# gSLIC: a real-time implementation of SLIC superpixel segmentation

Carl Yuheng Ren
University of Oxford, Department of Engineering Science
Parks Road, Oxford, UK
carl@robots.ox.ac.uk

Ian Reid
University of Oxford, Department of Engineering Science
Parks Road, Oxford, UK
ian@robots.ox.ac.uk

June 28, 2011

**Abstract**

We introduce a parallel implementation of the Simple Linear Iterative Clustering (SLIC) superpixel segmentation. Our implementation uses GPU and the NVIDIA CUDA framework. Using a single graphic card, our implementation achieves speedups of 10x∼20x from the sequential implementation. This allow us to use the superpixel segmentation method in real-time performance. Our implementation is compatible with the standard sequential implementation. Finally, the software is now online and is open source.

## 1 Introduction

Superpixels are becoming increasingly popular for use in computer vision applications. Unfortunately, most state-of-the-art superpixel segmentation methods suffers from a high computational cost, which make them unable to be used in real-time systems. R. Achanta et al. introduced simple iterative clustering algorithm in [1] to efficiently produce compact and nearly uniform superpixels. The simplicity, efficiency and the performance of the algorithm make it faster and more practical for real-time systems than other existing superpixel segmentation methods, like Normalized cuts [2] and QuickShift [3]. But still, the CPU-sequential implementation of SLIC works at 300∼400ms to segment a 640x480 image. By reducing the number of iterations for each cluster can make the algorithm faster, but this will suffer form loss of performance.

In this paper, we detail an implementation of the SLIC algorithm using NVIDIA CUDA framework. We present improvement 10x∼20x from the original cpu implementation of [1]. We are not the first to attempt fast image segmentation on GPU, notably [4], which presents exact GPU implementation of the quick shift superpixel segmentation algorithm. regardless of the difference in algorithm themselves at the moment, our gSLIC implementation is around 10x faster then [4].

Our full source code with a simple example can be downloaded from `http://www.robots.ox.ac.uk/~carl/code/gSLIC_with_Sample.zip`, in the following sections, we will describe in detail our algorithm and implementation.

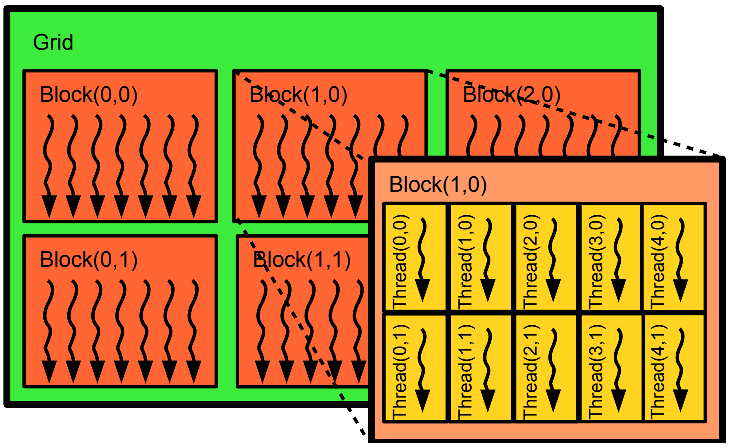

Figure 1: NVIDIA CUDA thread model after [5]

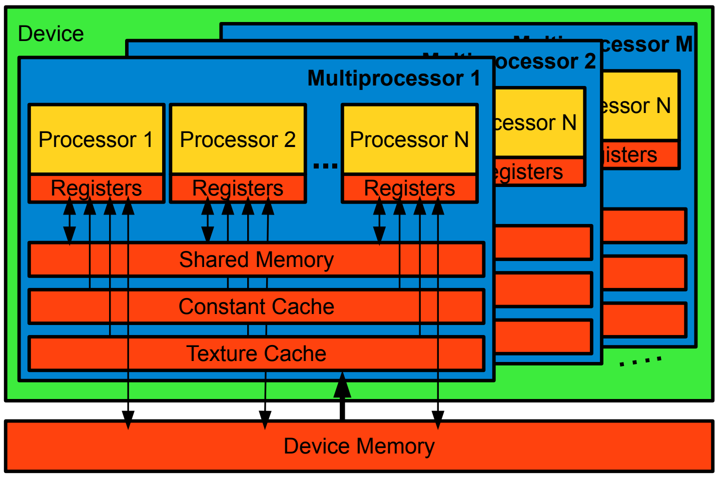

Figure 2: NVIDIA CUDA memory model after [5]

