Real-time tone mapping on GPU and FPGA

Raquel Ureña, Pablo Martínez-Cañada, Juán Manuel Gómez-López, Christian Morillas, Francisco Pelayo

EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

Real-time tone mapping on GPU and FPGA

Raquel Ureña*, Pablo Martínez-Cañada, Juán Manuel Gómez-López, Christian Morillas and Francisco Pelayo

Abstract

Low-level computer vision algorithms have high computational requirements. In this study, we present two real-time architectures using resource constrained FPGA and GPU devices for the computation of a new algorithm which performs tone mapping, contrast enhancement, and glare mitigation. Our goal is to implement this operator in a portable and battery-operated device, in order to obtain a low vision aid specially aimed at visually impaired people who struggle to manage themselves in environments where illumination is not uniform or changes rapidly. This aid device processes in real-time, with minimum latency, the input of a camera and shows the enhanced image on a head mounted display (HMD). Therefore, the proposed operator has been implemented on battery-operated platforms, one based on the GPU NVIDIA ION2 and another on the FPGA Spartan III, which perform at rates of 30 and 60 frames per second, respectively, when working with VGA resolution images (640 × 480).

Keywords: reconfigurable hardware, graphics processor, real-time system, low-vision aid, tone mapping, resource-constrained platforms

1. Introduction

Luminance levels can change dramatically over time and depending on the place. The average luminance in an outdoor scene can be 100 million times greater during the day than at night, and in the same scene the range of luminance can also vary with ratios on the order of 10,000:1 from highlights to shadows [1].

The human visual system is able to capture a wide range of light levels, and it functions across the changes in luminance employing diverse adaptation mechanisms. Some of them include the pupil, the rod, and the cone receptors. As a result, humans can recognize the details clearly in both dark and bright regions in the same scene. However, vision is not equally good under all conditions. Particularly, the elderly and those who suffer from visual disorders may be profoundly impaired by the low intensity, high dynamic range (HDR), and rapidly changing illumination conditions we often experience in our daily live as it is stated by Irawan et al. [1].

The human visual system can properly recognize details in both dark and bright regions in a scene, while the image captured by conventional digital cameras may be either too dark or too bright to present details [2]. This is due to the limited dynamic range of digital devices. Hence, some image-processing techniques must be applied to enhance these images and to map them on displays with a limited dynamic range.

In this article, we explain two parallel implementations of a new tone mapping operator (TMO) on portable and resource-limited devices based on GPU and on FPGA architectures. With these implementations we aim to obtain a new low-vision aid which seeks to accurately represent in a HMD images captured under non-uniform illumination environments and with sudden changes in the illumination conditions.

In the following sections, we review the properties of some of the most relevant TMOs, and their real-time implementations. Then, we briefly describe the new operator explaining its main advantages. In Sections 4 and 5, we focus on its implementation taking advantage of the parallelism provided by GPU- and FPGA-based platforms to achieve real-time processing when working with portable and resource-constrained devices.

Then, we show the obtained results explaining the main advantages and drawbacks of each implementation to understand the trade-off between the flexibility but relatively low frequency of an FPGA and the high frequency and fixed architecture of the GPU.

In the literature, we can find several GPU versus FPGA comparative works, for instance, in [3] five relatively simple image processing algorithms implemented on a Xilinx Virtex 4 FPGA and a GeForce GTX 7900

* Correspondence: ruperez@atc.ugr.es
Department of Computer Architecture and Technology, CITIC-ETSIIT, University of Granada, Granada, Spain

GPU are examined. On the other hand, the work developed by Pauwels et al. [4] compare both platforms using medium to highly complex vision algorithms that stretch the FPGA to its limits.

In those works, they employ high-performance FPGA and GPU devices, whereas, with our contribution, we aim to extend the state-of-the-art by comparing two resource-constrained GPU and FPGA implementations of a low level pixel-wise image processing algorithm.

2. Background

The development of techniques for HDR image capture and synthesis has made tone mapping an important issue in computer graphics. The fundamental problem is how to map the large range of intensities found in an HDR image into the limited range supported by a conventional display device.

Different TMOs have been introduced in the research literature which can be classified into two broad categories: global and local operators.

Global operators apply a single mapping function to all pixels of the image, whereas local operators modify the mapping depending on the characteristics of different areas of the image.

Some examples of well-known global operators are the one proposed by Drago et al. [5] and by Ward-Larsen et al. [6]. The former is based on logarithmic compression of luminance values imitating the human visual system response to light. The logarithmic bases employed are modified using a bias power function which produces a good preservation of the details and the contrast. The TMO proposed by Ward-Larsen et al. also performs logarithmic compression of the luminance as well as an iterative histogram adjustment constrained in slope by the human threshold versus intensity.

