Sequential and Parallel OpenMP RGB Image Histogram Equalization

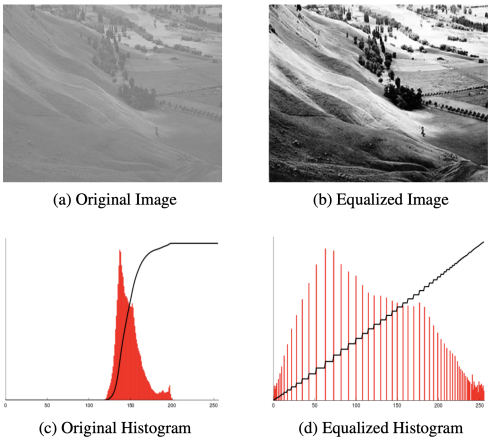
Andrea Simioni
andreasimioni5@gmail.com

Abstract

This project is about the implementation of an important task in Image Processing that makes clearer the vision of an image and homogeneous its histogram: Histogram Equalization. This technique has been applied to RGB images of different dimensions. The task has been realized in both sequential and parallel version, respectively in C++ and OpenMP languages. Sequential and OpenMP ones have been tested on a machine having Intel core i7, quad core. Performances have been finally analyzed in terms of speedup and curves that represent the times to compute equalized images.

1. Introduction

Histogram equalization is one of the best methods for image enhancement. In fact, it non only provides better quality of images with a simple approach, but also without loss of any information. In particular, histogram equalization is a spatial domain method, that produces output image with uniform distribution of pixel intensity. This means that the histogram of the output image is flattened and extended systematically. The application of HE usually increases the global contrast of an image, especially when it is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. HE accomplishes this by effectively spreading out the most frequent intensity values.
The method is useful in images with backgrounds and foregrounds that are both bright or dark. For example, an application to medical x-ray images may provide a better view of bone structures. A key advantage of HE is that it is a fairly straightforward technique and an invertible operator. So, from histogram equalization function known, it can be recovered the original histogram of the image. Furthermore, the calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing unfortunately the real signal.



(a) Original Image (b) Equalized Image

(c) Original Histogram (d) Equalized Histogram

Figure 1: Histogram equalization

1.1. Histogram Equalization: theoretical formulation

Consider a discrete grayscale image x and let n_i be the number of occurrences of gray level k. The probability of an occurrence of a pixel of level k in the image is:

$$p_x(k) = p(x = k) = \frac{n_k}{n} \quad 0 \leq k \leq L \qquad (1)$$

L being the total number of gray levels in the image (typically 256), n being the total number of pixels in the image, and $p_x(k)$ being in fact the image?s histogram for pixel value k, normalized to [0,1]. then, it can be introduced the *cumulative distribution function* corresponding to p_x as

$$cdf_x(k) = \sum_{l=0}^{k} p_x(l) \tag{2}$$

After normalizing cdf_x such that the maximum value is 255 $(h(k))$, each pixel in the new image $y = [y_{ij}]$ can be obtained using the following transformation:

$$y(i,j) = h(x(i,j)) = \\ round(\frac{cdf(x(i,j) - cdf_{min})}{(M \times N - cdf_{min})}) \tag{3}$$

where $x(i,j)$ return the intensity level of pixel $x_{i,j}$ and h is the histogram. Lastly, $M \times N$ gives the image's number of pixels.

1.2. From Grayscale to RGB color Image

The approach described above concerns the equalization for grayscale images. However it can be used also for color images by applying the same method separately to the Red, Green and Blue channels of the RGB color values of the image. Unfortunately, applying the same method on the Red, Green, and Blue components of an RGB image may yield dramatic changes in the image's color balance since the relative distributions of the color channels change as a result of applying the algorithm. So firstly, it is necessary a color space switch, in particular YUV space, obtained as follows:

$$\begin{cases} Y = R \times 0.299 + G \times 0.587 + B \times 0.114 \\ U = R \times (-0.168736) + G \times (-0.331264) + B \times 0.5 + 128 \\ V = R \times 0.5 + G \times (-0.418688) + B \times (-0.081312) + 128 \end{cases}$$

Figure 2: RGB to YUV Conversion

As shown in figure 2, the new model defines a color space in terms of one luma component (Y) and two chrominance components, called U (blue projection) and V (red projection) respectively. After this conversion the equalization has been applied to Y channel histogram, leaving U and V ones intact, and subsequently that image has been reconverted to RGB, using the following equations:

$$\begin{cases} R = Y + 1.4075 \times (V - 128) \\ G = Y - 0.3455 \times (U - 128) - (0.7169 \times (V - 128)) \\ B = Y + 1.779 \times (U - 128) \end{cases}$$

The final result is shown in figure 3.

