A PARALLEL IMAGE SEGMENTATION ALGORITHM ON GPUS

P. N. Happ [a,*], R. Q. Feitosa [a], C. Bentes [b], R. Farias [c]

[a] Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225, Gávea, CEP 22451-900, Rio de Janeiro, RJ, Brazil
{patrick, raul}@ele.puc-rio.br
[b] Dept. of Computer and Systems, Rio de Janeiro State University
Rua São Francisco Xavier 524, Maracanã, CEP 20550-900, Rio de Janeiro, RJ, Brazil
cris@eng.uerj.br
[c] Federal University of Rio de Janeiro
P.O. Box: 6851, CEP 21945-970, Rio de Janeiro, RJ, Brazil
rfarias@cos.ufrj.br

KEY WORDS: Image Segmentation, Parallel Processing, GPU

ABSTRACT:

Image segmentation is a computationally expensive task that continuously presents performance challenges due to the increasing volume of available high resolution remote sensing images. Nowadays, Graphics Processing Units (GPUs) are emerging as an attractive computing platform for general purpose computations due to their extremely high floating-point processing performance and their comparatively low cost. In the image analysis context, the use of GPUs can accelerate the segmentation process. This work presents a parallel implementation of a region growing algorithm for GPUs. The parallel algorithm is based on processing each pixel as a different thread so as to take advantage of the fine-grain parallel capability of the GPU. In addition to the parallel algorithm, the paper also suggests a modification to the heterogeneity computation that improves the segmentation performance. The experiments results demonstrate that the parallel algorithm achieve significant performance gains, running up to 6.8 times faster than the sequential approach.

1. INTRODUCTION

Image segmentation has been the subject of extensive research in the areas of digital image processing and computational vision. The segmentation process plays a key role in the image analysis process (Blaschke and Strobl, 2001), and many segmentation methods have been proposed in the literature (Riseman and Arbib, 1977; Fu and Mui, 1981; Haralick and Shapiro, 1985; Pal and Pal, 1993; Deb, 2008) together with metrics for quality assessment (Zhang, 1996; Correa and Pereira, 2000; Cardoso and Corte-Real, 2005; Zhang et al., 2008). Among the image segmentation methods, the region growing algorithm is one of the best known and the most widely used in the remote sensing area (Tilton and Lawrence, 2000).

Region growing algorithms group pixels or sub-regions in larger regions on an iterative way. The process starts with a set of initial points, called seeds, that grows by merging adjacent regions that contains similar properties such texture or color. However, this segmentation technique is computationally expensive when large images are considered (Wassenberg et al., 2009). In addition, region growing usually has some parameters that must be adjusted for each type of application, which implies in a number of executions until the optimal parameter values are found. Thus, the execution time of the segmentation is decisive for its operational use in automatic image interpretation systems. For this reason, computational acceleration is highly required.

Recent advances in the hardware architecture and programmability of Graphics Processing Units (GPUs) have turned them into an attractive platform for accelerating general purpose floating-point computations. They offer promising speedups, are available off-the-shelf, and it is likely that most computers will be equipped with such devices in the future. Modern GPUs can achieve performance of at least one order of magnitude higher compared to that of the traditional CPUs. However, the problem is how to program these devices efficiently. Parallelizing the algorithm to fit the highly parallel architecture of the GPU can be a challenging task.

Several GPU implementations of image segmentation methods have been proposed in the literature. Some of them were built on the facility of implementing the evaluation of partial differential equations in a stream processing model (Sherbondy et al., 2003; Lefohn et al., 2003). There are also some research efforts in the area of medical imaging (Ruiz et al., 2008; Erdt et al., 2008; Ahn et al., 2005; Unger et al., 2008; Pan et al., 2008). The particular case of satellites images has to be pointed out. Sun et al. (2009) implemented a parallel segmentation method in GPU for remote sensing images based on the clustering Mean Shift algorithm. Their approach starts from selected seeds and clusters the pixels near the seeds. The center of each cluster is computed and the regions grow from these centers. This two step method implies in a pixel independent parallel implementation that provided a speedup around 20 for IKONOS and Quickbird images. Nevertheless, as far as we know, there is no GPU implementation of unseed region growing algorithm.

