Simplest Color Balance

Nicolas Limare[1], Jose-Luis Lisani[2], Jean-Michel Morel[1], Ana Belén Petro[2],
Catalina Sbert[2]

[1] CMLA, ENS Cachan, France
(nicolas.limare@cmla.ens-cachan.fr, moreljeanmichel@gmail.com)
[2] TAMI, Universitat Illes Balears, Spain
({joseluis.lisani, anabelen.petro, catalina.sbert}@uib.es)

Communicated by Yann Gousseau   *Demo edited by* Jose-Luis Lisani

Abstract

In this paper we present the simplest possible color balance algorithm. The assumption under-
lying this algorithm is that the highest values of R, G, B observed in the image must correspond
to white, and the lowest values to obscurity. The algorithm simply stretches, as much as it can,
the values of the three channels Red, Green, Blue (R, G, B), so that they occupy the maximal
possible range [0, 255] by applying an affine transform ax+b to each channel. Since many images
contain a few aberrant pixels that already occupy the 0 and 255 values, the proposed method
saturates a small percentage of the pixels with the highest values to 255 and a small percentage
of the pixels with the lowest values to 0, before applying the affine transform.

Source Code

The source code (ANSI C), its documentation, and the online demo are accessible at the IPOL
web page of this article[1].

Keywords: color balance; affine; saturation; image; improvement; algorithm

1 Introduction

Color balance[2] algorithms attempt to correct underexposed images, or images taken in artificial lights
or special natural lights, such as sunset.

There are many sophisticated algorithms in the literature performing color balance or other
color contrast adjustments [1]. The performance of these many color correction algorithms can be
evaluated by comparing their result to the simplest possible color balance algorithm proposed here.
The assumption underlying this algorithm is that the highest values of R, G, B observed in the image

---

[1]https://doi.org/10.5201/ipol.2011.llmps-scb

[2]"Color balance", Wikipedia, The Free Encyclopedia (accessed January 14, 2010). http://en.wikipedia.org/w/
index.php?title=Color_balance&oldid=336334367

must correspond to white, and the lowest values to obscurity. If the photograph is taken in darkness, the highest values can be significantly smaller than 255. By stretching the color scales, the image becomes brighter. If there was a colored ambient light, for example electric light where R and G dominate, the color balance will enhance the B channel. Thus the ambient light will lose its yellowish hue. Although it does not necessarily improve the image, the simplest color balance always increases its readability.

The algorithm simply stretches, as much as it can, the values of the three channels Red, Green, Blue (R, G, B), so that they occupy the maximal possible range [0, 255]. The simplest way to do so is to apply an affine transform $ax + b$ to each channel, computing a and b so that the maximal value in the channel becomes 255 and the minimal value 0.

However, many images contain a few aberrant pixels that already occupy the 0 and 255 values. Thus, an often spectacular image color improvement is obtained by "clipping" a small percentage of the pixels with the highest values to 255 and a small percentage of the pixels with the lowest values to 0, before applying the affine transform. Notice that this saturation can create flat white regions or flat black regions that may look unnatural. Thus, the percentage of saturated pixels must be as small as possible.

The proposed algorithm therefore provides both a white balance and a contrast enhancement. However, note that this algorithm is not a real physical white balance: It won't correct the color distortions of the capture device or restore the colors or the real-world scene captured as a photography. Such corrections would require a captured sample of known real-world colors or a model of the lighting conditions.

2 Algorithm

The naive color balance is a simple pixel-wise affine transform mapping the input minimum and maximum measured pixel values to the output space extrema. As we explained before, a potential problem with this approach is that two aberrant pixel colors reaching the color interval extrema are enough to inhibit any image transform by this naive color balance.

A more robust approach consists in mapping two values $V_{\min}$ and $V_{\max}$ to the output space extrema, $V_{\min}$ and $V_{\max}$ being defined so that a small user-defined proportion of the pixels get values out of the $[V_{\min}, V_{\max}]$ interval.

3 Implementation

Our input image is an array of N numeric values in the [min, max] interval. The output is a corrected array of the N updated numeric values. Multiple channel images are processed independently on each channel with the same method.

We will perform a color balance on this data where we have saturated a percentage $s_1\%$ of the pixels on the left side of the histogram, and a percentage $s_2\%$ of pixels on the right side; for example, $s_1 = 0$ and $s_2 = 3$ means that this balance will saturate no pixels at the beginning and will saturate at most $N \times 3/100$ at the end of the histogram. We can't ensure that exactly $N \times (s_1 + s_2)/100$ pixels are saturated because the pixel value distribution is discrete.

3.1 Sorting Method

$V_{\min}$ and $V_{\max}$, the saturation extrema, can be seen as quantiles of the pixel values distribution, e.g. first and 99th centiles for a 2% saturation.

Thus, an easy way to compute $V_{\min}$ and $V_{\max}$ is to sort the pixel values, and pick the quantiles from the sorted array. This algorithm would be described as follows:

1. **Sort the pixel values**. The original values must be kept for further transformation by the bounded affine function, so the N pixels must first be copied before sorting.

2. **Pick the quantiles from the sorted pixels**. With a saturation level $s = s_1 + s_2$ in $[0, 100[$, we want to saturate $N \times s/100$ pixels, so $V_{\min}$ and $V_{\max}$ are taken from the sorted array at positions $N \times s_1/100$ and $N \times (1 - s_2/100) - 1$.

3. **Saturate the pixels**. According to the previous definitions of $V_{\min}$ and $V_{\max}$, the number of pixels with values lower than $V_{\min}$ or higher than $V_{\max}$ is at most $N \times s/100$. The pixels (in the original unsorted array) are updated to $V_{\min}$ (resp. $V_{\max}$) if their value is lower than $V_{\min}$ (resp. higher than $V_{\max}$).

4. **Affine transform**. The image is scaled to [min, max] with a transformation of the pixel values by the function f such that $f(x) = (x - V_{\min}) \times (\max - \min)/(V_{\max} - V_{\min}) + \min$.

3.2   Histogram Method

Sorting the N pixel values requires $O(N \log(N))$ operations and a temporary copy of these N pixels. A more efficient implementation is achieved by a histogram-based variant, faster ($O(N)$ complexity) and requiring less memory ($O(\max - \min)$ vs. $O(N)$).

1. **Build a cumulative histogram of the pixel values**. The cumulative histogram bucket labeled i contains the number of pixels with value lower or equal to i.

2. **Pick the quantiles from the histogram**. $V_{\min}$ is the lowest histogram label with a value higher than $N \times s_1/100$, and the number of pixels with values lower than $V_{\min}$ is at most $N \times s_1/100$. If $s_1 = 0$ then $V_{\min}$ is the lowest histogram label, i.e. the minimum pixel value of the input image. $V_{\max}$ is the label immediately following the highest histogram label with a value lower than or equal to $N \times (1 - s_2/100)$, and the number of pixels with values higher than $V_{\max}$ is at most $N \times s_2/100$. If $s_2 = 0$ then $V_{\max}$ is the highest histogram label, i.e. the maximum pixel value of the input image.

