Accelerated gSLIC for Superpixel Generation used in Object Segmentation

Robert Birkus*
Supervised by: Ing. Wanda Benesova, PhD.†

Institute of Applied Informatics
Faculty of Informatics and Information Technologies STU
Bratislava

Abstract

The goal of our work is to create a robust object segmentation method which is based on superpixels and will be able to run in real-time applications.

The SLIC algorithm proposed by Achanta et al. [2] is a superpixel segmentation algorithm based on k-means clustering , which efficiently generates superpixels. It seems to be a good trade-off between the time consumption and robustness. Important advancement towards the real time applications using superpixels has been proposed by the authors of the gSLIC - a modified SLIC implementation on the GPU (Graphics Processing Unit) [11].

In this paper, we present a significant acceleration of this superpixel segmentation algorithm gSLIC implemented for the GPU. A different strategy of the implementation on the GPU speeds up the calculation time twice and more over the presented GPU implementation. This implementation can work in real-time even for high resolution images. We also present our method for merging of similar superpixels. This method uses an adaptive decision procedure for merging of superpixels. Accelerated gSLIC is the first part of this proposed object segmentation method.

Keywords: Superpixel, Image segmentation, GPU, SLIC, gSLIC, Region merging, Real-time

1 Introduction

Superpixels are produced by a deliberate oversegmentation of an image with the goal to generate segments which can serve as basic units in the further image processing. Superpixels are able to increase calculation efficiency viewed from the next processing task because of the reduction of the redundancy in the image. Superpixels are expected to be regular sized as often as possible but on the other hand, they should follow the saliency edges in the image. To distinguish between a high salience and a low salience edge is quite a complicated task, mainly if the processing is performed on a small local area in the image. Hence, the development of a robust superpixel segmentation algorithm, which could run in the real-time applications, remains a challenge in the computer vision tasks. In many areas of application, the effort involved in computing in real-time is of high importance.

Typical real-time applications are from the area of video processing using superpixels, where several approaches could be combined with frame-based superpixel segmentation as, for example, the approach which has been presented in the paper by Chang et al. [3]. A reduction of the time consumption in the superpixel segmentation is needed also in applications based on still images, especially if we need to process large images.

One of the further image processing tasks after superpixel segmentation is the object segmentation. Automatic universal object segmentation is one of the most common problems in computer vision. To achieve object segmentation using the superpixels, superpixels, which belong to the same object must be merged. This seems to be a simple task, but in fact it is a big challenge. The real difficulties are already open in the implementation. The basic idea is to merge similar superpixels together, but the real question is: which superpixels are similar? There are lots of features and variables to consider.

In this paper we firstly summarize some selected already published methods of superpixel segmentation. We focus on the SLIC algorithm. Further, we also summarize some GPU implementations of superpixel segmentation algorithms. The focus is predominantly on the parallel implementation of the SLIC algorithm on GPU, called gSLIC. Our main contribution is the acceleration of the gSLIC implementation using parallel reduction. We describe our implementation in detail and discuss different key points of our implementation, specially those which are important from the point of view of efficiency. We also describe our CPU implementation of our superpixels merging algorithm, which merges the superpixels based on an adaptive threshold according to the color difference of the most similar neighbor. We also show some results of the merging procedure. The paper contains detailed results of the segmentation quality of SLIC and gSLIC and

*rbirkus@gmail.com
†vanda_benesova@stuba.sk

also the speedups of our implementation compared to the gSLIC implementation. At the end of this paper we briefly discuss our achieved results and present our future work.

2 Related work

Selected already published methods of superpixel segmentation and region merging are briefly summarized in this section.

2.1 Superpixel segmentation

In Spatially Coherent Clustering using Graph Cuts [14], Zabih and Kolmogorov propose a method with the goal to overcome the absence of spatial coherence in segmentation while a clustering in feature space is used. An energy function which consists of a term representing the energy in the spatial space and a term representing the energy in the feature space has to be minimized using graph cuts.

