1. Parallel segmentation algorithm diagram

1) *Initialize Seeds*: This function initializes the image pixels as seed segments. Basically, the function fills the entries corresponding to each pixel in the data structure.
2) *Evaluate Neighbors*: This function consists of two steps: a local one, comprising threads within each block, and a global one that consolidates local results. In the first step edge pixels of each segment are identified. Then, for each edge pixel the *fusion costs* relative to its adjacent segments are estimated. The identifier of the neighbor segment with the lowest *fusion cost* is stored locally in the GPU shared memory. Since this evaluation is performed by many threads in parallel, there may be conflicting identifiers. Such conflicts are solved within each block at the end of the first step. In the second step, existing conflicts between different blocks are solved, obtaining a unique identification of each segment's best neighbor. The best neighbor's identifier and the *fusion cost* of the segment makers are updated in the GPU global memory within a critical section.
3) *Process Fusions*: All segment makers whose *fusion cost* is lower than the allowed maximum (the square of the *scale* parameter) are selected for merge. A critical section avoids multiple simultaneous processing of a single segment. Basically, one segment maker of each pair of segments being merged is elected to be the maker of the new created segment, whereas the other maker becomes a regular pixel of that new segment.
4) *Redefine Segments*: After a fusion, the other pixels of the merged segments must be identified as belonging to the resultant segment. This task is performed in parallel by the *Redefine Segments* function. It assigns to these pixels the new segment identifier.
5) *Recalculate Borders*: Pixels along the common border of two segments being merged will not belong to the border of the resultant segment. This function checks for each pixel at the segment border if at least one of its adjacent pixels belongs to another segment. If this is not the case, the data structure is updated to indicate that these pixels are no longer part of the segment border.
6) *Write Image Result*: When no fusion can be performed, the segmentation result is generated and the algorithm is terminated.

Both the PBF and PLMBF algorithms basically use the same described functions with small changes. The key differences lie in the *Process Fusions* function. In PLMBF after having selected the best neighbor, *Process Fusions* additionally checks if the mutuality condition holds. If not, no fusion takes place. Furthermore, the PLMBF algorithm incorporates a mechanism that deals with situations in which multiple segments have the same *fusion cost*.

B. *Morphological Features*

The features defined in (2) and (3) must be computed whenever two adjacent segments are being considered for merging. It involves determining the edge length of the new segment that will result if the fusion actually occurs. Besides being computationally expensive, this operation implies in a large number of accesses to the data structure stored in the GPU global memory, whose latency is high. Moreover, it may lead to an unbalance load among the threads, since the associated computational load is proportional to the border length of each segment.

In order to avoid this problem, we propose two different *morphological features*, presented in (4) and (5), whose computation does not involve a prior computation of the edge length, to substitute *Compactness* and *Smoothness*.

The first morphological feature is defined as

$$Comp = d\,\mathrm{max}/\sqrt{4n/\pi}, \tag{4}$$

where *d*max is the axis of the ellipse with identical second order moment and n is the segment area. It is worth noting that this feature is also called *Compactness* (*Comp*) by John Russ in [11], possibly because it also reaches its minimum for circular shapes.

Second, we propose the replacement of the *Smoothness* by the *Solidity* (*Sol*) as defined in

$$Sol = n / nbox, \tag{5}$$

where *n* is the segment area and *nbox* is the area of its bounding box. Note that the *Solidity* is sensitive to segment's convexity and is minimum for rectangular shapes.

V. Experimental Analysis

A. *Speedups*

The purpose of the experiments reported in this section was to determine the speedup achieved by both proposed segmentation algorithms. The experiments were performed in two different configurations of GPUs, CPUs and operating systems. The first configuration represents a low-cost environment usually present in regular computers at the time this paper is published, while the second configuration represents a more sophisticated GPU environment based on the Fermi architecture for high-performance computing:

- GT 9600: GPU: NVIDIA GeForce 9600 GT with 64 cores and 1GB of memory; CPU: Intel Core 2 6300 @ 1.86GHz and 3.25GB of RAM; OS: Windows XP.

- GTX 480: GPU: NVIDIA GeForce GTX 480 with 480 cores and 1,5GB of memory; CPU: Intel i7 970 @ 3.2GHz and 16 GB of RAM; OS: Ubuntu 10 64 bits.

The experiments were conducted on five test images, labeled from IM1 to IM5, with distinct sizes in pixel (800^2, 1000^2, 1400x1500, 2000^2, and 2400^2) and different sensors (IKONOS, Quickbird and Aerial). Fig. 2 and Fig. 3 display, respectively, the speedups obtained by the PBF and PLMBF algorithms relative to their respective sequential version measured in the GT 9600 configuration. The speedup achieved by PBF ranged from 4.3 to 6.2, whereas PLMBF attained speedups between 3.9 and 8.5.

Fig. 4 and Fig. 5 show the speedups of PBF and PLMBF, for different scale values, relative to the corresponding sequential version running on the GTX 480 configuration. PBF reached speedups between 9.0 and 14.6, whereas the speedup of PLMB ranged from 10.2 to 19.0.