3. **Saturate the pixels**.

4. **Affine transform**. Same as for the sorting method.

3.3   Pseudo-code

The following algorithm (Algorithm 1) applies to images with pixel values in the 8 bit integer space (min $= 0$, max $= 255$) with one color channel only. See the following remarks for higher-precision images. Hereafter is the basic implementation, refinements are available in the provided source code.

In Algorithm 1 $image[i]$ are the pixel values, N is the number of pixels, $histo$ is an array of 256 unsigned integers, with a data type large enough to store N, initially filled with zeros. The arrays indexes start at 0.

Algorithm 1: Simplest Color Balance algorithm.

// build the cumulative histogram
for $i = 0, \cdots, N-1$ **do**
$\quad$ $histo[image[i]] := histo[image[i]] + 1$

for $i = 1, \cdots, 255$ **do**
$\quad$ $histo[i] := histo[i] + histo[i-1]$

// search $v_{\min}$ and $v_{\max}$
$v_{\min} := 0$
while $histo[v_{min} + 1] \leq N \times s_1/100$ **do**
$\quad$ $v_{\min} := v_{\min} + 1$

$v_{\max} := 255 - 1$
while $histo[v_{max} - 1] > N \times (1 - s_2/100)$ **do**
$\quad$ $v_{\max} := v_{\max} - 1$

if $v_{max} < 255 - 1$ **then**
$\quad$ $v_{\max} := v_{\max} + 1$

// saturate the pixels
for $i = 0, \cdots, N-1$ **do**
$\quad$ **if** $image[i] < v_{min}$ **then**
$\quad\quad$ $image[i] := v_{\min}$
$\quad$ **if** $image[i] > v_{max}$ **then**
$\quad\quad$ $image[i] := v_{\max}$

// rescale the pixels
for $i = 0, \cdots, N-1$ **do**
$\quad$ $image[i] := (image[i] - v_{\min}) \times 255/(v_{\max} - v_{\min})$

3.4 Higher Precision

For 16 bit integer pixel values, the histogram array method can be used, and needs 65.536 buckets (256 Kb on a 32 bit system, 512 Kb on a 64 bit system, to be compared with the 128 Kb used for a 256×256 image). But the determination of $v_{\min}$ and $v_{\max}$ would benefit of a faster search method, like bisection.

For 32 bit integer pixel values, the histogram size (4.294.967.296 buckets) becomes a problem and can't be properly handled in memory. We can switch to a multi-step process:

- build an histogram with buckets containing more than one single pixel value, such that the histogram size is limited (256 buckets for example, each for a pixel value interval);

- search for the buckets containing $v_{\min}$ and $v_{\max}$;

- restart the histogram construction and search on a subdivision of these buckets.

If an exact precision isn't required, the latest refinements can be skipped.

For floating-point data, the pixel value can no longer be used as an array index, and we must use either a sorting method, or a multi-step method with a histogram containing intervals (not values), then a sorting method on the buckets containing $v_{\min}$ and $v_{\max}$.

Note that the proposed pseudo-code can also be used for images with integer pixel values (as produced by common image capture devices and found in common image formats) stored as floating-

point data (often desired for image processing), by converting the pixel value $image[i]$ to its integer equivalent while filling the histogram.

3.5   Special Cases

If the image is constant (all pixels have the same value v), then, according to the described implementation and pseudo-code, the histogram values are 0 for labels lower than v, and N for labels higher or equal to v, and then for any value of s_1 and s_2 , $v_{\min} = v$, $v_{\max} = v$.

This ($v_{\min} = v_{\max}$) can also happen for non-constant images, the general case being images with less than $N \times s_1/100$ pixels with values below or with more than $N \times s_2/100$ above a median value v. This case can be handled by setting all the pixels to the value v.

4   Color Images

4.1   RGB Color Balance

For RGB color images we can apply the algorithm independently on each channel. We call this algorithm RGB color balance. The color of the pixels is modified in the process because each RGB channel is transformed by an affine function with different parameters and the saturation does not occur on the three RGB channels together. This can be desirable to correct the color of a light source or filter, but in some applications we may want to maintain the colors of the input image.

In that case, many solutions are possible, depending on how we define the "color" to be maintained (hue, chroma, R/G/B ratio) and what we want to correct with this algorithm (lightness, brightness, intensity, luma, ...). A discussion about these color correction variants will be published in a later article, and we present hereafter the simplest version.

4.2   IRGB Intensity Balance

The goal of IRGB intensity balance is to correct the intensity of a color image without modifying the R/G/B ratio of the pixels. We first compute the gray level intensity ($I = (R + G + B)/3$), then this intensity is balanced and transformed into I' by the affine transformation with saturation. Finally, for each pixel, the three color channels are multiplied by I'/I.

But the RGB color cube is not stable by this transformation. Multiplied by I'/I, some RGB components will be larger than the maximum value. This is corrected in a post-processing step by a projection on the RGB cube while maintaining the R/G/B ratio, i.e. replacing pixels out of the RGB cube by the intersection of the RGB cube surface and the segment connecting the (0,0,0) point and the color of the pixels to be corrected. This projection has three consequences:

- *commutativity*: computing the intensity I of an image after correction by this algorithm doesn't give the same result as computing the intensity of an image and correcting this intensity by the affine balance algorithm with saturation described at the beginning of this article;

- *monotonicity*: some pixels with intensities $I_1 < I_2$ can be transformed into pixels with intensities $I'_1 > I'_2$ if the second pixels has to be corrected by projection;

- *precision*: because the projection step is darkening the projected pixels, less than $s_2\%$ of the pixels will have their final intensity saturated to the maximum value.

Moreover, adjusting the saturation on the average I of the three RGB channels means that, unless the three channels are equal (gray image), before the projection less than $s_2\%$ of the pixels are saturated

to the maximum value on the three RGB channels while more than $s_2\%$ of the pixels are saturated to the maximum value on at least one of the RGB channels.

Better solutions to achieve a balance of a color image without these problems require the use of other color spaces and are beyond the scope of this article.

5   Online Demo

With the online demonstration[3] it is possible to test this algorithm on any image and set the desired percentage of saturated pixels. This demo presents the algorithm applied independently to the R, G and B channels (RGB color balance), and to the intensity channel while maintaining the R/G/B ratio (IRGB intensity balance).

For gray-scale images, these two versions are identical to applying the simple algorithm to the gray level.

6   Source Code

An ANSI C implementation[4] is provided and distributed under the GPL license.

This source code includes two implementations of the color balance: an 8-bit integer implementation based on the histogram algorithm with $O(N)$ complexity and a lookup table for fast affine transform is used for the RGB color balance; and a generic floating-point implementation based on $qsort()$, with $O(N \log(N))$ algorithmic complexity is used for the IRGB intensity balance.