Veksler and Boykov [13] formulate the superpixel partitioning problem in an energy minimization framework, and optimize with graph cuts. The presented energy function explicitly encourages regular superpixels and this method is also suitable for 3D supervoxel segmentation. An image is covered with overlapping square patches of fixed size. Hence, each pixel is covered by several patches, and the task is to assign a pixel to one of them. TurboPixels [8] is an iterative algorithm which starts by evolution from seeds placed regularly in the image. The algorithm then iterates until no further evolution is possible, i.e., when the speed at all boundary pixels is close to zero. The iteration loop involves: an evolution of this boundary, estimation of the skeleton of the unassigned region and updating of the speed of each pixel on the boundary and of unassigned pixels in the immediate vicinity of the boundary.

Shi and Malik [12] propose a graph-theoretic criterion for measuring the goodness of an image partition - the normalized cut. The authors showed that the minimization of this criterion can be formulated as a generalized eigenvalue problem. A computational method based on this idea has been developed and presented by the authors and applied to segmentation of brightness, color, and texture images.

Felzenszwalb and Huttenlocher [5] define a predicate for evaluating two regions of an image whether or not there is evidence for a boundary between two components in a segmentation. This predicate is based on measuring the dissimilarity between elements along the boundary of the two components relative to a measure of the dissimilarity among neighboring elements within each of the two components.

2.1.1 SLIC algorithm

R. Achanta et al. introduced the Simple Linear Iterative Clustering (SLIC) [1] method for producing compact and nearly uniform superpixels. The efficiency, simplicity and the performance of the algorithm makes it widely used and often modified with the goal to achieve even better performance. Yuheng Ren and Ian Reid introduced a parallel implementation of the SLIC superpixel segmentation [11], called gSLIC. The gSLIC implementation uses GPU and the NVIDIA CUDA framework and is able to achieve speedups of about 10x to 20x over the native SLIC sequential implementation.

The SLIC algorithm is based on the k-means clustering principles. Each pixel is associated with a 5-dimensional feature vector [L*a*b x y], where L*a*b are color coordinates in CIE L*a*b space and x,y are spatial coordinates. L*a*b color space was designed, so that color differences measured as Euclidean distance in the L*a*b space correspond with color differences given by human perception. Although the used conversion from RGB to L*a*b includes conversion errors due to missing information about the spectral characteristics of the used camera, this error seems to be irrelevant and using L*a*b coordinates better results could be achieved than with using RGB color coordinates. In the initialization step, positions of all seeds are defined in a regular grid step S, up to a small shift to avoid image edges. The grid size is calculated as in Equation 1, where N is the number of pixels in the given image and k is the required number of segments.

$$S = \sqrt{\frac{N}{k}} \tag{1}$$

The rough size of each superpixel is then given as a square of the grid size S. After the mentioned initialization step the k-means clustering will be calculated for each seed and subsequently the position of the seed will be iteratively updated - shifted into the center of the newly derived cluster. More iterations (typically 5 to 20) are necessary for the useful segmentation result. Finally, in the last step named enforce connectivity, extremely small superpixels will be removed - included within another superpixel. For a balance between a regular form of the superpixels and a form of superpixels given by the color differences, a compactness constant has to be defined and used in the distance measure definition as a weighting factor.

2.2 Superpixels - GPU implementations

Brian Fulkerson and Stefano Soatto introduced a parallel GPU implementation of Quick Shift: they called it Really Quick Shift [6]. This implementation is able to achieve speeding up of 10x to 50x over the original CPU version of Quick Shift. Quick Shift operates on each pixel in the image independently of its distant surroundings. Hence, the GPU implementation basically follows the steps of native

Quick Shift because the Quick Shift is a good parallelizable algorithm. Fulkerson et al. first built a simple GPU version of Quick Shift that simply copies the image to the device and breaks the computation of the density and the neighbors into blocks for the GPU to process. This simple GPU version is faster than the CPU version. However, Quick Shift needs many memory accesses and the global memory of the GPU is slow. Memory latency is a bottleneck for this GPU version. To avoid memory latency they used the advantage of GPUs shared memory. They load an apron of pixels surrounding the block into shared memory. However, this operation is not easily separable and the shared memory is quickly exhausted. So, instead of this solution they map the image and the estimate of the density to a 3D and 2D texture. This GPU version is faster than the previous one. They evaluated both GPU implementations and the CPU implementation as well. This GPU implementation of Quick Shift provides a 10 to 50 times speedup over the CPU implementation, resulting in a super-pixelization algorithm which can run at 10Hz on 256x256 images.