The main drawbacks that these algorithms present are that they require the calculations of global statistic quantities, the maximum, and the log average. These global calculations are very time-consuming when working with parallel architectures such as the GPU architecture. Moreover, they require the calculation of the logarithm of the luminance which demands a great deal of memory resources in the FPGA implementation, since a look-up-table (LUT) is required. Hence, these TMO are not suitable for being implemented in resource-constrained devices as we aim to do. As it is stated in [7], to achieve real-time performance more powerful GPUs are required.

One of the most relevant local TMO that can be found in the literature is the Mutiscale Retinex with Color Restoration (MSRCR) [8] which performs dynamic range compression, color constancy, and rendition. This operator brightens up dark areas of the image without saturating the areas of good contrast, preserving the chromatic component. It performs logarithmic range compression as well as combination of various Gaussian-based scales to preserve the color.

Another local TMO is the proposed by Hu et al. [2], which employs bilateral filters and divides the image in different regions according to the global histogram and then each region is enhanced according to its individual properties.

The algorithm proposed by Horiuchi and Tominaga [9] takes advantage of both global and local operators by performing global enhancement employing a model of photoreceptor adaptation based on the general level of luminance and local adaption inspired in the MSRCR.

We have decided to develop a new TMO since most of the TMOs that can be found in the literature require time- or memory-consuming operations such as global statistical calculations, iterative processing, or logarithmic compression of the dynamic range. Therefore, they are not appropriate to be implemented on resource-constrained systems as we aim to do. Actually most of the TMOs mentioned above have been implemented in high-end GPUs by Zhao et al. [7] to achieve real-time image performance. It is also true when working with FPGA-based devices. As shown in [10] existing hardware implementations require high-end FPGAs in order to get real-time operation. Moreover, the existing operators are not able to properly mitigate glares as well as enhance dark areas in the same scene while enhancing the details of the image such as the edges. Our system is required to attenuate glares in the images, since low-vision-affected persons present difficulties in their adaptation mechanisms to illumination changing conditions.

In the next section, we explain the details of the new contrast-enhancement technique which is aimed to fulfill the specific requirements of our target low-vision application without carrying out complex and time-consuming operations.

3. The new operator

The proposed system takes advantage of both global and local approaches. On the one hand, it performs global contrast enhancement to brighten up the areas of the image of poor contrast/lightness as well as preserving the regions of good contrast and without altering the color. On the other hand, it carries out local image processing to mitigate too bright regions and glares as well as to preserve and enhance the details of the image. The glare mitigation is one of the novelties of our approach, and it is specially aimed to the low-vision-affected persons who have difficulties to visualize properly and scene with too bright regions.

The global enhancement is based on the histogram adaptation of the brightness channel (V), when working in the HSV color space to produce images that accurately represent the threshold visibility of the scene

features. HSV color space is employed since it is the most similar to the way the human brain tends to organize the colors employing three components: the Hue (H) which is the chromatic component, the Saturation (S) which indicates how pure a color is and the brightness (V) which is a measure of the intensity of light.

The local enhancement is based on a more general retina-like processing scheme previously used in Retiner to extract the main regions of the scene and to produce an array of spike events for neural stimulation [11,12] and in Vis2Sound to detect the main objects of the scene producing a spatialized sound to indicate their location [13]. In this study, this procedure enhances the details of the scene as well as mitigates glares, but we do not perform any kind of sensorial transduction or neural encoding employing its output.

Using as inputs the Red (R), Green (G), and Blue (B) color channels and the intensity channel, obtained as an average of the R, G, and B channels, we can design a set of spatially opponent filters to model the function of retina bipolar cells. This opposition between the center and the periphery of the receptive field of these cells can be modeled with a difference-of-Gaussians filter (DoG), according to Equation (1):

$$\mathrm{DoG} = G_{\sigma 1} - G_{\sigma 2} = \frac{1}{\sqrt{2\pi}}\left[\frac{1}{\sigma_1}\exp\left[-\frac{x^2+y^2}{2{\sigma_1}^2}\right] - \frac{1}{\sigma_2}\exp\left[-\frac{x^2+y^2}{2{\sigma_2}^2}\right]\right] \quad (1)$$

where Gaussians $G_{\sigma 1}$ and $G_{\sigma 2}$ are applied to different color channels or combinations, and σ_1 and σ_2 are the standard deviation of the Gaussian Mask. Typical values for σ_1 and σ_2 are 0.9 and 1.2, respectively, when the size of the filtering mask is 7 × 7. Incrementing the value of σ has the effect of increasing the receptive field.