The histogram code is used for the online demo. The source code history and future releases are available on an external page[5].

7   Examples

In figures 1 to 9 we show, from left to right, the original image, and its result by RGB color balance with $s = 0\%$, 1%, 2% and 3% of the pixels saturated, half at the beginning of the histogram and half at the end of the histogram (that is, $s_1 = s_2 = s/2$). In these examples, the algorithm has been applied independently on each color channel. It is quite apparent that some saturation is almost always necessary, but that the needed percentage is variable.

---

[3]https://doi.org/10.5201/ipol.2011.llmps-scb
[4]https://doi.org/10.5201/ipol.2011.llmps-scb
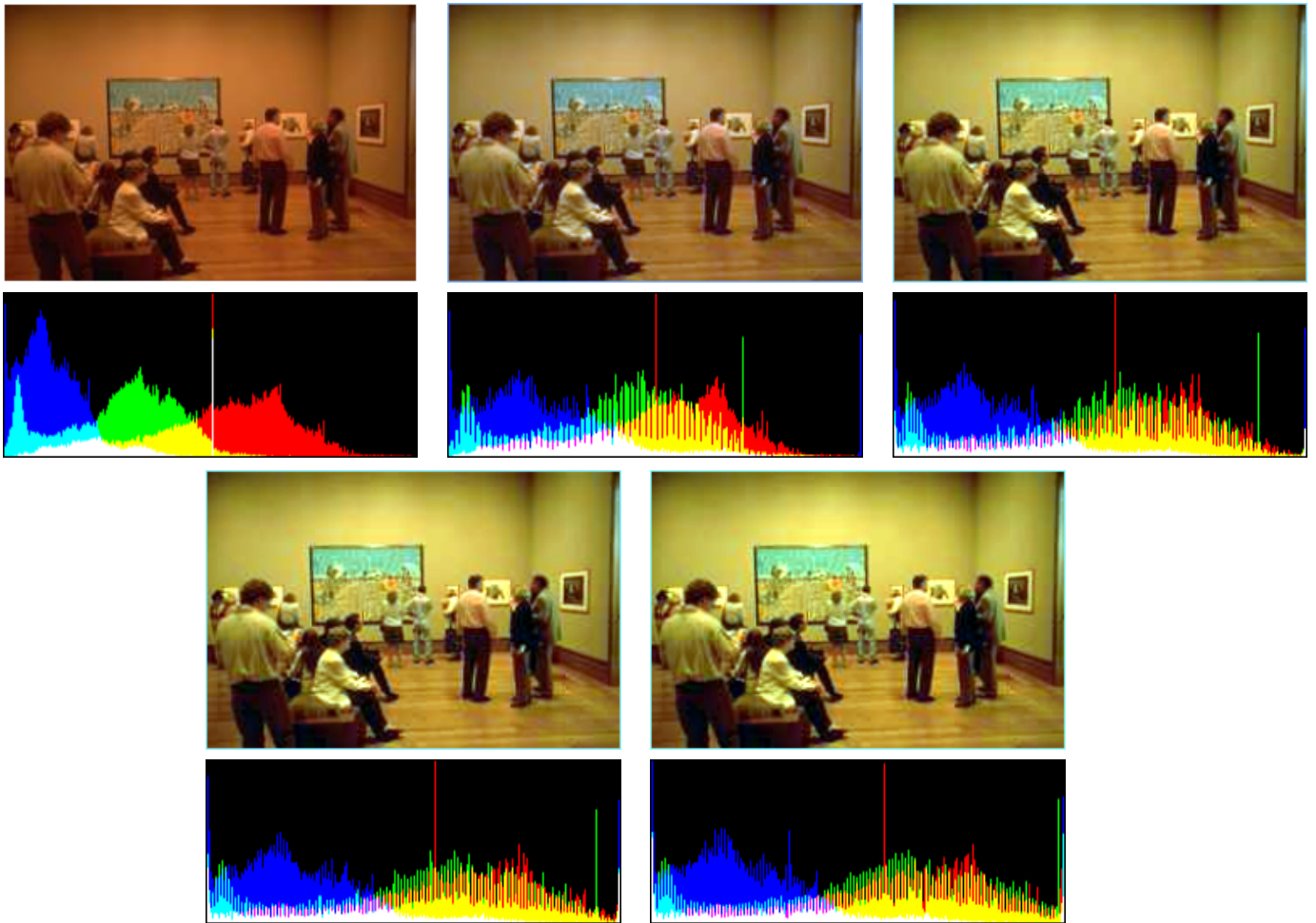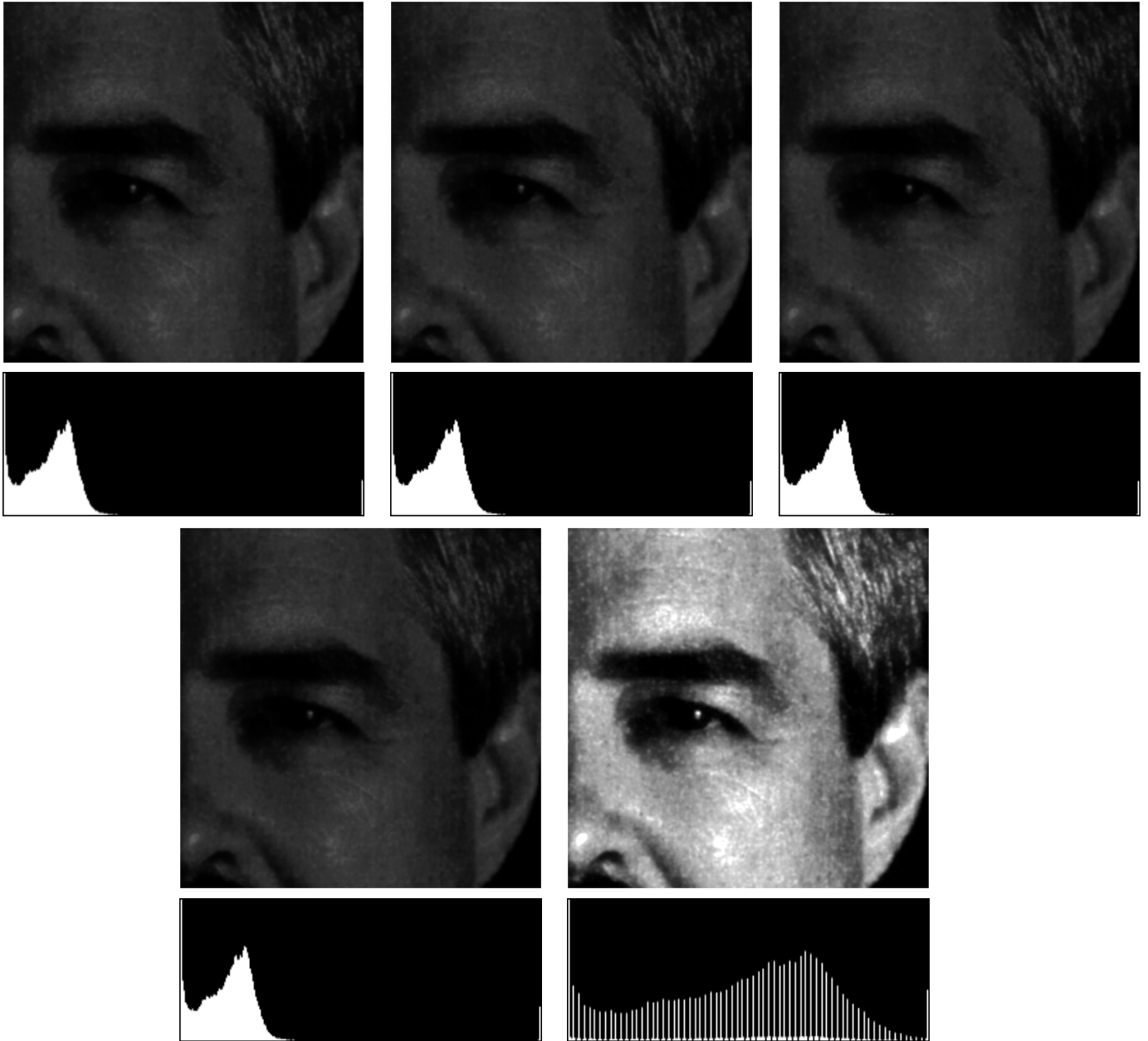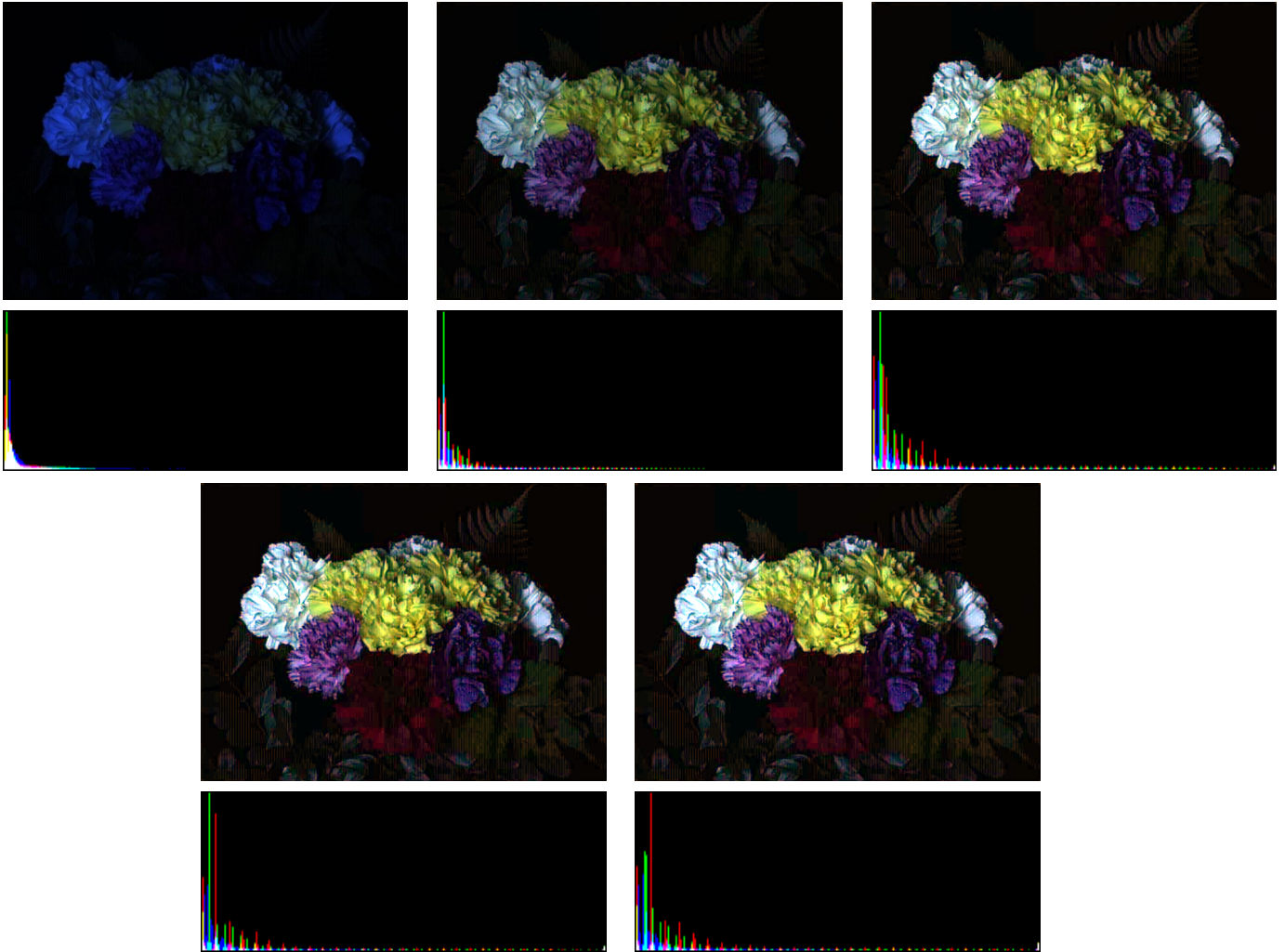[5]http://dev.ipol.im/git/?p=nil/simplest_color_balance.git