2.2.1 gSLIC implementation

Parallel implementation of the Simple Linear Iterative Clustering (SLIC) superpixel segmentation (gSLIC) is proposed by Carl Yuheng Ren and Ian Reid [11]. The implementation was applied and tested using GPU and NVIDIA CUDA framework. The authors have presented speedups of 10x to 20x on a single graphics card compared with the CPU sequential implementation. The presented gSLIC algorithm differs from the originally proposed SLIC algorithm in the way in which the clustering is carried out. Whereas the SLIC algorithm uses a clustering procedure based on an evaluation in the surrounding of each seed in the 2S x 2S region of pixels, gSLIC algorithm is running in the opposite way. Each pixel is associated with the 9 nearest cluster centers and the search is running for the nearest of the nearby 9 cluster centers. Therefore, the pixel will be labeled with the nearest cluster's index.

Hence, gSLIC has been modified in order to carry out the reasonable part of the parallel computing on GPU by one-thread-per-pixel computing. In general, gSLIC can be split into two parts: CPU and GPU. The image has to be acquired by the host function running on the CPU, then it can be transferred to the GPU device memory. Henceforth, the main part of the algorithm: color space transformation (RGB to L*a*b) and clustering will be carried out on the GPU. Subsequently the derived segmentation mask will be transferred back to the host function again, where a recursion-based post processing function runs to enforce the connectivity of all superpixels.

The color space transformation part is naturally pixel-wise parallelizable, so gSLIC uses one thread per pixel on 16 x 16 blocks. Then one thread per cluster will be used to initialize cluster centers. Next in the assignment step each thread takes care of one pixel. However, the block assignment is a little more complex. The initial size of each cluster is determined by S, where S is the grid interval calculated as in Equation 1. In most cases, the size of each cluster is larger than the thread block size, thus clusters consist of multiple thread blocks. This block assignment guarantees that all threads of the thread block need to search the same set of cluster centers in the neighborhood for the nearest one. Thus gSLIC pre-load the cluster centers' information into local shared memory for higher efficiency. In each iteration after all pixels have been assigned with a label (which is the index of the nearest center), gSLIC uses one thread per cluster to update the cluster center. After the k-means iteration has converged, the labeled image will be transferred back to the host as a segmentation mask. The post-processing to enforce connectivity is the same implementation as in SLIC.

2.3 Region merging

J. Ning et al. [10] present a region merging based interactive image segmentation method. The image is initially segmented by mean shift segmentation and the users only need to roughly indicate the main features of the object and background by using some strokes, which are called markers. With the similarity-based merging rule, a two-stage iterative merging algorithm was presented in the paper [10] to gradually label each non-marker region as either object or background.

H. Dunlop et al. [4] propose a detection and segmentation method incorporating features from multiple scales. This method was tested for the identification of rock appearances and has been evaluated on representative images from the Mars Exploration Rover catalog. This method uses a superpixel segmentation followed by region-merging to search for the most probable groups of superpixels. The authors believe that the method provides promising results for object identification in natural scenes.

3 Our contributions

In this section we summarize all of our contributions in detail.