This vision model performs a linear combination of three DoG filtering operations: two of them enhancing color-opponent contrast (magenta versus green, and yellow versus blue), and an additional one enhancing the edges of the scene. Equations (2) to (5) describe this procedure:

$$\mathrm{magenta\_vs\_green} = \mathrm{DoG}\left(\frac{\mathrm{R}+\mathrm{B}}{2}, \sigma_1, \mathrm{G}, \sigma_2\right) \quad (2)$$

$$\mathrm{yellow\_vs\_blue} = \mathrm{DoG}\left(\frac{\mathrm{R}+\mathrm{G}}{2}, \sigma_1, \mathrm{B}, \sigma_2\right) \quad (3)$$

$$\mathrm{achromatic\_channel} = \mathrm{DoG}\left(\frac{\mathrm{R}+\mathrm{B}+\mathrm{G}}{3}, \sigma_1, \frac{\mathrm{R}+\mathrm{B}+\mathrm{G}}{3}, \sigma_2\right) \quad (4)$$

$$\mathrm{retina\_output} = \mathrm{w1}\cdot\mathrm{magenta\_vs\_greem} + \mathrm{w2}\cdot\mathrm{yellow\_vs\_blue} + \mathrm{w3}\cdot\mathrm{achromatic\_channel} \quad (5)$$

where w1, w2, and w3 are the weightings factors used to combine the output from the three channels which are obtained according to Equations (6) to (8).

$$w3 = \%\mathrm{darkpixels} \quad (6)$$

$$w1 = 0.6 \cdot (1 - w3) \quad (7)$$

$$w2 = 1 - w1 - w3 \quad (8)$$

We have chosen this combination of the color input channels according the way the human retina combines the signals from the three cone types, two chromatic, and one achromatic system [14]. The system sets up the weighting factors according to the percentage of dark pixels in the image, so that if the scene is too dark the system mainly enhances the edges of the image, as the human visual system does taking into account that color sensitivity is reduced in dark environments. The percentage of dark, medium, and bright pixels is calculated according to Equations (9) to (14). These parameters make possible to adjust the enhancement automatically according to the lighting conditions.

$$\%\mathrm{dark\ pixels} = \frac{\mathrm{Number\ of\ dark\ pixels}}{\mathrm{Total\ number\ of\ pixels}} \quad (9)$$

$$\%\mathrm{mediumpixels} = \frac{\mathrm{Number\ of\ medium\ pixels}}{\mathrm{Total\ number\ of\ pixels}} \quad (10)$$

$$\%\mathrm{bright\ pixels} = \frac{\mathrm{Number\ of\ bright\ pixels}}{\mathrm{Total\ number\ of\ pixels}} \quad (11)$$

$$\mathrm{dark\ pixel} \in \left[\mathrm{minValue}, \frac{1}{3}\cdot\mathrm{maxValue}\right] \quad (12)$$

$$\mathrm{mediumpixel} \in \left(\frac{1}{3}\cdot\mathrm{maxValue}, \frac{2}{3}\cdot\mathrm{maxValue}\right] \quad (13)$$

$$\mathrm{brightpixel} \in \left(\frac{2}{3}\cdot\mathrm{maxValue}, \mathrm{maxValue}\right] \quad (14)$$

where minValue is the minimum possible value of brightness and maxValue is the maximum possible value. In our case, minValue = 0 and maxValue = 255.

Finally, the whole system performs a linear combination between the processed brightness channel (V2), the original brightness channel (V), and the retina output obtaining the final brightness channel, according to Equation (15).

$$V_{\mathrm{final}} = \left(1 - \mathrm{retina}_{\mathrm{weight}}\right)\cdot\left(\beta V_2 + (1-\beta)\,V\right) + \mathrm{retina}_{\mathrm{weight}}\cdot\mathrm{retina\_output} \quad (15)$$

where

$$\mathrm{retina}_{\mathrm{weight}} = \begin{cases} 0.5, \text{ if } \%\mathrm{darkpixels} \geq 0.5 \text{ or } \%\mathrm{brightpixels} \geq 0.5 \\ \%\mathrm{brightpixels}, \text{ otherwise} \end{cases}$$

Typical values for β are between 0.2 and 0.6. The Hue and the Saturation components remain unaltered. Finally, a conversion to the RGB color space is carried out to visualize the image on the HMD. A block diagram of the new TMO is depicted in Figure 1.