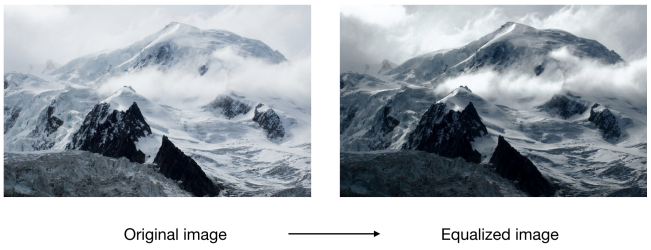


Figure 3: RGB Equalized Image

2. Implementation

As told before, the approach was implemented in two version: sequential and parallel for time evaluation. In this section it's described how they works in detail.

2.1. sequential

the procedure described above was implemented through 3 functions (called in the following order): $make\_histogram$, $cumulative\_hist$ (grouped in the same script, $histogram$) and $equalization$. The first one receives the image

histogram as formal parameter and computes the RGB to YUV conversion, updating the Y histogram. The second function computes the cumulative distribution function and makes the equalization applying. The third one implements the conversion from YUV back to RGB image.

Algorithm 1: make_histogram

Data: $image, hist, YUVimage$
for $i = 0$ *to image.rows* **do**
  for $j = 0$ *to image.cols* **do**
    $R, G, B = image(i, j)$;
    $Y = R \times 0.299 + G \times 0.587 + B \times 0.114$;
    $U = R \times (-0.168736) + G \times (-0.331264) + B \times 0.5 + 128$;
    $V = R \times 0.5 + G \times (-0.418688) + B \times (-0.081312) + 128)$;
    $hist[Y++]$;
    $YUVimage(i, j) = Y, U, V$;
  end
end

Algorithm 2: cumulative_hist

Data: $hist, equalized\_histogram, image.cols, image.rows$
$cumulative\_histogram[256]$;
initiliaze histogram;
for $i = 0$ *to* 256 **do**
  $cumulative\_histogram[i] = hist[i] + cumulative\_histogram[i1]$;
  $equalized\_histogram[i] = ((cumulative\_histogram[i] - hist[0])/(image.cols \times image.rows - 1) \times 255)$
end

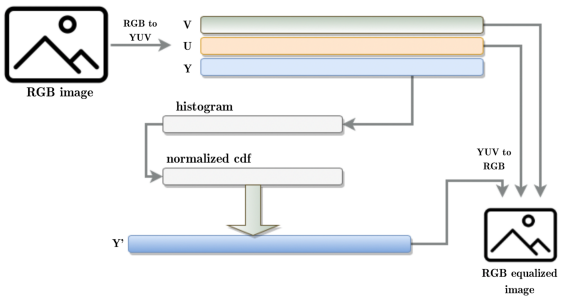


Figure 4: Sequential Implementation

Algorithm 3: equalize

Data: $image, equalized\_histogram, YUVimage$
for $i = 0$ *to image.rows* **do**
  for $j = 0$ *to image.cols* **do**
    $Y = equalized\_histogram[YUVimage(i, j)]$;
    $U = YUVimage(i, j)$;
    $V = YUVimage(i, j)$;
    $R = max(0, min(255, (int)(Y + 1.4075 \times (V - 128)))))$;
    $G = max(0, min(255, (int)(Y - 0.3455 \times (U - 128) - (0.7169 \times (V - 128))))))$;
    $B = max(0, min(255, (int)(Y + 1.7790 \times (U - 128)))))$;
    $image(i, j) = R, G, B$;
  end
end

2.2. parallel OpenMP

For the parallelization of the procedure described above, it was used OpenMP, an API which supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran. The OpenMP parallel version of the code is implemented as the sequential one, with the main difference that, as OpenMP rules, the same functions implemented in the sequential version have been written into a particular OMP structure that parallelizes the sequential for loops: *#pragma omp parallel for*.
The use of this structure has been really easy and convenient: in fact it allows developers to choose the number of threads to be used, depending on the task to execute and the machine which runs it.