## 2 GPU computing and NVIDIA CUDA

GPUs are traditionally been designed to be used for dense 3D graphic rendering, new-generation GPUs has made GPGPU (General-Purpose Computation on GPUs) available for sovling no-graphic computer vision problems. NVIDIA CUDA provides a set of SDK, software stack and compiler that allows for the implementation of programs in C for execution on GPU. The thread model of CUDA is shown in figure1; CUDA allows C functions (also called *kernels*) to be executed multiple times by multiple *threads*, on multiple GPUs. Each thread carries a *kernel*, and for complete utilization of GPU, thousands of *threads* will be used. *Threads* are grouped in *blocks*, and *blocks* are grouped into *grids* (Figure1). Threads in a *block* share memory and synchronize while *blocks* in a *grid* are independent. Each thread block is executed on only one multiprocessor but a multiprocessor can execute several blocks at the same time. The memory model of CUDA is shown in Figure 2. As it is shown, a thread (executed on a processor within a multiprocessor) have a access to 6 different types of memory: register, local, shared, global (device), constant and texture memory. Each processor in multiprocessor has its own set of register and local memory; each multiprocessor has its on-chip shared, constant and texture cache and shared memory (constant texture cache are both read-only, shared memory is read-write). Global (device) memory access is much slower than on-chip and built-in memory.

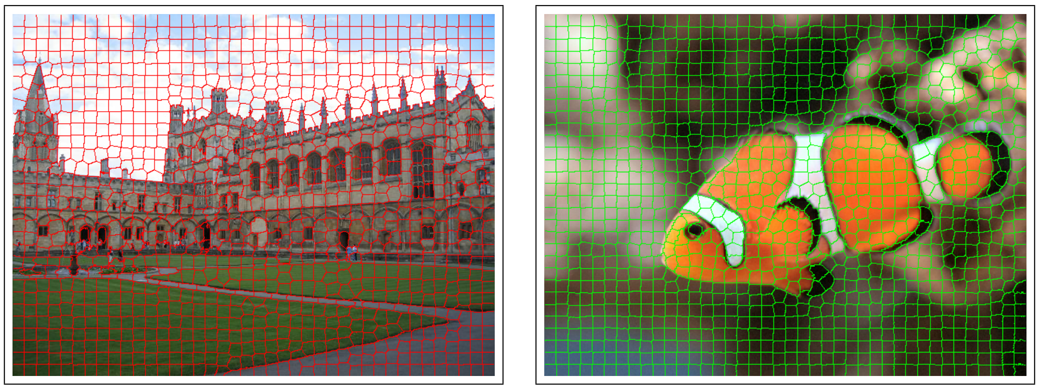
Figure 3: example result of SLIC superpixel segmentation

Consequently the bottle neck in CUDA based software is often global (device) memory access.

## 3 Simple Linear Iterative Clustering (SLIC)

The Simple Linear Iterative Clustering (SLIC) algorithm for superpixel segmentation is proposed in [1]. An example of segmentation result is shown in 3.

The SLIC superpixel segmentation algorithm is a k-means-based local clustering of pixels in the 5-D $[labxy]$ space defined by the $L, a, b$ values of the CIELAB color space and the $x, y$ pixel coordinates. The reason why CIELAB color space is chosen is that it is perceptually uniform for small color distance. Instead of directly using the Euclidean distance in this 5-D space, SLIC introduce a new distance measure that considers superpixel size. The SLIC algorithm takes as input a desired number of approximately equally-sized superpixel $K$, then for a image with $N$ pixels, the approximate size of each superpixel is $N/K$. For roughly equally sized superpixels there would be a superpixel center at every grid interval $S = \sqrt{N/K}$. Let $[l_i, a_i, b_i, x_i, y_i]^T$ be the 5-D point of a pixel, cluster center $C_k$ should be of the same form$[l_k, a_k, b_k, x_k, y_k]^T$. The distance measure $D_k$ is defined as:

$$\begin{aligned} d_{lab} &= \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (n_k - b_i)^2} \\ d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\ D_s &= d_{lab} + \frac{m}{S} d_{xy} \end{aligned} \tag{1}$$

where $D_s$ is the sum of the *lab* distance and the *xy* plane distance *normalized* by the grid interval $S$. Variable $m$ is introduced to control the compactness of superpixels. The greater the value of $m$, the more spatial proximity is emphasized and the more compact the cluster.