To finalize this section the main novelties and advantages of our proposed operator are outlined:

- Tone mapping and contrast enhancement using histogram adaptation of the brightness channel without

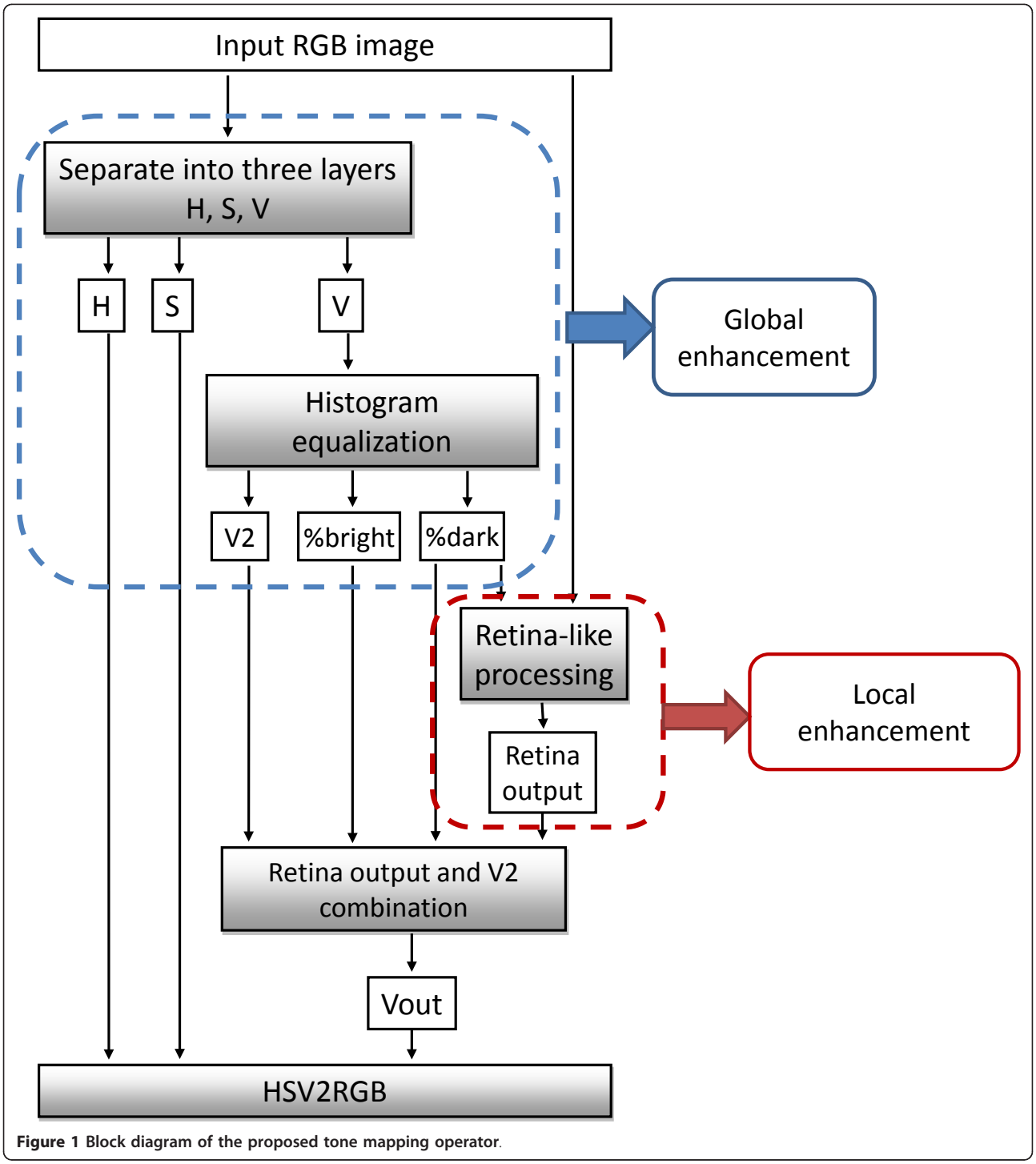

Figure 1 Block diagram of the proposed tone mapping operator.

requiring logarithmic compression of the dynamic range and other time-consuming computations.
- Effective mitigation of glares using a bio-inspired processing which also preserves and enhances the details of the image such as the edges.
- Automatically adjustment of the processing parameters according to the illumination conditions.

Such pixel-wise operator inherently takes advantage of massively parallel architectures like GPU and FPGA. In Sections 4 and 5, the parallel implementation of this operator in different platforms is explained. Section 6 compares the output from the new operator with the output provided by some of the most relevant TMOs, to show that the proposed operator provides similar or even better results than the others without requiring time-consuming operations and providing new facilities such as glare mitigation and edge enhancement, as required in the scope of the target application.

4. GPU implementation

The requirements for this application as portability, limited power consumption, and real-time performance led us to consider the GPU NVIDIA ION2 [15] as a good option to design and develop the system. The NVIDIA CUDA API [16] is used to parallelize the operator since this GPU is supported. More information about the configuration interface of this aid device and other available image enhancements are explained in [17].