* Corresponding author.

Algorithms that consider every pixel as a seed pose an extra difficulty to the parallel implementation due to the large number of processes/threads and the synchronization required among them. In (Happ et al., 2010), we suggest a parallel strategy for multicore architectures that deals with unseeded region growing using a tile image division approach. In this paper, we propose a different parallelization scheme that takes benefit of the highly parallel architecture of the GPU. Instead of dividing the image into tiles, each pixel is processed by a different thread. Also, two new attributes for calculating spatial heterogeneity are presented in order to maximize computational efficiency. The algorithm is implemented using the programming languages C and CUDA and the computational performance is evaluated for a given set of remote sensing images.

The organization of this paper is as follows. In Section 2, the GPU architecture is briefly described. In Section 3, the sequential region growing is depicted. In Section 4, the parallel algorithm is exposed. In Section 5, comparisons between the serial and parallel algorithms are presented. In the last section, the conclusions are presented.

2. GPU ARCHITECTURE

Modern GPUs are massively parallel processors that support a great number of fine-grain threads. They are especially well-suited to explore computations on many data elements that have high arithmetic intensity. The GPU architecture is composed of a scalable array of so-called streaming multiprocessors. One such multiprocessor contains amongst others a number of scalar processor cores, a multi-threaded instruction unit, a number of registers and a shared memory. The number of multiprocessors and processor cores depends of the architecture and model of the GPU.

CUDA (NVidia, 2010) is the NVidia C-based development environment for GPUs, that includes a parallel programming model and an instruction set architecture. CUDA allows the programmer to define special C functions, called kernels, which are executed in parallel by different CUDA threads. The programmer organizes these threads into a hierarchy of grids of thread blocks. A thread block is a set of concurrent threads that can cooperate among themselves through barrier synchronization and shared accesses. During execution, the threads can access data at different levels of hierarchy: registers, shared memory and global memory. The global memory is accessible by all threads, but its access time is about 500 times slower than the access time to shared memory and registers.

Thread processing is not independent on the GPU. Threads are executed in groups called warps. Within a warp, all the threads execute the same instruction. If one thread diverges from the others, there is performance degradation, since this thread starts to operate singly while the remaining are disabled.

3. REGION GROWING ALGORITHM

As we focus in remote sensing applications, we choose a popular region growing algorithm, proposed originally by Baatz and Schäpe (2000), as the basis of our parallel implementation. This method was considered as one of the most effective segmentation algorithms (Neubert and Meinel, 2003). Furthermore, variants of this algorithm are available as operators on the InterIMAGE platform (InterIMAGE, 2012) and on the Definiens system (Definiens, 2008).

This algorithm consists of an iterative method that seeks to minimize the average heterogeneity of the image objects. All image pixels are first considered as seeds or initial segments and, at each step, the heterogeneity increase is calculated as a result from merging two adjacent segments. This value is given by a fusion cost that must be below a given threshold to enable merging both segments into a single one. The process is repeated until no merge is possible.

The fusion cost (*f*) represented by Equation 1, is defined by a weighted sum between a component related to spectral heterogeneity (h_{color}) and another referred to spatial heterogeneity (h_{shape}). The importance of these components is defined by a relative weight between color and shape (w_{color}) and for both heterogeneity components the formula is based on the difference between the merged object (obj_3) and the sum of the separated objects (obj_1 and obj_2) as it can be seen in Equation 2.

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

$$h_x = h_{obj3} - (h_{obj1} + h_{obj2}) \tag{2}$$

Spectral heterogeneity (h_{color}) is given by the standard deviation of each pixel value, considering each color band separately and given a different weight for each band. On the other hand, spatial heterogeneity (h_{shape}) is composed by two different shape components: one related to compactness and another related to smoothness. Compactness (*Cmp*), formulated in Equation 3, is given by the ratio between the edge length (*l*) and the square root of the object area (*n*). Smoothness (*Smt*), as seen in Equation 4, refers to the ratio between the object edge length (*l*) and the edge length of its bounding box (*b*) It is worth to note that there is also a weight to manage the importance between compactness and smoothness on the composition of the spatial heterogeneity.

$$Cmp = \frac{l}{\sqrt{n}} \tag{3}$$

$$Smt = \frac{l}{\sqrt{b}} \tag{4}$$

The algorithm has, therefore, an adjustable heterogeneity criterion. Parameters such as the relevance of each spectral band and the relative importance of shape and color and between compactness and smoothness can be tuned in order to achieve a better segmentation result. A final parameter called *scale*, which defines the maximum admissible fusion cost directly influences the size of the generated objects.