With the distance matric defined, the SLIC superpixel segmentation algorithm is simply local k-means algorithm, which is summarized in the **Algorithm 1** in [1]. In order to make the make the most of the parallel computing on GPU, we modified the algorithm to enable one-thread-per-pixel computing, as summarized in Table 1

## 4 gSLIC implementation

As is shown in Figure 4 (left), Our algorithm can be split into CPU and GPU two parts. The image is acquired by the host function running CPU, then transferred to GPU device memory. After color space transformation and segmentation has been done by GPU, segmentation mask

```
Algorithm: SLIC superpixel Segmentation
1: Initialize cluster centers $[l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps $S$.
2: Perturb cluster centers in to the lowest gradient position.
4: for each pixel do
5:     Assign the pixel to the nearest cluster center based on initial grid interval $S$;
6: end for
7: repeat
8:     for each pixel do
9:         Locally search the nearby 9 cluster centers for the nearest one,
           then label this pixel with the nearest cluster's index.
10:    end for
11:    Update each cluster center based on pixel assignment and compute residual
       error $E(L1$ distance) between last and current iteration.
12:until $E \leq threadshould$
13:Enforce Connectivity.
```

Table 1: Modified SLIC superpixel segmentation algorithm.

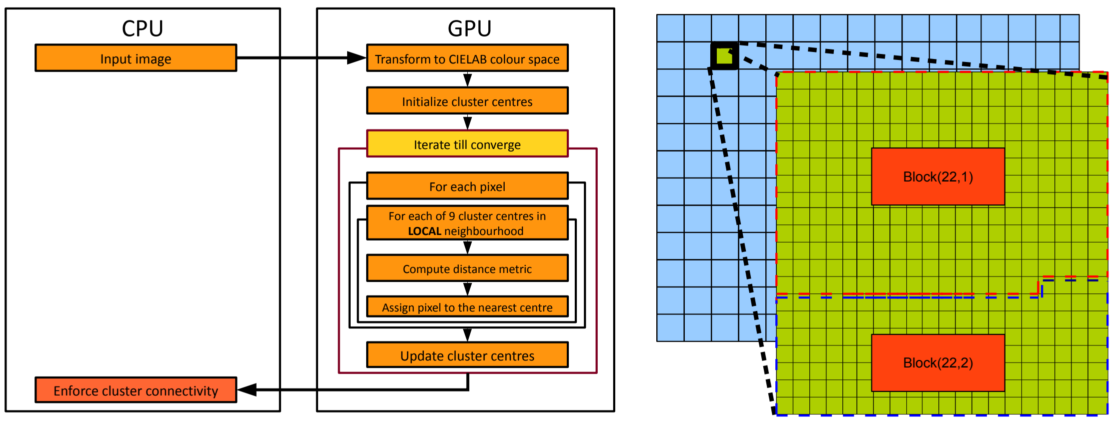
CPU
Input image
Enforce cluster connectivity
GPU
Transform to CIELAB colour space
Initialize cluster centres
Iterate till converge
For each pixel
For each of 9 cluster centres in LOCAL neighbourhood
Compute distance metric
Assign pixel to the nearest centre
Update cluster centres
Block(22,1)
Block(22,2)

Figure 4: Work flow of gSLIC (Left) and Block arrangement example for gSLIC (Right)

is transferred back to host function again, where we run a recursion-based post processing function to enforce the connectivity of all superpixels.

