Advanced Programming for HPC
Final Project – *Image segmentation and filtering*

This year, the final project considers an image as input. The final goal is to partition this image: this process is called image segmentation. It results into a new representation of the image, using some regions.
The first step of this project consists in filtering the image using a Median Filter. This imply to convert the input from RGB to HSV space, and then to make a gray image thanks to the value (V coordinate). The second step is the segmentation itself. It relies on the calculation of the minimal spanning tree (MST) of the graph associated to the filtered image.

MF: Median Filter
You may find a lot of information concerning MF on Internet. In few words, it is a filter that reduces the noise, but keeping the edges in the input image. At each pixel, it relies on the calculation of the median of the neighboring. This is done using the Value (in HSV). A median need a kind of sort, but per pixel. The filter then give the median value to the pixel.

MST: Minimal Spanning Tree
The MST is a very useful tool in graph theory (and applications). Starting with a connected (non oriented or oriented) graph $\mathbb{G} = (V, E)$ with positively valued edges, it builds a tree that contains all the nodes from V, and such that the sum of the values of the edges is minimal. This last condition means that you cannot obtain another tree spanning all the nodes with a lower sum of edges. Most well known algorithms are Prim, Kruskal, Boruvka. Notice that the later is quite easy to parallelized, so it is the one you should use.

Segmentation by graph cut
The final segmentation is possible thanks to a MST made on the input image. It consists to remove the edges whose values are above a given threshold. The result is a forest tree, where each tree is a segmented region.

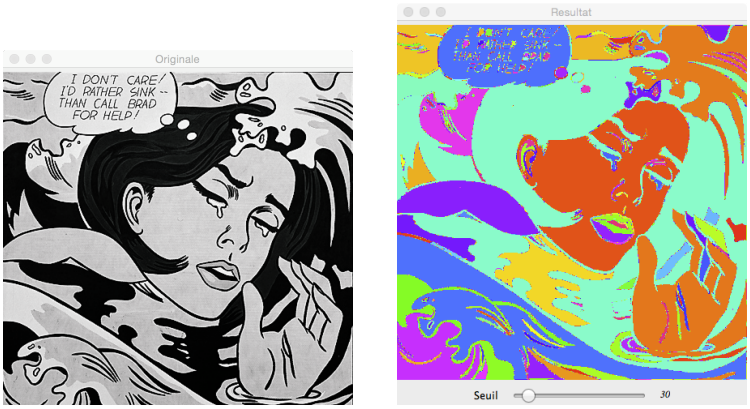

Figure 1: Example of segmentation result: on the left, the image (gray levels, using the Value) ; on the right, the obtained regions.

N.B. : For each exercise, analyze the obtained performances for several parameters (size of kernel, threshold, etc.) and conclude in a report. You have to send your report and your work before January 27, at my email address: lilian.aveneau@univ-poitiers.fr. The delivery have to be compressed into an archive containing one folder called `student`, including the files `student{1|2|3|4}.hpp` and `student{1|2|3|4}.cu` (so one folder, containing 8 source files plus your report).

Exercice 1: Median Filter in Thrust

This exercise concerns the Median filter made on GPU using Thrust. The work have to be done in the files `student1.hpp` and `student1.cu`.

Many solutions are possible here. One consists to scan the values of the pixel, and to compute into a functor the median value of the neighborhood, and then returns it ... so it is a MAP, but with a very slow functor.

Another solution may relies on a kind of sort by key: Values need one byte, and you will have less than 2^{24} pixels (or else you can split the work). So gluing the two, you have integers! Into each key, you have to fill the neighborhood of the current pixel. After the sort, the mean is quite easy to obtain.

Notice that the maximum size of the filter is quite short, allowing you to test your algorithm with classical CUDA device.

At last, you have to save the result into a PPM file.

Exercice 2: Median Filter in CUDA

This exercise consists to do the same work but without using Thrust. So, the goal is to write old good CUDA kernels, one for the RGB for conversion to Value, the other for the filtering itself. Obviously, here, you have to choose the number of blocks and threads per blocks, and to apply the BLOCKING pattern (hence, to load data into shared memory and so to do a synchronization ...)

Notice that $d \in [1 \dots 15]$, so that $(d+1)^2 = 256$, at most.

The sort will be made per pixel, and so for a small set of values. You can reduce the problem by thinking about a divide-and-conquer algorithm, but just for calculating the median but not for a full sort (do some tests!).

Exercice 3: MST in Thrust

This exercise relies on two steps. In the first one, you have to build a MST. The graph to use is made from the filtered image: each pixel becomes a graph vertex, having 4 edges corresponding to its four neighbors (at north, west, south and east). Each edge has a weight corresponding to the absolute difference of pixels values. Then, the Boruvka algorithm can produce a MST.

In the second step, you can remove edges having a weight greater than a given threshold (this last should be tuned on command line). This leads to a forest. Each tree corresponds to a region of the image, that can be filled (scanning the tree) with a random color (you may use cuRand library for that purpose).

At least, you should save the resulting image into a PPM file.

Exercice 4: MST in CUDA

In this last exercise, you have to do the same calculation than in the previous one but using CUDA (when possible) in order to obtain better computation times.

If, for example, your Thrust implementation need a prefix scan, then you can still use the Thrust function for this part. But, scatter, gather, map and so on have to be made using Cuda.