4. PARALLEL ALGORITHM

The fundamental characteristic of the GPU architecture is that it has a highly parallel architecture that supports a great number of fine-grain threads. In this way, the proposed parallel algorithm assigns the processing of each pixel of the image to each thread. This parallelization scheme exploits the massive computational capacity of the GPU and also provides a good load balancing, since each thread deals with the same amount of computation. Another advantage of this parallelization scheme is the ability to process every image segment directly, without dividing the image into tiles and having to deal with bordering issues.

A data structure is created on the GPU global memory for each image pixel to store information about the pixel and the segment it belongs to, as shown in Figure 1. This structure is organized in a vector whose indexes represent the pixel identifiers (pixel *Id*). The structure holds the following information: a) the segment identifier (segment *Id*) the pixel belongs to; b) if the pixel is part of the segment border; c) the previous and next pixel of that segment; d) segment area; e) segment spectral and spatial attributes; f) the *Id* of its best neighbor segment, as will be later explained; and g) the fusion cost. The pixel called hereafter *segment maker* is the one, whose *Id* coincides with the *Id* of its segment. The information from d) to g) is only relevant for *segment makers*.

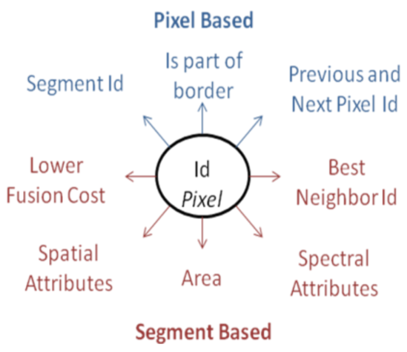

Figure 1. Data structure

The parallel algorithm consists of six kernels to be executed by the GPU (see Figure 2) described below:

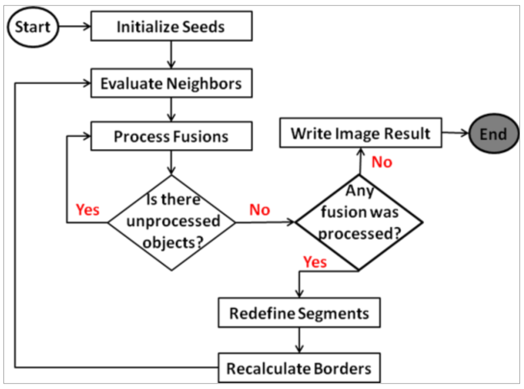

Figure 2. Diagram of the parallel segmentation algorithm

Initialize Seeds
The function *Initialize Seeds* marks each image pixel as a seed, which will represent an initial segment. Then, the attributes of these pixels are computed and stored in a specific data structure (see Figure 1).

Evaluate Neighbors
The function *Evaluate Neighbors* first detects the pixels over segment borders. The fusion cost of the adjacent segments to each of those pixels is then calculated. That one representing the lowest cost for a pixel is denoted as its best neighbor in the shared memory. As the processing is pixel based and a segment can have lots of border pixels, there must be a comparison between the local result (by pixel) and the global result (by segment). This task is executed inside a critical section to update the *Id* of the best neighbor and the fusion cost at the segment former structure on GPU global memory.

Process Fusions
If any of the *segment makers* has the fusion cost lower than the maximum fusion cost (a given threshold), it is selected to grow. A critical section should then be created to avoid information overlapping like two threads attempting to merge its segment with the same adjacent segment. The fusion itself then occurs, updating the attributes of the segment chosen to grow by merging it with its best neighbor. Consequently, the adjacent segment is no longer considered as a valid segment and its representing pixel (*segment maker*) is included in the merged segment. It should be noted that in order to avoid performance loss, we created a mechanism to perform control over the waiting threads on the critical section. Therefore, when a thread is waiting, its pixel is marked as "unprocessed" and the thread is aborted. To ensure the execution of every pixel, the function *Process Fusions* is called repeatedly until there are no pixels marked as "unprocessed".