3.1 Accelerated gSLIC

Our main contribution is a different strategy of the implementation which has been done in the cluster centers updating part. It is also mentioned by the authors of gSLIC [11] that this part could be accelerated by using a parallel reduction algorithm. Based on the technical report by Mark Harris [7] using all of the optimization techniques a significant improvement of the time consumption has been achieved. We also optimized the color space conversion (RGB to Lab) using floating-point constants instead

of double-precision constants. This optimization eliminates the redundant conversions from double to float.[1]

The principle of our implementation does not differ from the principle of gSLIC. The only difference is in the work management of threads. To calculate the new cluster center gSLIC searches over a 2S x 2S region around the current cluster center with one thread. The reason why the authors of gSLIC used only one thread to calculate the mean value of this region of pixels is to avoid atomic operation. To avoid both the atomic operation and using only one thread an implementation of parallel reduction is needed. In contrast to gSLIC, our implementation also works with the 2S x 2S region of pixels around the current cluster center, but instead of one thread we are using a block of threads to do the mean value calculation.

In most cases the size of the 2S x 2S region is larger than the size of thread block. To maximize the number of threads working on one region we could use more thread blocks than gSLIC did at the k-means iteration. However in parallel reduction the threads need to share some data. To efficiently share data between threads we have to use the shared memory of CUDA, even if we can use only one block of threads per region due to the shared memory's block restriction. We also considered using the global memory to share data between multiple blocks of threads, but it is much slower than the shared memory and even with L1 cache memory it cannot compete with the shared memory in this task. However, in case of extremely large regions, maybe the solution with multiple thread blocks using global memory would be better. We have not tested it yet.

In the first step of our implementation due to the larger region size as the block size each thread of the thread block calculates preliminary results from multiple pixels and stores it in the shared memory. When the amount of data is reduced to the number of threads per block the parallel reduction begins.

3.1.1 Occupancy

The occupancy can be defined as a proportion of active threads and maximum active threads. To hide the latency and gain maximum efficiency we must have as many active threads as possible. Let us consider Kepler GK104 architecture in the following calculations. Its limitations are shown in the Table 1.

In our implementation we need 24 bytes of shared memory per thread to store the temporary results. So, on GK104 architecture we can have $48kB/24B = 2048$ active threads per multiprocessor, which is the maximum. That means the shared memory usage does not degrade our efficiency. However, if we would need, we can decrease the usage of shared memory by using short int variables instead of int variables. We decided to set the block size to 128. This block size gives us occupancy equal to 1. If we would choose a smaller block size, for example 64, due to the maximum number of blocks per multiprocessor (16) we could have only $64 * 16 = 1024$ active threads per multiprocessor. However, we could choose larger block sizes up to 1024 and the occupancy would be still equal to 1. The reason why we chose the block size 128 is explained in the next section. The most efficient block size for our implementation is architecture-dependent. The register usage does not affect the efficiency of our implementation because each thread can use up to $65536/2048 = 32$ registers without decreasing the occupancy and in our implementation each thread uses only 28 registers.

[1]The source code will be available on the web site http://vgg.fiit.stuba.sk/image-segmentation-on-gpu/

	KEPLER GK104
Compute Capability	3.0
Threads / Warp	32
Max Warps / Multiprocessor	64
Max Threads / Multiprocessor	2048
Max Thread Blocks / Multiprocessor	16
32-bit Registers / Multiprocessor	65536
Max Registers / Thread	63
Max Threads / Thread Block	1024
Shared Memory Size (bytes)	48K

Table 1: Limitations of Kepler GK104 architecture

3.1.2 Parallel reduction optimization

After the amount of data is reduced to the number of threads per block we make a parallel reduction in shared memory. We are using almost all of the optimization techniques by Mark Harris [7]. We are avoiding using % operator wherever it is possible because it is very slow.

To achieve high memory bandwidth for concurrent accesses, shared memory is divided into equally sized memory modules, called banks, that can be accessed simultaneously. To avoid bank conflicts we are using sequential addressing instead of interleaved addressing. In Figure 1 we show the two types of addressing. Let us consider memory banks of size 4 bytes. To access four elements sequentially we need only one transaction because all of the accessed elements are in the same memory bank. However, using the interleaved addressing we need two transactions because those four accessed elements are situated in two different memory banks.

