MI3.22 – Advanced Programming for HPC
Labwork 4 – *Atomic and SCAN in CUDA*

With numerical images, the histogram equalization is a well-known method. It is used to adjust the contrast level of a given image (*cf.* `http://en.wikipedia.org/wiki/Histogram_equalization`). For a gray-level image, this method starts by computing an histogram on the gray levels. Next, it calculates the cumulative distribution function of this histogram. At least, it finishes by spreading the gray levels.

In a mathematical presentation, let $\{x_i\}$ be the set of pixels of an image defined other L gray levels. The histogram is an array counting each gray level l, for $l \in [0 \ldots L-1]$:

$$h(l) = \sum_{i=0}^{n-1} \delta(x_i - l)$$

where n is the number of pixels of the image, and δ the Dirac function such that:

$$\delta(\xi) = \begin{cases} 1 \text{ if } \xi = 0, \\ 0 \text{ else.} \end{cases}$$

The cumulative distribution function r is defined other the gray level as the sum of the occurrence number of the previous values:

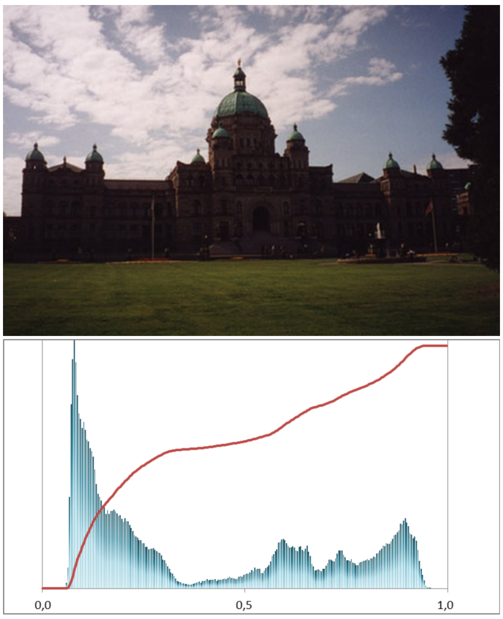
$$r(l) = \sum_{k=0}^{l} h(k).$$

Yes, you are right: This is exactly the lecture 3.
To spread the histogram, we just have to apply the following transformation:

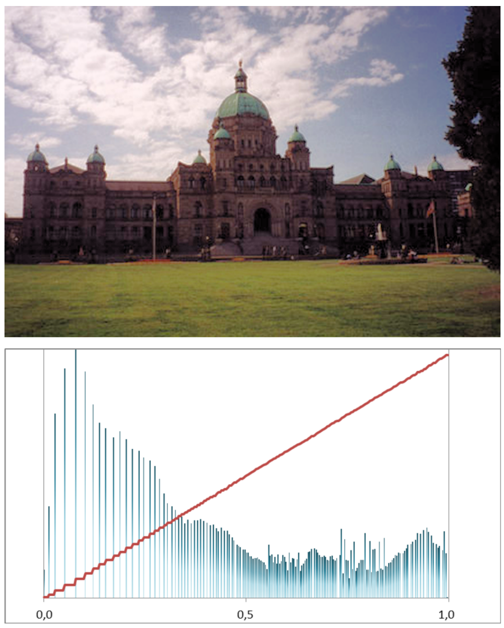
$$T(x_i) = \frac{L-1}{L \cdot n} r(x_i).$$

The technique can be applied to color images using the value component V of the color in the HSV model: *Hue*, *Saturation* and *Value* (*cf.* `http://en.wikipedia.org/wiki/HSL_and_HSV`).

Let us study an example:



⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒
Histogram equalization
⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒



The purpose of the following exercises is to let you code the histogram equalization on color images, and obviously in CUDA (without thrust now). We wait you to apply the optimization notion that you have seen in the previous Labworks.

Exercise 1: Changing color space

1. Implement kernel `rgb2hsv`. Its purpose is, for each pixel of the input image, to compute its value in the HSV space (using the function `RGB2HSV`), and then to split the result in three different arrays. Notice that this is a MAP pattern. The kind of splitting is useful to optimize the memory throughput of kernel in CUDA applications.

2. Implement the inverse transform (`hsv2rgb`), from HSV to RGB. Yes, it true: this transform is very simple, using the function `HSV2RGB`. Notice that this is again the MAP pattern.

3. Verify your work by running the program.

Exercise 2: Computing the histogram

1. Implement kernel `histo`. It takes as input the V component of each pixel (value). It computes the histogram of the image (see *Atomic Operations* in Lecture 4). This question should be made using atomics.
N.B.: The histogram size is 256 (`#define SIZE_HISTO 256`), and the Value $V \in [0 \ldots 1[$. The kernel is launched using `SIZE_HISTO` *threads* per *block*.

Exercise 3: Cumulative Distribution Function

1. Implement a CUDA version using shared memory. Starting from the histogram, this kernel applies the cumulative distribution function $r(l)$. Note that actually it is a scan (*cf. Lecture 3*). Verify the good behaviour of your kernel.
N.B.: This kernel is launched using 1 *block* containing `SIZE_HISTO` *threads*.

Exercise 4: Equalization

At least, implement the equalization (a simple transform) with Thrust and then with CUDA (using shared memory).