The color space transformation part is naturally pixel-wise parallelizable, so we use 1 thread per pixel on $16 \times 16$ blocks. Then we use 1 thread per cluster to initialize cluster centers. The initial size of each cluster is determined by $S$ defined in section 3. In order to keep compatibility with CUDA 1.0, in which the maximum number of thread per block is 512, we still use $16 \times 16$ fixed sized block in the local k-means iteration step. For most cases, the size of each cluster is larger than 256, thus clusters are consisted of multiple blocks, Figure 4(right) is an example of our block assignment. By using this block assignment, it is guaranteed that all threads within the same block need only to search the same set of cluster centers in neighborhood for the nearest one. Thus we pre-load the cluster centers' information into local shared memory for efficiency.In each iteration, after all pixels has been assigned a label (which is the index of the nearest center), we use one cluster per thread to update cluster center. The reason why we use on thread per cluster instead of on thread per pixel is to avoid atomic operation, which will slow down the whole algorithm greatly. Besides, This part could be accelerated by using parallel reduction algorithm, but in current version, since we have already obtained real-time

| Implementation | 320x240 | 640x480 | 1280x960 |
|---|---|---|---|
| gSLIC | 9ms | 21ms | 86ms |
| SLIC [1] | 88ms | 354ms | 1522ms |

Table 2: Example full processing times for different image sizes

| Implementation | Kmeans Iteration (GPU) | Enforce Connectivity (CPU) |
|---|---|---|
| 320x240 | 6.5ms | 2.5ms |
| 640x480 | 13ms | 8ms |
| 1280x960 | 53ms | 33ms |

Table 3: GPU and CPU time consumption for different image sizes

performance, we did not implemented the parallel reduction. When the K-means iteration has converged, we transfer the labeled image back to host as segmentation mask. The post processing to enforce connectivity is the same algorithm as in [1]. Since it is recursion based method, is not suitable for GPU computing, thus we put it on cpu host function.

## 5 Library Usage

Because the source code for gSLIC is available online, we have decided to include a brief description of the usage of the library, as below.

**FastImgSeg** is the main class. It implements the full gSLIC superpixel segmentation algorithm. It need to be initialized by class constructor or by **initializeFastSeg**. Initialization take the size of image and number of segments as parameter. After initialization, call **LoadImg** to load user image. Note that current lib only take 4-channel image as input, the forth channel is reserved for depth information in later version. When user image has been loaded, call **DoSegmentation** to segment the image. Currently 3 methods are available: **SLIC** (SLIC in CIELAB space), **XYZ_SLIC** (SLIC in XYZ space), **RGB_SLIC** (SLIC in RGB space). The second parameter of **DoSegmentation** is the weight $m$ for spatial distance, as defined in section 3. When segmentation is finished, resulting segmentation mask will be stored in the public buffer **segMask**. User can also call **Tool_GetMarkedImg** to draw segmentation boundary on **markedImage** or call **Tool_WriteMask2File** to write segmentation mask to a file.

## 6 Results

Our implementation is designed to produce the same result as the sequential SLIC implementation of [1], thus we use the windows executable provided by [1] as our baseline method in our speed test experiment. We used an Intel Core i7-2600 (3.60GHz) machine with a NVIDIA GTX460 graphic card to run all our speed test. In Table 2 we show the comparison between the processing time consumed by SLIC [1] and gSLIC for an single image at three sizes. The times of speeding up by using our GPU implementation increases with the size of image, achieving 10 20 times faster than the original sequential implementation. In Table 3 we show the processing time consumed by both the GPU Kmeans iteration part and CPU enforce connectivity. The time consumption by the recursive enforce connectivity increases much faster than the GPU iteration part, so in the future work, we will introduce a parallel version of enforcing connectivity algorithm.

## 7 Conclusion

In this paper, we introduced a GPU implementation of Simple Linear Interactive Clustering (SLIC) superpixel segmentation algorithm. We achieved a speed up of 10x$\sim$20x with a single video card, making superpixel segmentation methods capable for real-time application. Besides, our implementation makes the SLIC framework extendable for different color spaces and distance measure, so extensions and modifications can be made easily on top of gSLIC. Finally, all the source code is available online now.

## References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk, "SLIC Superpixels," tech. rep., EPFL, EPFL, 2010.

[2] G. Mori, "Guiding model search using segmentation," in *ICCV*, pp. 1417–1423, 2005.

[3] A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," in *ECCV (4)*, pp. 705–718, 2008.

[4] B. Fulkerson and S. Soatto, "Really quick shift: Image segmentation on a gpu," tech. rep., Department of Computer Science, University of California, Los Angeles, 2010.

[5] NVIDIA, *NVIDIA CUDA Programming Guide 3.0.*, 2010.