Figure 1: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. Here, the 0% saturation already gives a good result, and 1% is optimal. Notice how the orange ambient light has been corrected to a more daylight image.
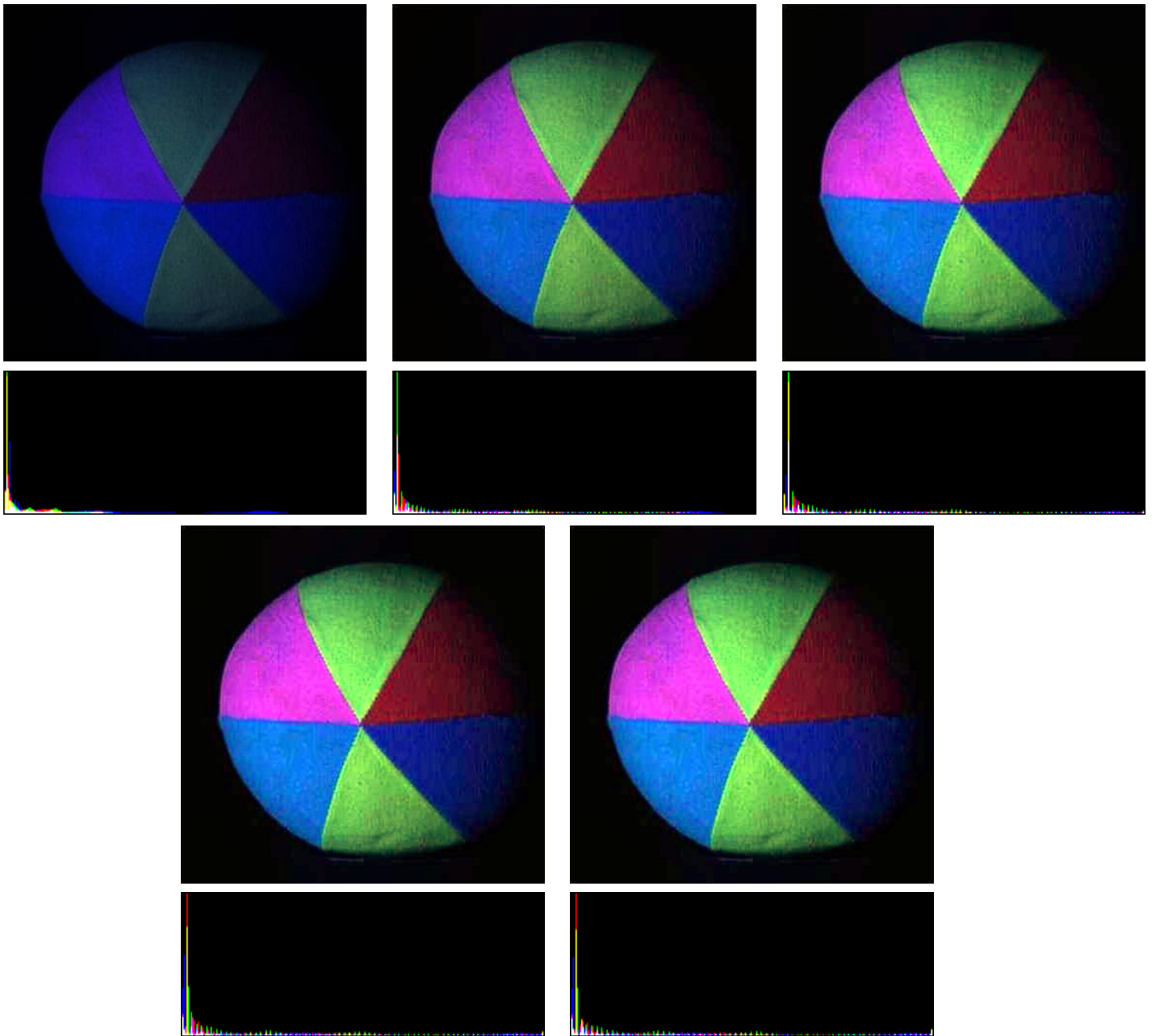
Figure 2: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its intensity histogram is displayed. In this case a white thin rim surrounds the image. This rim occupies more than 2% of the image. Hence, the 3% threshold is the right one.