Redefine Segments
This function is responsible for updating the pixels belonging to segments that were merged with their new segment identifier - the one from the segment which have encompassed them. This function should be performed only if a fusion has occurred in the previous function.

Recalculate Borders
This function aims at excluding from processing those pixels no longer lying on any segment border. Thus, for each border pixel it is checked whether at least one of its adjacent pixels belongs to a different segment. Otherwise, the pixels are no more part of the edge.

Write Image Result
When no more merging is possible, the algorithm is finished by the *Write Image Result* function. Each pixel is processed in parallel writing the average of each spectral band of every segment on a resulting image. It is worth to mention that the segments borders are printed with a particular given color.

Spatial Attributes

The features defined in equations (3) and (4) must be recalculated whenever two segments are merged. It requires the calculation of the border length of the new segment. This can be performed by adding the border lengths of both segments being merged and then subtracting the pixels on the common border. The execution time of this operation increases as segments grow, and may become computationally expensive. Since this operation must visit each border pixel, it may involve a large number of accesses to the structure of the segments stored in the high latency GPU global memory. In addition, this operation leads to load imbalance as the involved processing effort is proportional to the border length of each segment.

In order to circumvent this problem, we propose the replacement of these features by other ones, which do not involve the border length computation and are semantically equivalent, in the sense they reach their minimum for similar shapes. Inspired by the discussion of Russ (1998) on shape features, we propose a new definition for compactness (*Comp*) and a new feature called solidity (*Sol*) to describe shape in the region growing algorithm described in section 3. The former, as the feature in equation (3), diminishes as the object becomes more compact. It is defined in Equation 5, where *n* is the object area and *dmax* is the diameter of the adjusted ellipse around the object. The latter, like the smoothness, varies according to object´s convexity and involves only the object area *n* and the area of its bounding box *nbox* as defined in Equation 6.

$$Comp = \frac{\sqrt{\frac{4n}{\pi}}}{d\max} \qquad (5)$$

$$Sol = \frac{n}{nbox} \qquad (6)$$

5. EXPERIMENTAL ANALYSIS

5.1 Impact of the new shape features on the segmentation outcome

The goal of these experiments was to test if the segmentation results obtained by formulating heterogeneity in terms of the features proposed in equations (5) and (6) may be similar to the outcome obtained with the original formulation by a proper adjustment of the segmentation parameters.

Using a crop of a QuickBird image we first delineated manually three sets of segments to represent three distinct reference segmentation outcomes. Next, applying the approach proposed in (Costa et al., 2008) a genetic algorithm searched the parameter space for the set of values that optimized the level of agreement between the reference and the segmentation outcome. This experiment was performed for each set of references and for both variants of the segmentation algorithm.

Table 1 shows the dissimilarity between references and outcomes as measured by the RBSB function (Reference Bounded Segments Booster) (Costa et al., 2008) in each experiment. The values in the same column, which correspond to the same set of segment references, did not differ substantially between both heterogeneity formulations.

Attributes	Disparity values according to references		
	Homogeneous	Heterogeneous	Mixed
Cmp & Svd	0.14	0.56	0.46
Comp & Sol	0.16	0.57	0.40

Table 1. Disparity values according to references

Fig. 3 provides a visual perception of the segmentation differences in each case for both pairs of shape features. Figures (a) to (c) refer to the use of the original features, while Figures (d) to (f) are related to the use of the proposed features. Experiments have shown that these differences are comparable to what is observed when two slightly displaced crops of the same image are used for testing. This changes the order the region growing procedure visits each pixel, so that the segmentation outcomes differ quite in the same amount as can be seen in each column in Figure 3.

This evidences, that both proposed shape features are nearly equivalent to the original ones as far as the segmentation result is concerned, provided that the segmentation parameters are properly tuned.