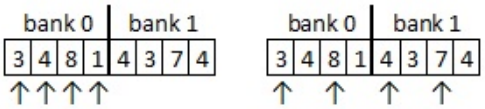


Figure 1: Sequential addressing (left) and interleaved addressing (right)

In parallel programming to achieve high efficiency we have to avoid idle threads as much as we can. Thankfully

for the cases in which the region size is larger than the block size, each thread has to handle multiple pixels and there are no idle threads. Mark Harris also mentioned in his technical report [7] based on Brent's theorem that it is beneficial for each thread to do more sequential work and this is the reason why we chose the smallest block size (128), which give us occupancy equal to 1. If we would choose larger block sizes then the sequential work for each thread would be less and there could be idle threads.

To save useless work in all threads of the thread block we unrolled the for loop by putting #pragma unroll directive right before the loop. #pragma unrolls is a compiler optimization which unroll the loop. However, the number of iterations of the loop have to be known in compile time. In our case the number of iterations is determined by the block size, which is defined as a macro constant. Without unrolling the for loop, all threads would do additional operations every iteration of the loop.

3.2 Superpixel merging

Accelerated gSLIC is the first part of the object segmentation algorithm proposed as bottom-up segmentation using superpixels. The main idea can be briefly described as the merging of similar superpixels. Despite this quite simple and native approach, more challenges are already open in the implementation. The first one is a decision about the similarity of two superpixels. This is quite a complicated task which is a crucial part of our research. The second one is time optimization of the whole segmentation procedure using GPU.

In this paper we present the results of our CPU implementation of the presented merging algorithm using the metric:

$$\Delta D_{Lab} = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2} \quad (2)$$

where (L_1^*, a_1^*, b_1^*) is the mean $L^*a^*b^*$ of the first superpixel and (L_2^*, a_2^*, b_2^*) is the mean $L^*a^*b^*$ of the second superpixel. The whole merging algorithm is presented in Figure 2.

Fast superpixel oversegmentation has been implemented by accelerated gSLIC as presented in the previous section.

The goal of the next section is to label all similar neighbor superpixels of each given superpixel. As mentioned, the decision about the similarity is a complicated task. We wanted to avoid a fixed threshold in the decision about the similarity because of the low invariance of such a threshold. Our decision is based on the relative threshold in relation to the superpixel whose distance ΔD_{Lab} is the least of all neighboring superpixels. The decision about the accepted similar superpixels is the following:

The neighboring superpixel will be labeled as similar if it satisfy the condition:

$$Dist < C * Dist_{min} \quad (3)$$

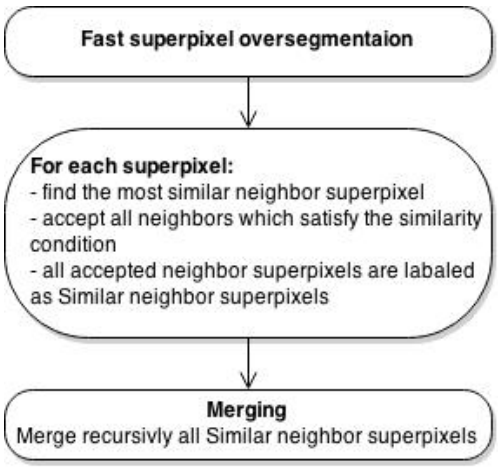
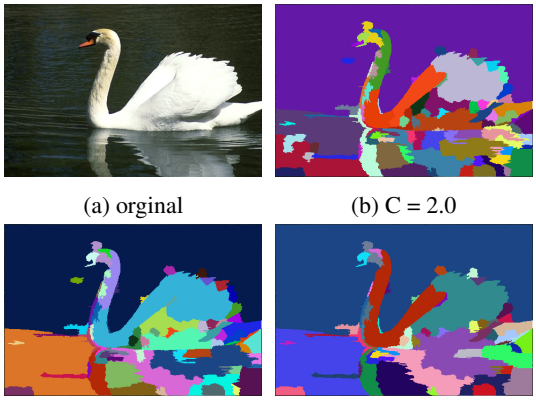