Figure 3: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. Like the preceding ones, this image in completely unnatural blue light is often used to illustrate color balance, or color contrast adjustment algorithms. A trivial affine transform corrects it adequately by removing the bluish effect. A still more contrasted result is obtained by saturating only 1%.
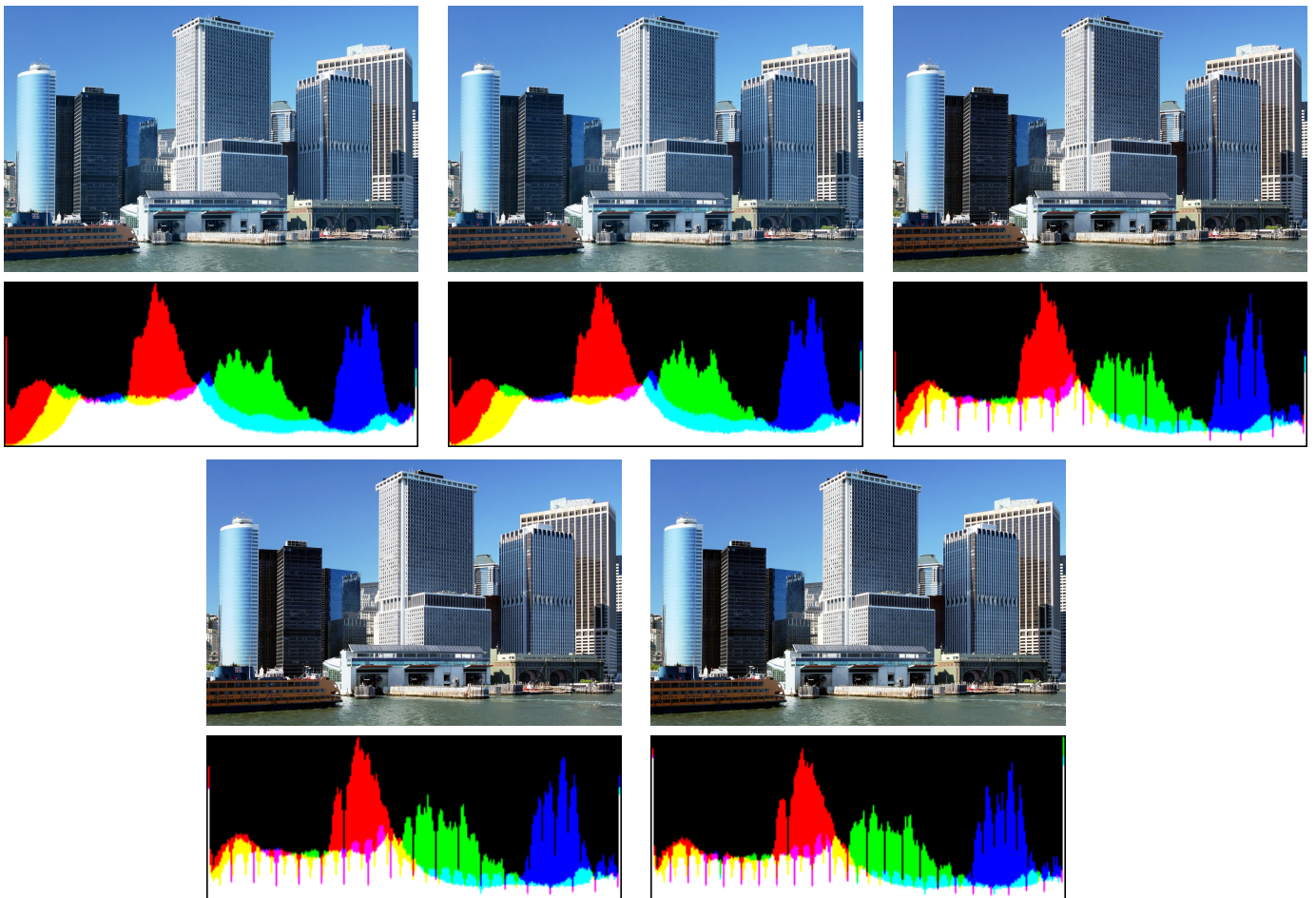
Figure 4: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. As in the previous example, this image in completely unnatural blue light is often used to illustrate color balance, or color contrast adjustment algorithms. A trivial affine transform corrects it adequately by removing the bluish effect. A still more contrasted result is obtained by saturating only 1%.