Fig. 2 to 5 further reveal that the speedups of both algorithms tend to lessen as the *scale* increases. As described in Section IV, each image pixel is assigned to a GPU core when the segmentation starts. For a segment comprising multiple pixels, the cores in charge of the border pixels and the so called segment makers carry out most of the computation related to that segment. The cores assigned to the other pixels of the same segment do comparatively less work. In consequence, the load tends to concentrate on fewer cores as segments grow and the speedup diminishes. Hence, speedups decreases with *scale* as *scale* is directly related to the size of the segments obtained in the end of the segmentation procedure.

The results in both configurations show a superiority of PLMBF over the PBF in terms of the achieved speedups. This is due to the fact that “*local mutual best fitting*” algorithm is more complex than “*best fitting*”. So, PLMBF involves more computation than PBF not only in absolute terms but also relative to the time spent transferring data between CPU and GPU.

Moreover, the experiments did not detect a substantial dependence of the speedup on the input image. Indeed, as the scale increases the sensitiveness of the speedup to the input image became even less significant.

Finally, it is worth remarking that, despite achieving accelerations of up to 19.0, these were far below the number of processors available in the GPU. Since threads are assigned to segments and the region growing paradigm requires the inspection of relations between adjacent segments for each merging operation, threads’ dependency is considerable. This evidently hinders the full exploitation of the parallel processing capability available in the GPU hardware.

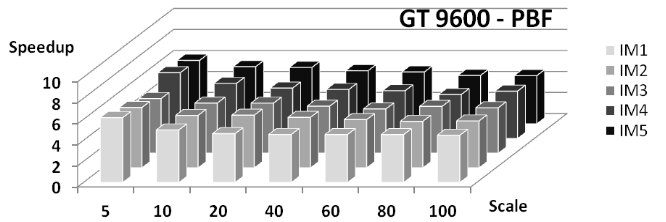

Fig. 2. Chart of speedup as a function of scale for the PBF algorithm regarding its sequential version using GT_9600

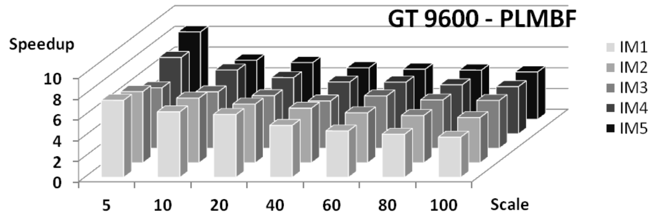

Fig. 3. Chart of speedup as a function of scale for the PLMBF algorithm regarding its sequential version using GT_9600

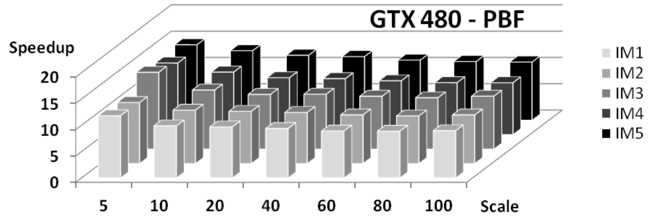

Fig. 4. Chart of speedup as a function of scale for the PBF algorithm regarding its sequential version using GTX_480

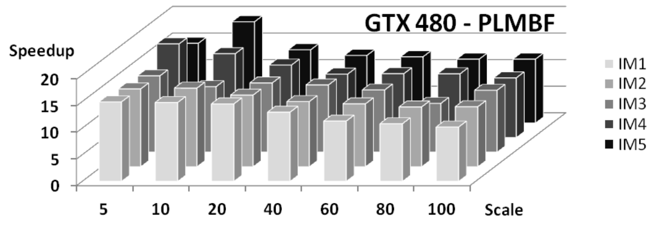

Fig. 5. Chart of speedup as a function of scale for PLMBF algorithm regarding its sequential version using GTX_480

B. Segmentation Outcome

The outcomes produced by the parallel algorithm and by its original sequential counterpart may differ for two main reasons: first, because different features are used to measure morphological heterogeneity and, second, due to different segments’ processing orders.

The difference caused by the use of distinct morphological features can be reduced by a proper adaptation of segmentation parameters. In respect to the processing order, it affects the sequential as well as the parallel version. By using a preset processing sequence, the sequential algorithm will produce always the same outcome every time it is executed for a given input image and parameter setting. However, whatever it may be, the adopted sequence will be always arbitrary in the sense that other plausible sequences may exist, which produce different but equally good and possibly even better outcomes.

In contrast, parallel region growing algorithms will generally produce different results each time it is executed.

This happens because threads' processing times is not constant every time they are executed, what affects the segments' processing order and consequently the final segmentation outcome.

This experiment aims at demonstrating that the outcomes produced by the proposed parallel algorithm diverge from a result yielded by the sequential counterpart as much as sequential algorithm's outcomes running with different segments' processing orders may differ from each other.