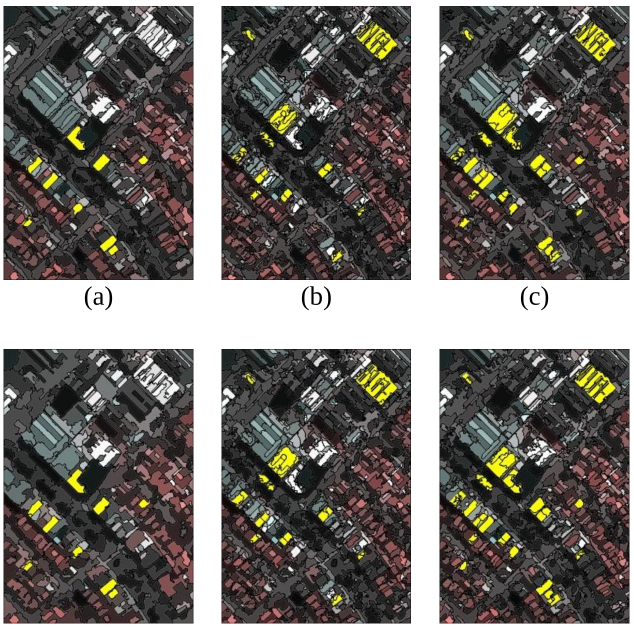

Figure 3. Segmentation result using C*mp* and *Svd* for homogeneous (a), heterogeneous (b) and mixed (c) objects and using *Comp* and *Sol* for homogeneous (d), heterogeneous (e) and mixed (f) objects.

5.2 Speedup of the parallel version

A second sequence of experiments had the objective to assess the performance gains in terms of execution time. The experiments were performed in two different environments using two GPUs, different processors (CPU), and operating systems (OS):

GT_9600

GPU: NVidia GeForce 9600 GT with 64 cores and 1GB of memory

CPU: Intel Core 2 6300 processor @ 1.86 GHz and 3.25GB of RAM.

OS: Windows XP.

Tesla:

GPU: NVidia Tesla C1060 with 240 cores and 4GB of memory

CPU: Intel Xeon @ 2.5GHz and 7.8 GB of RAM,

OS: GNU/Linux.

For the experiments, we used six clippings from an aerial image of Jardim Tropical in Resende, Brazil to generate images of different sizes as shown in Table 2.

Label	Rows	Cols	Pixels
Resende_500	500	500	250000
Resende _1000	1000	1000	1000000
Resende_1500	1500	1500	2250000
Resende_2000	2000	2000	4000000
Resende _2500	2500	2500	6250000
Resende_2800	2800	2800	7840000

Table 2. Image clippings and their respective sizes.

Figures 4 and 5 shows the speedups obtained for all the images running respectively on the environments GT_9600 and Tesla. The speedup of a parallel algorithm refers to how much it is faster than the corresponding sequential algorithm. The speedups reported here are computed as the sequential execution time divided by the parallel execution time.

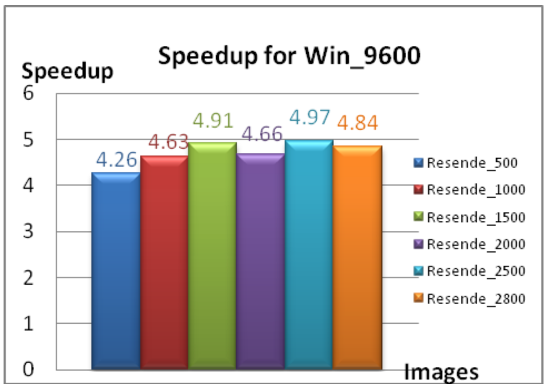

Figure 4. Speedup for Win_9600 environment

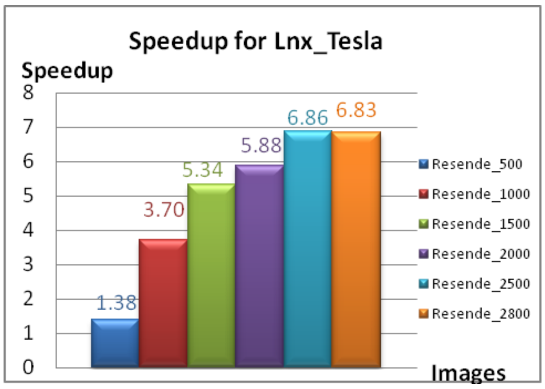

Figure 5. Speedup for Lnx_Tesla environment

Figure 4 shows that the speedups obtained by the GT_9600 execution were always above 4, reaching almost 5 in some cases. Although the images are different in terms of the number of pixels to process, the speedup obtained present small variations when the size of the image increases. In this environment, the GPU used is quite limited and has only 64 cores, which is underestimated for the images sizes. So the amount of parallelism obtained for all the images are almost the same. The difference in the performance results comes from the difference in the regions found in each image.