Figure 2: Merging algorithm



(a) orginal (b) C = 2.0

(c) C = 2.3 (d) C = 2.5

Figure 3: First example of merged superpixels

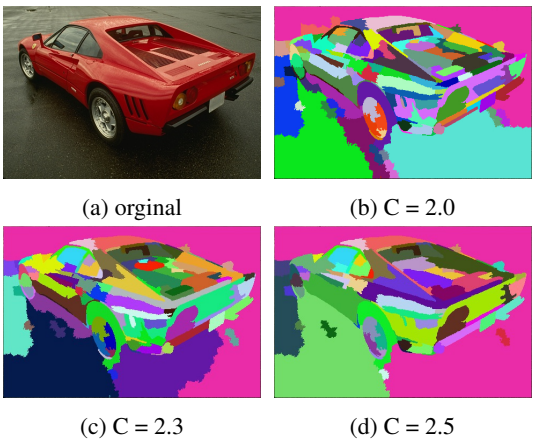
where *Dist* is distance between the superpixels and C is a constant. Our results have been evaluated for $C = 2.0$, $C = 2.3$ and $C = 2.5$.

In last step "Merging", all superpixels labeled as similar will be recursively merged into one segment.

Examples of merged superpixels can be seen in Figure 3 and Figure 4. Merging using a higher constant C produces less new segments, but the undersegmentation error will be probably higher.

4 Results

The time needed for the superpixel calculation using modified gSLIC in comparison with the gSLIC [11] has been evaluated. Table 2 gives the profiling of the clustering part of the algorithm calculated on the NVIDIA GT 740m. The table gives the evaluation of the clustering part of the SLIC algorithm. Time profiling using the NVIDIA GTX 770 for the SLIC clustering is presented in the Table 3. The transfer times between host and GPU memory are not considered in the presented results.



(a) orginal (b) C = 2.0

(c) C = 2.3 (d) C = 2.5

Figure 4: Second example of merged superpixels

Image size	gSLIC	Accelerated gSLIC		Speed-up
	CC 1.0	CC 1.0	CC 3.0	CC 1.0
320x240	7.06 [ms]	1.69[ms]	1.61 [ms]	4.18x
640x480	17.08 [ms]	6.72 [ms]	6.45 [ms]	2.54x
1280x960	66.16 [ms]	27.89 [ms]	26.31 [ms]	2.37x

Table 2: Time evaluation on the NVIDIA GT 740m.

As mentioned, SLIC and gSLIC clustering work differently. Actually the gSLIC is limited in the searching area of every pixel. Although this limitation is marginal, the question is, if this limitation has influence on the quality of the clustering. Therefore, we have evaluated the quality of the segmentation using SLIC and gSLIC and also the modification of the post-processing using the Superpixel Benchmark Toolbox [9] in order to achieve comparable results. The evaluation requires a ground truth segmentation made by humans, available in the dataset. Boundary recall (BR) is the fraction of hand-segmented edges which lie within a threshold distance k of any superpixel edge (in our experiments, k = 2). Since there can be multiple ground truth images for a single input image, they are added together using the OR operation.

The true positives (TP) count is the number of pixels in hand-segmented image, for which there is a superpixel boundary pixel in range k. The false negative (FN) count is the number of pixels in the hand-segmented image for which there is no superpixel boundary pixel in range k. Given these, we can calculate the boundary recall BR as in Equation 4:

$$BR = \frac{TP}{TP + FN} \tag{4}$$

The disadvantage of this metric is that it does not take into account the direction of the edges. Superpixel borders which intersect hand-segmented edges also contribute to the boundary recall. This metric also does not distinguish between superpixel edges which are off by 0, 1 and 2 pixels they all contribute to the boundary recall equivalently.