Figure 5: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. Even a good quality image can benefit from a moderate 1% color balance. A contrast improvement is noticeable when switching between the 0% and 1% versions.
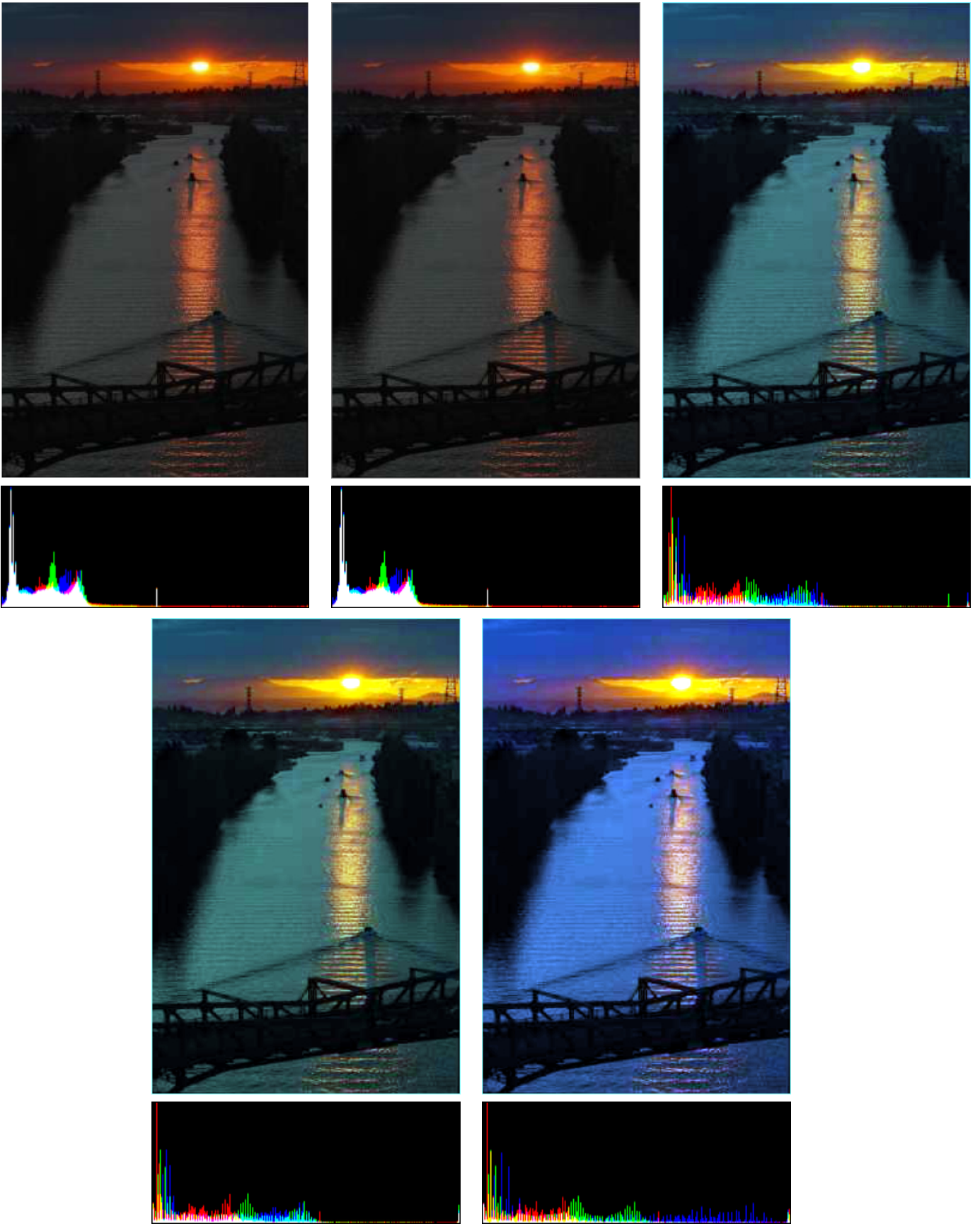
Figure 6: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. There is no real good solution for this sunset image. The colors are strongly blue/orange and will stay so. By pushing too far the saturation (3%), the orange pixels diminish and a completely unnatural blue color is created.
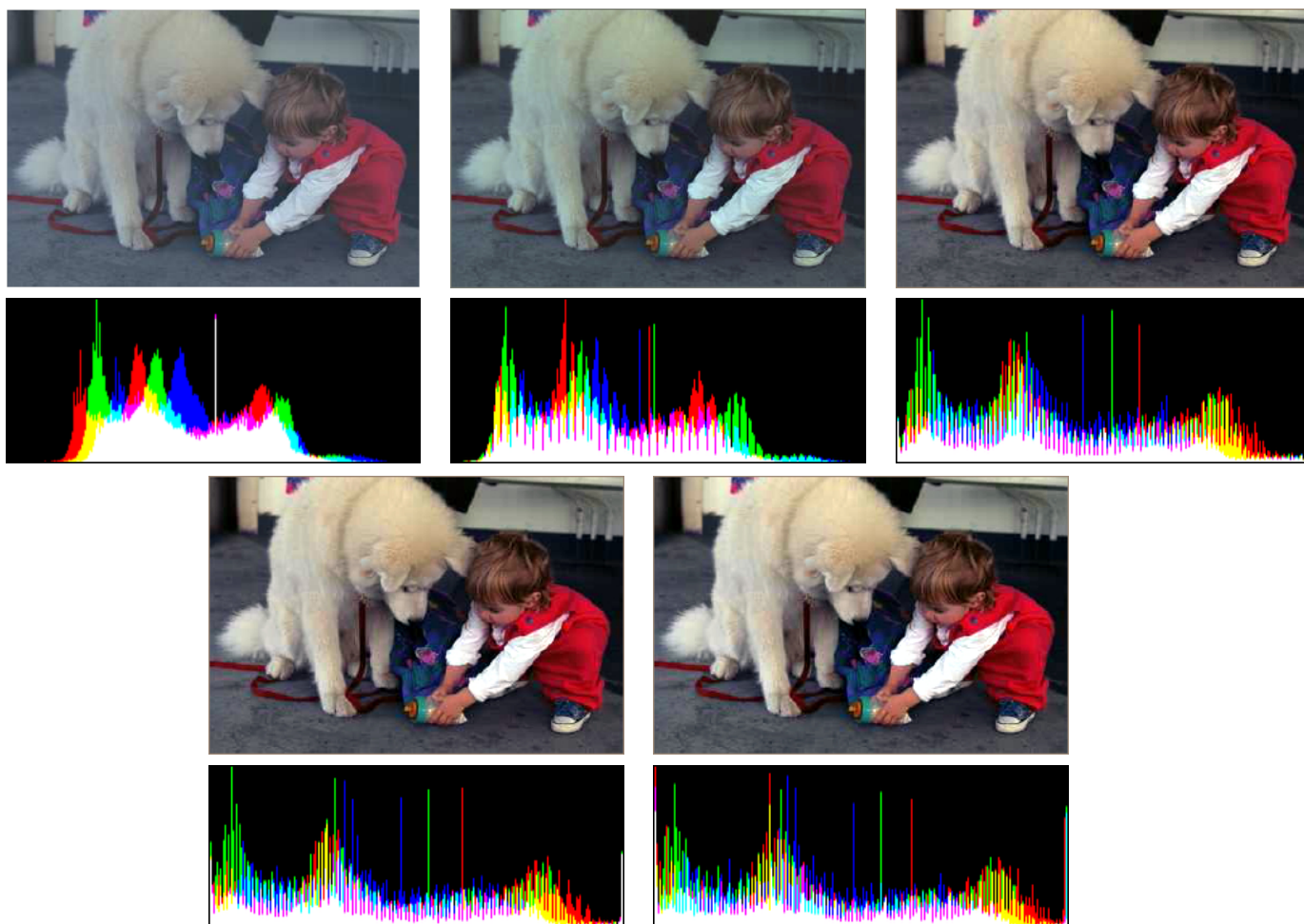
Figure 7: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. This image has been used in recent papers on color perception theory (Retinex).