Figure 5 shows that when a more powerful GPU is used, it is possible to observe the benefits of the parallelization when the image size grows. The speedup obtained for Resende_500 and Resende_1000 were relatively low. This occurs because of the underutilization of the GPU, since the relative cost of data transfer from CPU to GPU and the GPU memory accesses are considered relevant when comparing to the processing time. From Resende_1500 on, the speedups are greater than 5 reaching 6.86 for Resende_2500.

It is worth noting that despite achieving speedups up to 6.86 times compared to the sequential version, these results were well below to the number of cores present in the GPUs used. This occurs due to the characteristics of the original segmentation algorithm that implies in a strong dependency among the threads and the need to constant access to GPU global memory.

6. CONCLUSION

The main objective of this work was to propose and to evaluate the performance of a parallel region growing segmentation algorithm that takes benefit of the highly parallel architecture of the GPU. The parallel algorithm essentially assigns a particular thread to each image pixel so as to exploit the GPU support of fine-grain threads and the large number of processing elements available. The heterogeneity criterion that controls the region growing is formulated in terms of both spectral and spatial features of the segments. Two different shape features were proposed to replace the ones introduced in the original sequential algorithm in order to provide a more efficient use of the GPU architecture.

Experiments have demonstrated that the proposed shape features do not imply in a significant change of the segmentation results, as long as the algorithm's parameters are properly adjusted. Moreover, experiments for performance evaluation indicated the potential of using GPUs to accelerate this kind of application. For a simple hardware (GeForce 9600 GT), the parallel algorithm reached a maximum speedup of 4.97 and with a more powerful GPU (Tesla C1060) an acceleration of 6.86 was achieved. Considering that segmentation is responsible for a significant portion of the execution time in many image analysis applications, especially in object-oriented analysis of remote sensing images, the experimentally observed acceleration values are significant. It should also be noted that these performance gains can be obtained with low investment in hardware, as GPUs with increasing processing power are currently available on the market at declining prices.

ACKNOWLEDGEMENTS

This study is part of the MSc-Thesis from the first author. He acknowledges the support given by the Pontifical Catholic University of Rio de Janeiro (PUC-Rio, Brazil) and the National Council for Scientific and Technological Development (CNPq, Brazil) for supplying his Masters scholarship.

REFERENCES

Ahn, I., Lehr, M. and Turner, P., 2005. Image processing on the gpu. *Technical report*, *University of Pennsylvania*, February 2005.

Blaschke, T. and Strobl, J., 2001. What is wrong with pixels? Some recent developments interfacing remote sensing and GIS, *GIS - Zeitschrift für Geoinformationssysteme*, n. 6, pp. 12-17.

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

Cardoso, J. S. and Corte-Real, L., 2005. Toward a generic evaluation of image segmentation, *Image Processing*, IEEE Transactions on, vol. 14 (11), pp. 1773-1782.

Correia, P. and Pereira, F., 2000. Objective evaluation of relative segmentation quality, *Image Processing*, Proceedings. 2000 International Conference on, vol.1, no., pp.308-311.

Costa, G. A. O. P., Feitosa, R. Q., Cazes, T. B. and Feijó, B., 2008. Genetic Adaptation of Segmentation Parameters. In: Blaschke, T., Lang, S. and Hay, G. (Eds.). *Object-Based Image Analysis: Spatial concepts for knowledge-driven remote sensing applications*. Heidelberg: Springer, 2008. pp. 679-695.

Deb, S., 2008. Overview of image segmentation techniques and searching for future directions of research in content-based image retrieval, *Ubi-Media Computing, 2008 First IEEE International Conference on* , vol., no., pp.184-189.