Some other aid systems use FPGA or DSP devices since they provide a high computation capability in a small and low power device. However, the selected GPU has 16 processors and it is already available in a lightweight netbook with sufficient battery autonomy (about 4 h).

Moreover, the system takes advantage of the Intel ATOM N450 processor, integrated in the netbook ASUS EEPC 1201 PN [18], which is faster than FPGA built-in processors, such as PowerPC. Furthermore, the GPU technology provides more flexibility to develop and customize dynamically the application.

Our target GPU consists of two streaming multiprocessors. Each streaming multiprocessor has one instruction unit, eight stream processors (SPs), and one local memory (16 KB), so it has 16 SPs in total. The eight SPs in the same streaming multiprocessor are connected to the same instruction unit, so they execute the same instruction stream on different data (called thread). In order to extract the maximum performance of SPs by hiding memory access delays, we provide four threads for each SP, which are interleaved on the SP. Therefore, at least 32 threads for each streaming multiprocessor are required.

In our case, the GPU requires full utilization to take advantage of the hardware's latency hiding design. Another way to achieve the maximum performance is using fewer, but more work-intensive threads and relying on instruction-level parallelism as it is stated in [19].

To optimize the use of the available multiprocessors, the parameters to be determined are the number of threads and the shared memory required per block. Figure 2 summarizes the computation flow followed by the video streaming from the image capture to the enhanced image output. As we can observe, our GPU implementation of the TMO is comprised of several CUDA kernels. The general structure of the implemented CUDA modules is depicted in Figure 3.

To accurately size the kernels we have used the CUDA Occupancy Calculator tool that shows the occupation of the multiprocessor's cache and its percentage of utilization [16]. The thread block size is chosen in all cases so that multiprocessor occupancy is 100%. The size of the GRID (number of processing blocks to be executed by the kernel) is dynamically set according to the size of the image. The streaming multiprocessors are connected to large global memory (512 MB in ION2), which is the interface between the CPU and the GPU. This DRAM memory is slower than the shared memory; therefore, before starting the computations, all the threads of a block load the required image fragment in the shared memory.

Depending on how the data are encoded in the GPU global memory, each thread can load one element if working with 4-byte datum or 4 data if working with 1-byte datum. The global memory accesses of the GPU for both reading and writing are done so that in one clock cycle all the threads of a warp (L) access to $4{\cdot}L$ bytes of RAM, where L is equal to 32 in CUDA Compute Capability 1.2 GPUs.

Before turning to the processing stage all the threads of the processing block have to wait in a barrier to ensure that all of them have loaded its corresponding data. After the calculation step may be a second stage of synchronization of the block threads before writing to the GPU global memory.

Since we are unable to connect the camera directly to the GPU, image transfers to and from the GPU via the PCI-Express bus are required. The interface between the host and the GPU global memory is the bottleneck of the application so as it is depicted in Figure 2, the data that transfer between the host and the device have been minimized. Furthermore, each image data are encoded as 1-byte unsigned integer. Therefore to encode a color pixel 3 bytes are required. When more precision is needed a conversion to floating point is done once the image is stored in the GPU global memory, exploiting the parallelism provided by the GPU.

4.1. GPU implementation of the retina model

In order to optimize the spatial filtering process, the system takes advantage of the linear property of the