Results of the evaluation are presented in Figure 5. The

Image Size	gSLIC	Accelerated gSLIC	Speed-up
320x240	5.326 [ms]	0.426 [ms]	12.5x
640x480	8.0 [ms]	1.539 [ms]	5.20x
1280x960	19 [ms]	6 [ms]	3.17x
2560x1920	87.68 [ms]	24.12 [ms]	3.63x

Table 3: Time evaluation on the NVIDIA GTX 770 (using Compute Capability 1.0).

number of iterations was 5 and 10 and the compactness constant was set to 10 and 20. More detailed comparison for the nominal number of superpixels 150 is shown in Figure 6 and Figure 8. From Figure 5 it can be seen that with the rising number of segments the boundary recall results are better. We also can see that the increased number of iterations have not improved the quality of the segmentation that much. However, the compactness constant has a much bigger impact on the segmentation quality. Compactness constant 10 gives much better results than the compactness constant 20. From the evaluation of boundary recall on multiple images in Figure 6 we can see that results are unequivocal. In some of the images SLIC has better results and in others gSLIC has better results, but in average gSLIC gives us better results in boundary recall than SLIC.

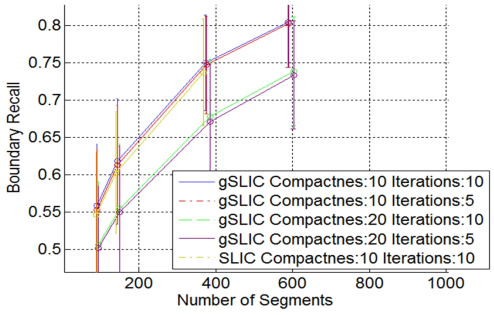


Figure 5: Boundary recall

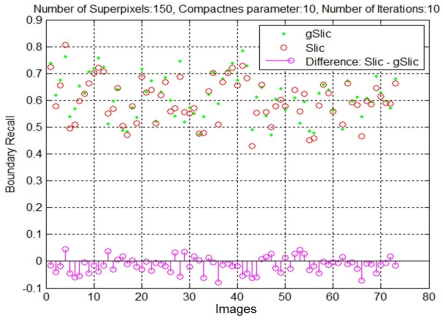


Figure 6: Comparison of the boundary recall for 73 images (No. of superpixels:150)

The undersegmentation error (UE) describes how much area of superpixels crosses the hand-segmented edges. Please refer to the original paper [9] for more information

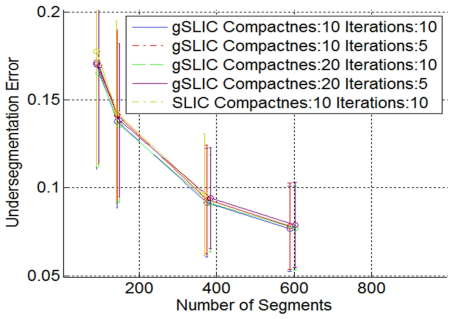


Figure 7: Undersegmentation error

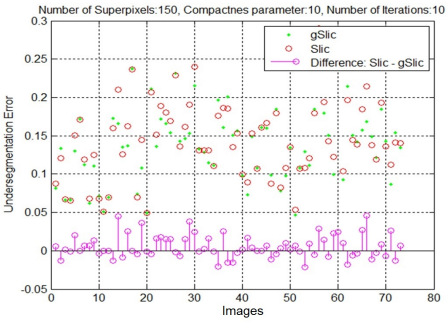


Figure 8: Comparison of the undersegmentation error for 73 images (No. of superpixels:150)

on its calculation. The evaluation of the undersegmentation error is presented in Figure 7 and Figure 8. From Figure 7 it can be seen that with the rising number of segments the undersegmentation error evaluation is better. We can also see that the different number of iterations or different compactness constants have very little impact on the results of the undersegmentation error evaluation. From the evaluation of undersegmentation error on multiple images in Figure 8 we can see that in some images SLIC is better and in others gSLIC is the better one. However, in average SLIC gives better results in undersegmentation error than gSLIC.