Definiens, 2008. Image Analysis Software for Earth Sciences, http://www.definiens.com/imageanalysis-for-earthsciences_45_7_9.html (acessed nov. 2008).

Erdt, M., Raspe, M. and Suehling, M., 2008. Automatic hepatic vessel segmentation using graphics hardware. In *MIAR '08: Proceedings of the 4th international workshop on Medical Imaging and Augmented Reality*, pages 403–412.

Fu, K. S. and Mui, J. K., 1981. A survey on image segmentation, *Pattern Recognition*, vol. 13, Issue 1, pp. 3–16.

Happ, P. N., Ferreira, R. S., Bentes, C., Costa, G. A. O. P, and Feitosa, R. Q., 2010. 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. v.XXXVII.

Haralick, R. M. and Shapiro, L. G., 1985. Image Segmentation Techniques, *Computer Vision Graphics, and Image Processing*, vol. 29, Issue 1, pp. 100–132.

InterIMAGE, 2010. An open source knowledge based framework for automatic image interpretation, http://www.lvc.ele.puc-rio.br/projects/interimage/index.html (acessed jan. 2012).

Lefohn, A., Cates, J. and Whitaker, R., 2003. Interactive, gpu-based level sets for 3d brain tumor segmentation. In *Medical Image Computing and Computer Assisted Intervention*, pages 564–572.

Neubert, M. and Meinel, G., 2003. Evaluation of segmentation programs for high resolution remote sensing applications. In: Schroeder, M., Jacobsen, K., Heipke, C. (Eds.). *Proceedings Joint ISPRS/EARSeL Workshop "High Resolution Mapping from Space 2003"*, Hannover, Germany, October 6-8.

NVidia, 2010. NVIDIA CUDA C ProgrammingGuide,v3.2, http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf (accessed jan. 2012).

Pal, N. R. and Pal, S. K. A., 1993. A review of image segmentation techniques, *Pattern Recognition*, vol. 26, Issue 9, pp. 1277-1294.

Pan, L., Gu, L. and Xu, J., 2008. Implementation of medical image segmentation in CUDA, *Information Technology and Applications in Biomedicine, ITAB 2008.* International Conference on , vol., no., pp.82-85.

Riseman, E. M. and Arbib, M. A., 1977. Computational Techniques in the Visual Segmentation of Static Scenes, *Computer Graphics and Image Processing*, vol. 6, Issue 3, pp. 221–276.

Ruiz, A., Kong, J., Ujaldon, M., Boyer, K. L., Saltz, J. H. and Gurcan, M. N., 2008. Pathological image segmentation for neuroblastoma using the gpu. In *ISBI*, pages 296–299.

Russ, J. C., 1998. *The image processing handbook* - 3rd ed. Materials Science and Engineering Department North Carolina State University Raleigh - North Carolina, 1998.

Sherbondy, A., Houston, M. and Napel, S., 2003. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *IEEE Visualization*, pp. 171–176.

Sun Xiao-gu, Li Man-chun, Liu Yong-xue, Liu Wei, and Tan Lu, 2009. Accelerated segmentation approach with cuda for high spatial resolution remotely sensed imagery based on improved mean shift. In *Urban Remote Sensing Joint Event*, pages 1–6.

Tiltonand, J. C. and Lawrence, W. T., 2000. Interactive analysis of hierarchical image segmentation, *Geoscience and Remote Sensing Symposium*, 2000. Proceedings. IGARSS 2000. IEEE 2000 International, vol.2, no., pp.733-735.

Unger, M., Pock, T. and Bischof, H., 2008. Continuous globally optimal image segmentation with local constraints. *Intelligence* 2008, pages 1-8.

Wassenberg, J., Middelmannand, W. and Sanders, P., 2009. 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.

Zhang, H., Fritts, J. E. and Goldman, S. A., 2008. Image segmentation evaluation: A survey of unsupervised methods, *Computer Vision and Image Understanding*, vol. 110, Issue 2, pp. 260-280.

Zhang, Y. J., 1996. A survey on evaluation methods for image segmentation, *Pattern Recognition*, vol. 29, Issue 8, pp. 1335-1346.