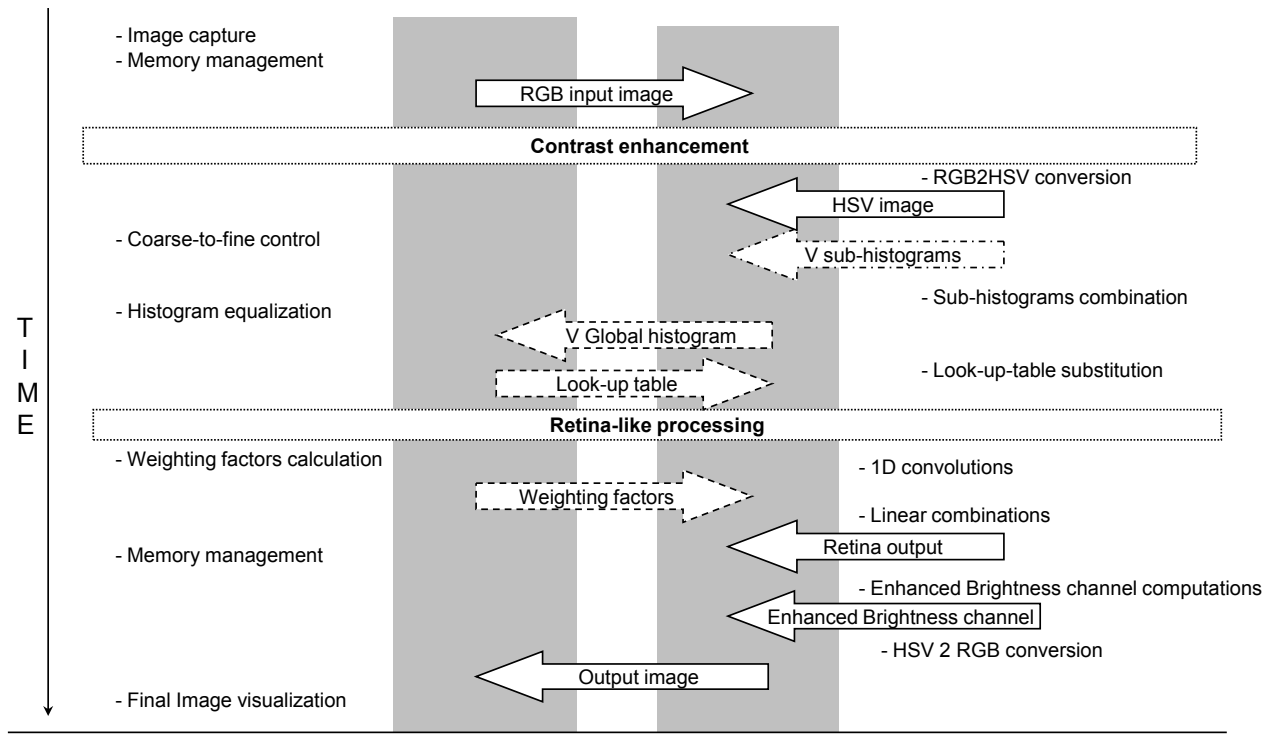

Figure 2 System overview illustrating the sequence of processes running on the CPU and GPU. Solid and dashed arrows depict large and small data transfers, respectively.

convolution operator. Therefore, we can reduce the processing only to the convolution of each color channel with two different Gaussian masks. Then, these filtered channels are linearly combined. Equations (16) to (18) describe mathematically this simplification:

$$\text{Magenta\_vs\_Green} = \frac{1}{2}\left(G_{\sigma_1}(R) + G_{\sigma_1}(B)\right) - G_{\sigma_2}(G) \quad (16)$$

$$\text{Yellow\_vs\_Blue} = \frac{1}{2}\left(G_{\sigma_1}(R) + G_{\sigma_1}(G)\right) - G_{\sigma_2}(B) \quad (17)$$

$$\text{Achromatic\_output} = \frac{1}{3}\left(G_{\sigma_1}(R) + G_{\sigma_1}(G) + G_{\sigma_1}(B)\right) - \frac{1}{3}\left(G_{\sigma_2}(R) + G_{\sigma_2}(G) + G_{\sigma_2}(B)\right) \quad (18)$$

Moreover, the GPU implementation of the retina-like processing relies extensively on 2D separable convolution operations that are highly data-parallel and thus well matched to the GPU architecture. Therefore, the computational complexity is reduced from $O(n^2)$ to $O(2n)$ being $n \times n$ the filter mask size.

In order to carry out these convolutions in real-time we have developed two CUDA filtering kernels, one for the rows and another one for the columns. Both modules are very similar so we detail only the one for rows.

The convolution operation requires a neighborhood with the same width that the filter mask to calculate the result for each pixel. So, each thread transfers one datum from the global memory to the shared memory. In order to get the maximum precision and to avoid bank conflicts in shared memory, these data are stored as floating point data. Therefore, there are n-1 threads per block that only load data, but do not calculate any filtered pixel. So, as to not waste too many threads in the loading stage, the block size must be large enough compared to the filter mask width. In this case, the block size is set to 1×128, and the filter width is 7, so only 6 threads are wasted per processing block. The block width is set to 128 to achieve the required alignment when accessing to global memory, and also to optimize the multiprocessor utilization.

When all the data are stored in the shared memory each thread multiplies the filter coefficients, stored in the constants memory, with the corresponding pixel and its neighborhood. Then, the result is stored in the global memory. Each thread repeats this procedure twice, once for each mask. Therefore, we carry out the two rows filtering with just one read access to the global memory.

The column filters are computed in a similar way, taking as input the previously filtered images.