Figure 8: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. This image, as the one in the previous figure, has been used in recent papers on color perception theory (Retinex).

Figure 9: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. This image, as the ones in the previous figures, has been used in recent papers on color perception theory (Retinex).

7.1 Examples on Gray-scale Images

We present two examples on grayscale images. The algorithm is the same, since the three channels are equal. In the first image (Figure 10) we observe a small improvement of the image. In the second (Figure 11), the improvement is more significant, but we lose the little details in the image.

Figure 10: Top row: left, original image; right, result of the algorithm with $s = 3\%$. Bottom, intensity histograms of the above images.
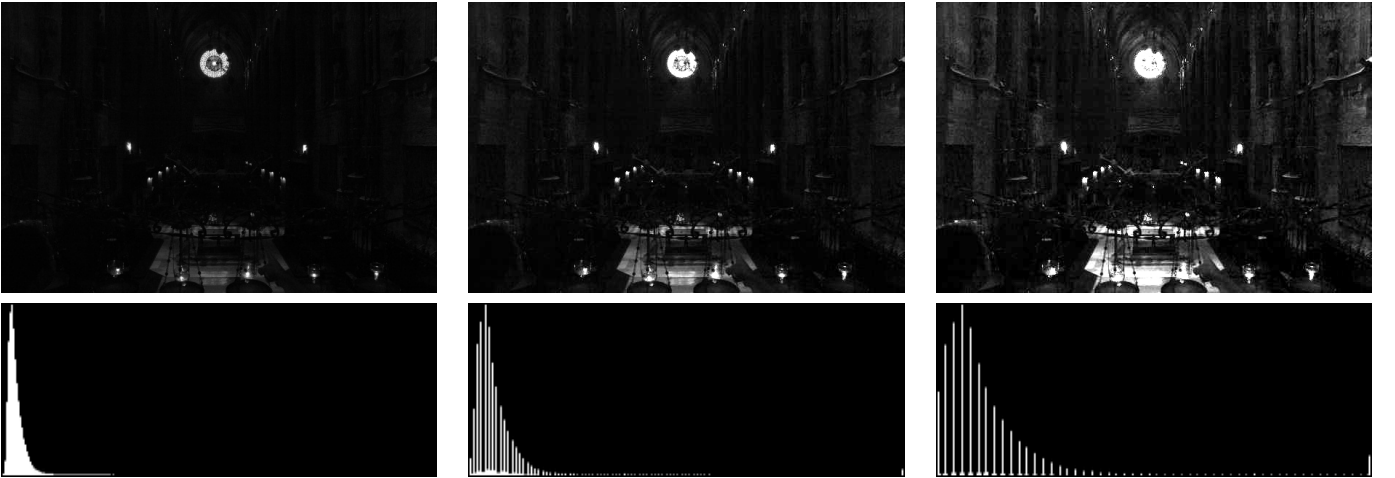

Figure 11: Top row: left, original image; center, result of the algorithm with $s = 1\%$; right, result of the algorithm with $s = 3\%$. Bottom, intensity histograms of the above images.

7.2   Examples with Little or no Improvement

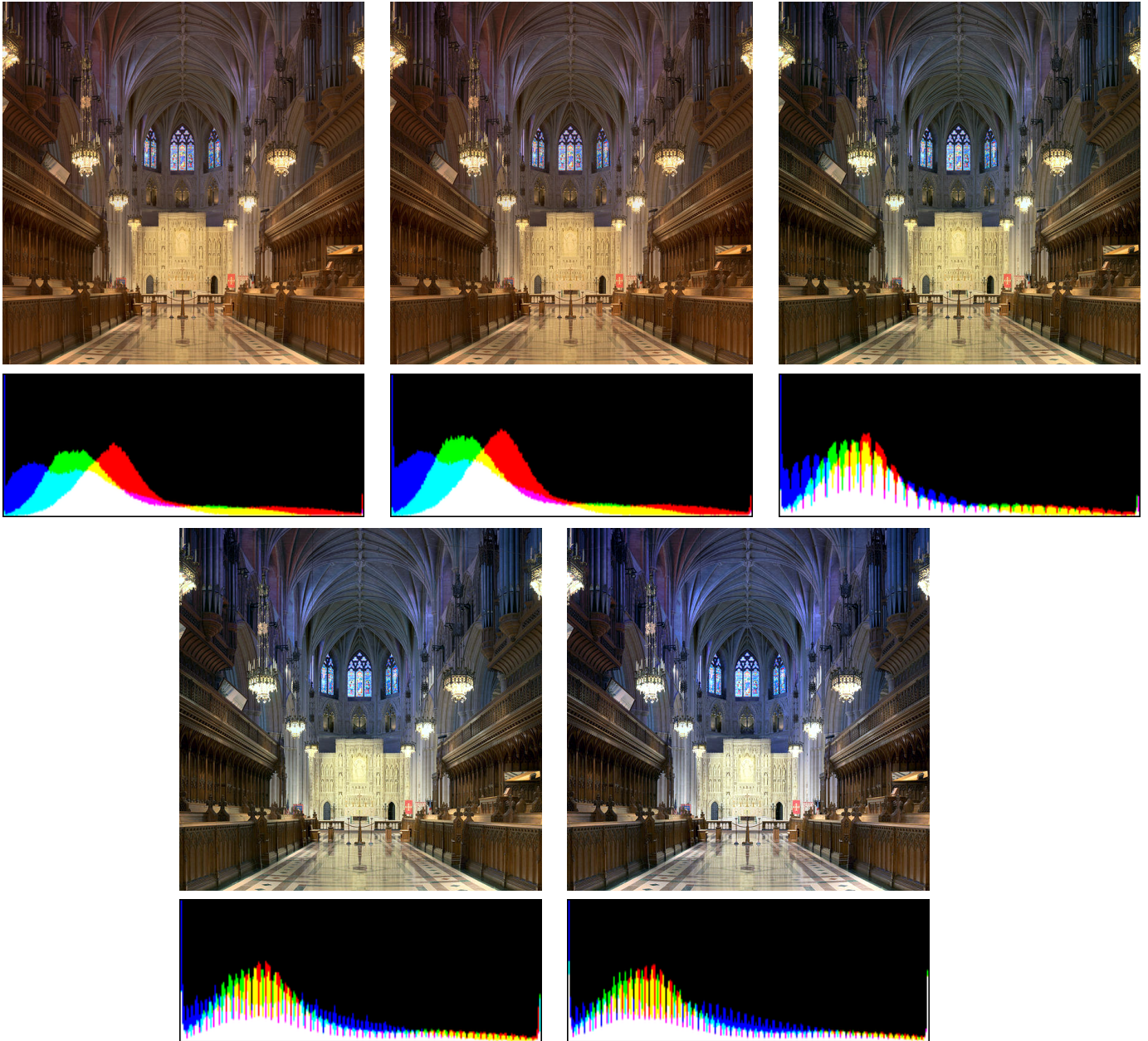Figures 12 to 14 show examples where little improvement is achieved by using the simplest color balance algorithm.

