

PC-2020/21 Histogram Equalization

Lorenzo Arena

lorenzo.arena@stud.unifi.it

Abstract

This project was made as the final work for the “Parallel Programming” exam at the University of Florence. The objective was to create three version of a software to equalize the histogram of a given JPEG image. The first version had to be implemented as a sequential program, while the other two had to be parallel; for the parallel versions one has been implemented using OpenMP on top of the already implemented sequential code and the other is built using CUDA to take advantage of a GPU computing power. The projects has been developed in C on a Ubuntu machine; all tests were made on a Ryzen 7 1700 CPU and a NVIDIA Quadro P2000. The project is hosted on GitHub at <https://github.com/lorenzo-arena/histogram-equalizer>.

Future Distribution Permission

The author(s) of this report give permission for this document to be distributed to Unifi-affiliated students taking future courses.

1. The algorithm

Histogram equalization in image processing can be used to increase an image’s contrast. This is accomplished by spreading the most frequent intensities values across the whole histogram. It works especially well on images which presents foreground and background which are both dark or both bright.

1.1. Implementation

We can start by considering a grayscale image of n pixels, in which n_i is the number of occurrences of gray level i . The probability of an occurrence of a pixel of level i (with $0 < i < L$) in the image is:

$$p_x(i) = \frac{n_i}{n}$$

The histogram is the distribution of the pixel levels in the range $[0, L - 1]$.

We can define the *cumulative distribution function* as the cumulative sum of all the probabilities lying in its domain:

$$cdf_x(i) = \sum_{j=0}^i p_x(x = j)$$

To get a linearized *cdf*, which will produce a flatten histogram, we must normalize the *cdf* to the $[0, L - 1]$ range by using the following formula:

$$cdfn_x(i) = \frac{cdf(i) - cdf_{min}}{(M * N) - cdf_{min}}$$

where M and N are the image’s dimensions. The normalized *cdf* must then be applied to the original histgram:

$$h(v) = round(cdfn_x(i) * (L - 1))$$

While this can be applied to grayscale images by using the pixel value, for color images applying the equalization process to the R, G and B channels would break the image’s color balance since the relative distribution of the color channels would be changed. Thus, the pixels must be converted to another color space (like HSL) where the algorithm can be applied to the *luminance* channel without creating changes in the *hue* or *saturation* channels.

1.2. On HSL color space

HSL (*hue, saturation, lightness*) is an alternative representation of the RGB color space. Instead of describing colors by using the red, green and blue components, it works by using a cylindrical geometry where hue (the angular dimension) gives the description of the perceived color (starting from red at 0° , green at 120° and blue at 240°). Saturation describes the "colorfulness" of the color, giving the "pure color" at value 1, at the edge of the cylinder, while turning to grayscale as the center of the cylinder is approached. Lightness is used as the vertical axis dimension and ranges from pure white (lightness 1) to pure black (lightness 0).

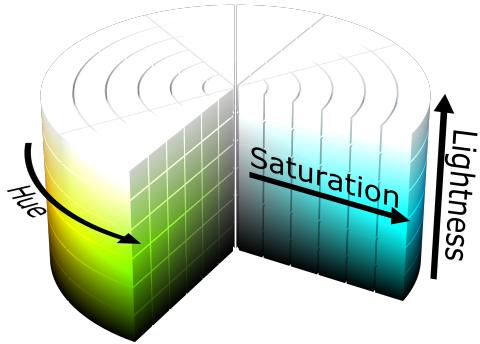


Figure 1. The HSL cilinder

2. The sequential implementation

The sequential solution was implemented by applying one by one all the algorithm steps to an image which filename can be read from the command line when running the program. The histogram is computed in 512 bins for the luminance; by using a special command line option it can also be printed out using *gnuplot*. The benchmarks have been made using `clock_gettime` with `CLOCK_MONOTONIC`, both for getting a measure of the complete execution time (including image reading from disk and output writing to disk) and for measuring the algorithm execution time. The function to convert RGB pixels to HSL color space and viceversa has been implemented following the process reported in [2] under the

sections "*From RGB*" and "*To RGB*".

The tests were done on an image with low contrast, showed in Figure 2.

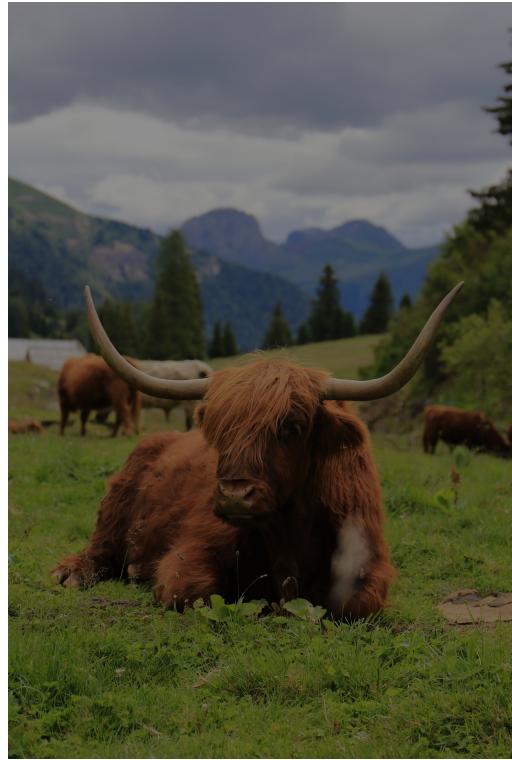


Figure 2. The test input image

The output after the equalization process is showed in Figure 3. The contrast has been increased, making the borders more clear while also darkening the low lights on the animal body and lightening the high lights of the clouds.

3. The OpenMP implementation

4. The CUDA implementation

5. Results

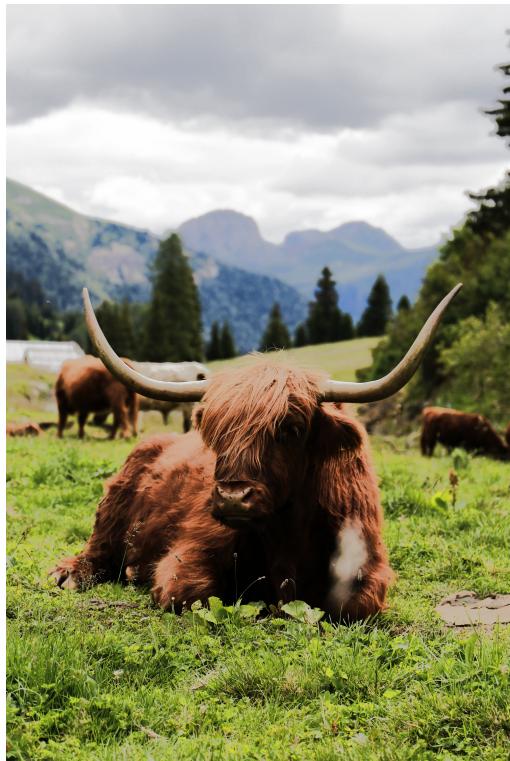


Figure 3. The output image after the equalization process

References

- [1] Wikipedia. Histogram equalization — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Histogram%20equalization&oldid=993129511>, 2021.
- [2] Wikipedia. HSL and HSV — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=HSL%20and%20HSV&oldid=998445401>, 2021.