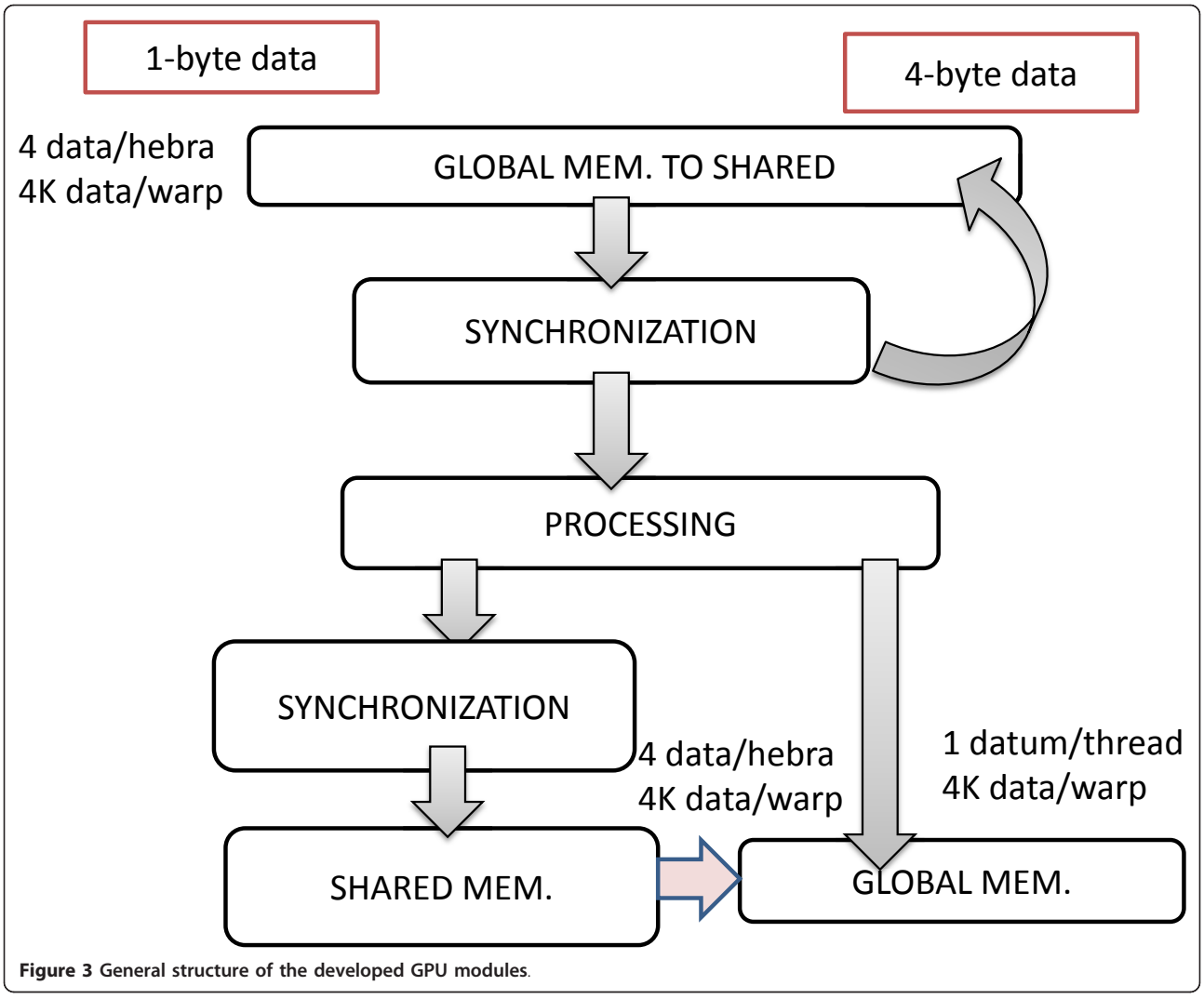

Figure 3 General structure of the developed GPU modules.

Once the filtering process is finished we just need to linearly combine partial results as we have explained earlier, exploiting the parallelism provided by the GPU to obtain a gray level image. This image will be used to modulate the degree of enhancement applied to the brightness channel.

4.2. HSV conversion and brightness equalizer

This kernel uses thread blocks with size 256 (64 rows and 4 columns) and 4 KB of shared memory to store the necessary pixels and the histogram of the block. According to the NVIDIA CUDA Occupancy calculator, the occupancy of the multiprocessor is 100% and the maximum number of active blocks per multiprocessor is 4.

First of all, each block loads 1024 pixels in the GPU shared memory, each thread loads four data. Once all the data are stored in shared memory each thread computes three converted pixels and stores them in shared memory. During the process, each block calculates its own histogram of the brightness channel. To avoid mistakes when calculating the histogram we use the *atomicAdd* operation available in GPUs with CUDA Compute Capability 1.2. Finally, we transfer the HSV pixels to the GPU global memory and we merge each sub-histogram in a global histogram using atomic operation. This way the brightness histogram is computed at the same time the RGB to HSV conversion is performed requiring only an additional transfer of each block histogram from shared to global memory. Then, the histogram equalization is performed using a LUT substitution obtaining a new brightness channel (V2 in Figure 1), which is linearly combined with the output from the retina-like processing.