Figure 12: Top row, from left to rigth: original image, results of the algorithm with $s = 0\%$ and $s = 1\%$. Third row, results of the algorithm with $s = 2\%$ and $s = 3\%$. Below each image its RGB histogram is displayed. Here, it is probably best not to apply any color balance: the colors become quickly unearthly.
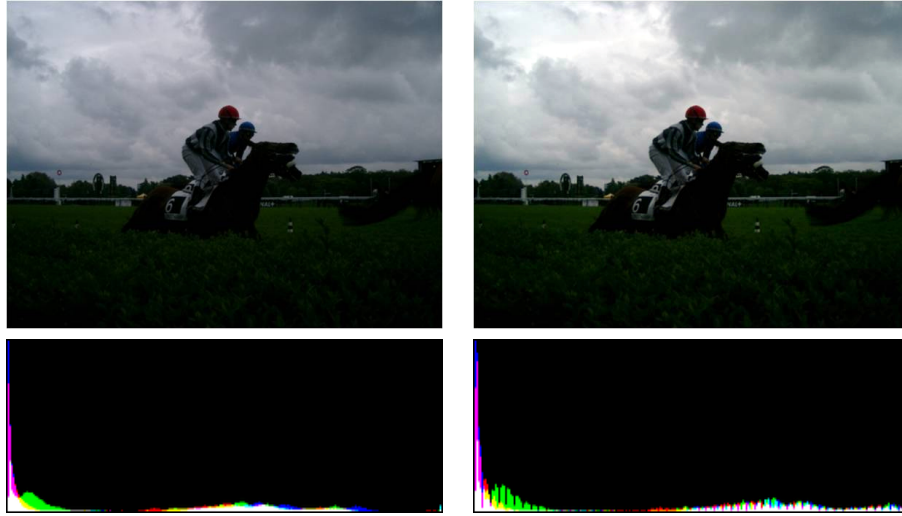
Figure 13: Top row: left, original image; right, result of the algorithm with $s = 3\%$. Bottom, RGB histograms of the above images. The back-lighting problem has no simple solution.
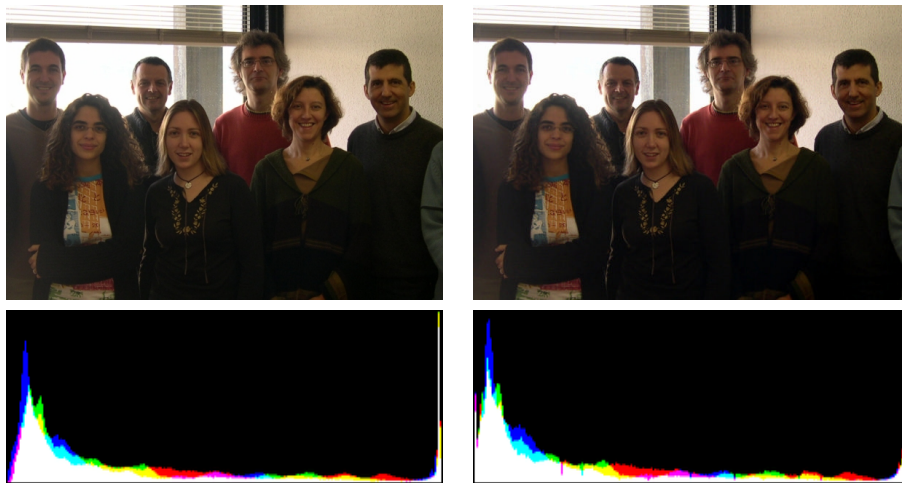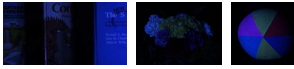

Figure 14: Top row: left, original image; right, result of the algorithm with $s = 3\%$. Bottom, RGB histograms of the above images. As in the previous image, the back-lighting problem has no simple solution.

Image Credits

 Courtesy Philip Greenspun[6]

 Kobus Barnard, SFU Computational Vision Laboratory[7]

 Noclip, Wikimedia Commons[8], public domain

 CC-BY Ana Belén Petro

 Frank Gualtieri, Wikimedia Commons[9], public domain

 Daniel Schwen, Wikimedia Commons[10], GFDL/CC-BY-SA

 Drew Streib, Flickr[11], CC-BY-NC

 standard test image

References

[1] Marc Ebner, "Color Constancy", John Wiley & Sons, 2007, p. 104. http://dx.doi.org/10.1002/9780470510490

---

[6]http://philip.greenspun.com/
[7]http://www.cs.sfu.ca/~colour/data/
[8]http://commons.wikimedia.org/wiki/File:National_Cathedral_Sanctuary_Panorama.jpg
[9]http://commons.wikimedia.org/wiki/File:Lily-M7292-As-shot-and-manual.jpg
[10]http://commons.wikimedia.org/wiki/File:Staten_Island_Ferry_terminal_crop.jpg
[11]http://www.flickr.com/photos/dtype/145118964/