3. Result

The testing phase has been performed using three images (Montebianco, Lavaredo, Forest), resizing their own size from 1000x1000 to 10000x10000 and calculating the time to complete the equalization procedure. To ensure greater precision, every result is calculated on the average of 5 runs. Regarding the parallel OpenMP version, the number of threads considered to testing the procedure were 2,4, and 8.

Lastly, it was calculated the speedup term to evaluate performance increase using parallel versions with 8 thread and 4 thread, compared to the sequential one. The results shown below are all referred to the Montebianco image.

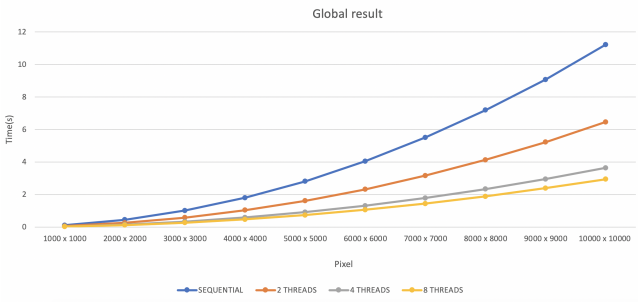


Figure 5: global result

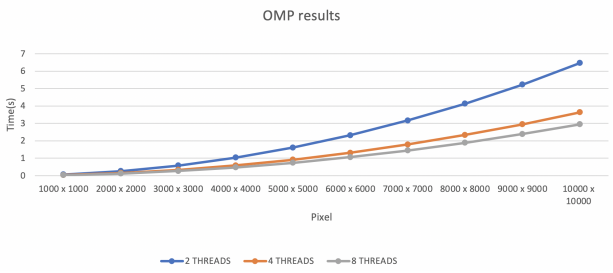


Figure 6: Parallel OMP results

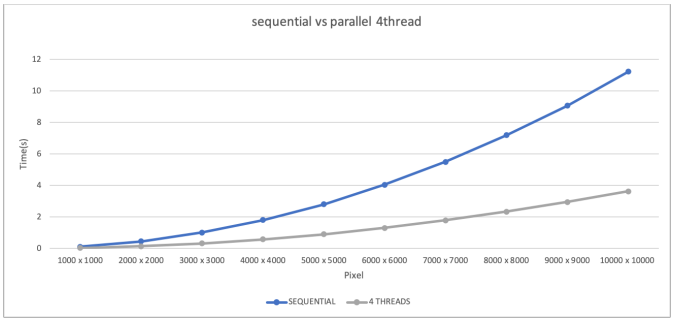


Figure 7: Sequential vs parallel 4 thread

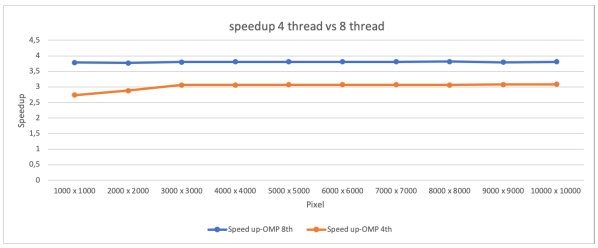


Figure 8: Speedup: 8 thread vs 4 thread

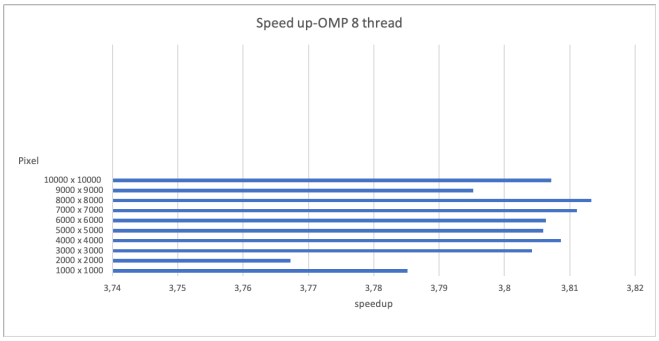


Figure 9: Speedup 8 thread

4. Notes

OpenCV library has been used to load and resize the mages. Two different time evaluators have been used to compute execution time:

- *omp_get_wtime()* for OMP versions;
- *gettimeofday()* for sequential version.

References

[1] Wikipedia - Histogram Equalization,
https://en.wikipedia.org/wiki/Histogram_equalization

[2] Wikipedia - YUV conversion
https://en.wikipedia.org/wiki/YUV

[3] Materiale didattico - Parallel Computing