The experiment was conducted as follows. First, the sequential algorithm working with the original morphological features was applied to one of the test images and a set of segments from its outcome were taken as reference for comparison purpose. Subsequently, parameters of the parallel version were tuned in order to obtain a result as coherent as possible with the references. A stochastic optimization method similar to the one described in [12] was applied for that purpose. Specifically, parameters of the parallel algorithm were adapted to minimize the dissimilarity between its outcome and the selected reference segments. The Reference Bounded Segment Booster (RBSB) [12] was used as dissimilarity metric.

Then, segments' processing order of the sequential version was changed by using a simple trick. The input image was first mirrored vertically and then segmented. The outcome was mirrored back and the result compared with the references through the RBSB. The same procedure was performed with a horizontally mirrored image. The worst (highest) RBSB score among the tested processing orders was kept.

This experiment was performed 50 times for PBF, each time changing either the input image and/or the reference segments. In every instance the RBSB score of PBF was inferior (better) to the worst score obtained by changing the processing order of the sequential algorithm and in 90% of cases PBF achieved the best score among all. The average RBSB score was 67% and 6% respectively for the sequential algorithm in the worst case and for the PBF parallel algorithm. These values indicate that the variability of the outcome due to changes in processing order of the sequential algorithm is mostly higher than the difference of the outcome produced by the parallel algorithm.

Conversely, for PLMBF the RBSB score was superior (worse) in 56% of cases to the worst score recorded for the sequential version working with distinct processing orders. However, the average scores computed on 50 experiments were 7% and 4% respectively for the sequential algorithm in worst case and for the PLMBF algorithm. Recall that a RBSB score equal to 4%, means that the outcomes being compared agreed in average in 96% of the pixels comprised in the reference segments.

VI. Conclusions

This paper proposed a novel parallel algorithm for image segmentation inspired in a sequential version widely used by the remote sensing community. In the proposed algorithm a heterogeneity criterion that controls the region growth is formulated in terms of both spectral and morphological features.

Two variants of the parallel algorithm were described, each one characterized by a different decision heuristic for merging segments. In both variants, each pixel is processed by a distinct thread in order to exploit the parallel processing capability provided by GPUs.

The acceleration of both proposed variants relative to their sequential counterparts achieved in our experimental analysis values up to 14.6 and 19.0.

It has also been found experimentally that the segmentation outcome of the parallel algorithms is generally similar to the outcome produced by their sequential counterpart.

An implementation of both parallel algorithms as well as their sequential versions, are available exclusively for educational and research purposes at the following address: http://www.lvc.ele.puc-rio.br/wp/?cat=41.

References

[1] W. K. Pratt, *Digital Image Processing (4th Edition)*. Wiley-Interscience, 2007, pp. 579-622.

[2] G. Meinel and M. Neubert, “A Comparison of segmentation programs for high resolution remote sensing data”, *Proceedings XXth ISPRS Congress*, Istanbul, 14-23. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXV-B4, p. 1097-1102, July 2004.

[3] J. Wassenberg, W. Middelmannand, and P. Sanders, “An efficient parallel algorithm for graph-based image segmentation”, in *CAIP '09: Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, pp. 1003-1010, Berlin, Heidelberg. Springer-Verlag.

[4] P. Lenkiewicz, M. Pereira, M. M. Freire, and J. Fernandes, “A new 3D image segmentation method for parallel architectures”, Multimedia and Expo, 2009. ICME 2009. *IEEE International Conference on*, vol., no., pp.1813-1816, June 28 2009-July 3, 2009.

[5] P. N. Happ, R. S. Ferreira, C. Bentes, G. A. O. P. Costa, and R. Q. Feitosa, “Multiresolution Segmentation: a Parallel Approach for High Resolution Image Segmentation in Multicore Architectures”, in: *3rd International Conference on Geographic Object-Based Image Analysis*, 2010, Ghent, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Enshede: ITC, 2010.

[6] S. Schenke, B. Wuensche, and J. Denzler, “GPU-based volume segmentation”, in *Proc. of IVCNZ '05* (2005), pp. 171 – 176.

[7] L. Pan, L. Gu, and J. Xu, “Implementation of medical image segmentation in CUDA”, Information Technology and Applications in Biomedicine, 2008. ITAB 2008. *International Conference* on , vol., no., pp.82-85, 30-31 maio 2008.

[8] NVIDIA, “CUDA C Programming Guide”, v.5.0, Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.

[9] M. Baatz and A. Schäpe, “Multiresolution segmentation: an optimization approach for high quality multi-scale image segmentation”, in: *XII Angewandte Geographische Informationsverarbeitung*, Wichmann-Verlag, Heidelberg, 2000.

[10] P. N. Happ, “Image segmentation on GPUs: a parallel approach to region growing”, MSc. Dissertation, Rio de Janeiro, Brasil: Pontifical Catholic University of Rio de Janeiro, 2011.

[11] J. C. Russ, *The Image Processing Handbook* (*3rd Edition*). Materials Science and Engineering Department North Carolina State University Raleigh - North Carolina, 1998, pp. 553.

[12] R. Q. Feitosa, G. A. O. P. Costa, T. B. Cazes, and B. Feijó, “A genetic approach for the automatic adaptation of segmentation parameters”, *International Conference on Object-based Image Analysis* – ISPRS Proceedings, v. 36, n. 4/C42, 2006.