The goal of the merging of the superpixels is to reduce the number of segments while the boundary recall is high. The best possible value of boundary recall is the value at the original number of superpixels. In our case the original number of superpixels is 1027 as presented in the Figure 9. Boundary recall values of merged segments are remarkably better compared to the original SLIC segmentation.

5 Conclusions

We have presented modifications in the clustering part of the gSLIC algorithm. We implemented a parallel reduction and achieved significant acceleration as you can see above in Table 2 and Table 3. The modification does not have impact on the quality of the segmentation. We have also presented a comparison between SLIC and gSLIC.

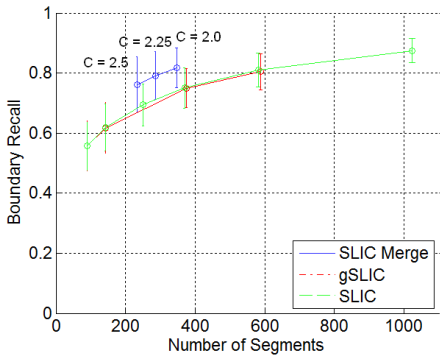


Figure 9: Boundary recall of the merged SLIC superpixels (C = 2.0, C = 2.25 and C = 2.5) in comparison with original SLIC and gSLIC

The results of the comparison is in average the same, depending on the evaluation method. SLIC has better results in undersegmentation error, in Boundary recall gSLIC is the better one. Finally, we have also presented in this paper a bottom-up method of segmentation using superpixels. The achieved results in boundary recall values are better than with using the orginal SLIC segmentation.

6 Future work

The gSLIC image segmentation algorithm consists of two parts. We successfully accelerate the clustering part, but the ”enforce label connectivity” part is hardly parallelizable because it is a recursive function. We tried a completely different strategy based on the morphological processing. We achieved some small acceleration, but it was at the cost of segmentation quality. In our future work we would like to continue in this task to achieve acceleration without any quality degradation.

We presented an algorithm of merging superpixels. In the future we would like to accelerate the presented algorithm using the GPU implementation.

Our future work also contains research of features and decision-making procedures about the similarity of two superpixels. Our next experiments will include texture description and advanced classifications.

7 Acknowledgments

This research has been supported by a grant VEGA 1/0625/14.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels. Technical report, Ecole Polytechnique Fedrale de Lausanne , Report No. EPFL-REPORT-149300, 2010.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. SLIC Superpixels. Technical report, EPFL, 2010.

[3] Jason Chang, Donglai Wei, Fisher III, and John W. A Video Representation Using Temporal Superpixels. *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2051–2058, June 2013.

[4] Heather Dunlop, David R Thompson, and David Wettergreen. Multi-scale features for detection and segmentation of rocks in mars images. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[5] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.

[6] Brian Fulkerson and Stefano Soatto. Really quick shift: Image segmentation on a gpu. Technical report, Department of Computer Science, University of California, Los Angeles, 2010.

[7] Mark Harris. Optimizing Parallel Reduction in CUDA. Technical report, nVidia, 2008.

[8] Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009.

[9] P. Neubert and P. Protzel. Superpixel benchmark and comparison. In *Proc. of Forum Bildverarbeitung*, 2012. Regensburg, Germany.

[10] Jifeng Ning, Lei Zhang, David Zhang, and Chengke Wu. Interactive image segmentation by maximal similarity based region merging. *Pattern Recognition*, 43(2):445–456, 2010.

[11] Carl Yuheng Ren and Ian Reid. gSLIC: a real-time implementation of SLIC superpixel segmentation. Technical report, University of Oxford, Department of Engineering, Technical Report (2011)., 2011.

[12] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 731–, Washington, DC, USA, 1997. IEEE Computer Society.

[13] Olga Veksler, Yuri Boykov, and Paria Mehrani. Superpixels and supervoxels in an energy optimization framework. In *Proceedings of the 11th European conference on Computer vision: Part V*, ECCV'10, pages 211–224, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Ramin Zabih and Vladimir Kolmogorov. Spatially coherent clustering using graph cuts. In *CVPR (2)*, pages 437–444, 2004.