Finally, de HSV to RGB conversion is performed in a similar way, and the RGB resulting image is transferred to the CPU.

5. FPGA implementation

Our target FPGA platform is the SB video-processing mobile platform [20], which includes a Xilinx Spartan 3 XC3S2000 FPGA with 2 million gates, 36-Mbit SRAM memory for data exchange, and all the interfaces required for the video input channel and the video output for HMD. We have chosen this platform because of its reduced power consumption, and small size. It includes a battery that provides up to 10 h of autonomy.

Figure 4 summarizes the processing carried out over the video frames, from the image grabbing stage to the enhanced image output.

The analog video input is passed to an analog-to-digital converter, which encodes each pixel in YUV format. In order to separate the two different clock domains (the input working at 27 MHz and the VGA output at 40 MHz) we have used a double buffering technique.

So, as to maximize the performance of the system and to exploit the inherent parallelism on the programmable device selected for the implementation we chose a pipelined architecture, able to process a pixel every clock cycle.

As in the previous section, we have implemented two main procedures, the retina-like filtering and the brightness equalization. In the next sections, we explain each of these modules in detail, as well as the HSV conversion module, which requires a specific implementation adapted to the FPGA.

5.1. Implementation of the retina-like filtering

The convolution process requires a set of pixels of its neighborhood to calculate each pixel as we have explained before. To resolve this problem we use the convolution computation architecture proposed by Ridgeway [21], depicted in Figure 5. In order to use 7 × 7 filtering masks we need 7 FIFO buffers that store the first 7 image rows and seven shift registers that are responsible for storing the 49 neighboring pixels for the current convolution. The serial connection of the FIFO memories emulates the vertical displacement of the mask and the transfer of values of the FIFO memories to the shift registers emulates the horizontal scrolling. Then, the accumulators marked as "ACCUM x" add those pixels that are multiplied by the same coefficients of the mask. We have six products as a result of breaking down the process of convolution taking advantage of the linear property of the convolution as we have mentioned before. Then, we have to take into account the weighting to be applied according to the percentage of dark pixels present in the image.

5.2. HSV conversion

The calculation of the Hue component (H) requires complex computation. To avoid that, it has been implemented using a LUT with 215 inputs of 8-bits each one mapped into the FPGA BlockRAM modules. This LUT employs a 36% of the available memory (262 Kbits). Besides, to get the other two color components, saturation (S) and brightness (V), two dividers have been implemented. The design of the dividers is fully pipelined, and they can achieve a throughput of one division per clock cycle. The division of the S component needs a fractional remainder because the minimum of R, G, and B always is equal or less than the sum of them. This fact results in an 18-cycle delay which has to be considered in the H component computation in order to get synchronization.

5.3. Brightness equalizer

To implement the brightness equalizer the whole V color plane, whose size is 640 × 480 pixels, the image is divided in 35 blocks with 100 × 100 pixels and their cumulative distributions are calculated. While the cumulative distributions for the current frame are being computed in parallel, its brightness channel is being equalized using the distributions computed for the previous frame. The distributions computation is performed in five steps, calculating seven distributions in parallel at each step, as image is being scanned. This procedure finishes when the 35 cumulative distribution functions are stored in the RAM memory. To develop the equalizer we rely extensively in the implementations explained in [10].

6. Results

Regarding to the results provided by the proposed operator, Figure 6 shows the output from the different

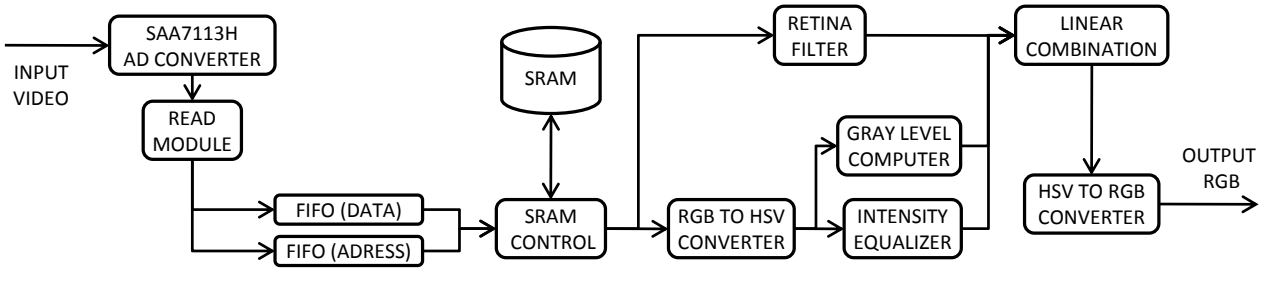

Figure 4 Computation flow for the input images from the camera in the SB platform.

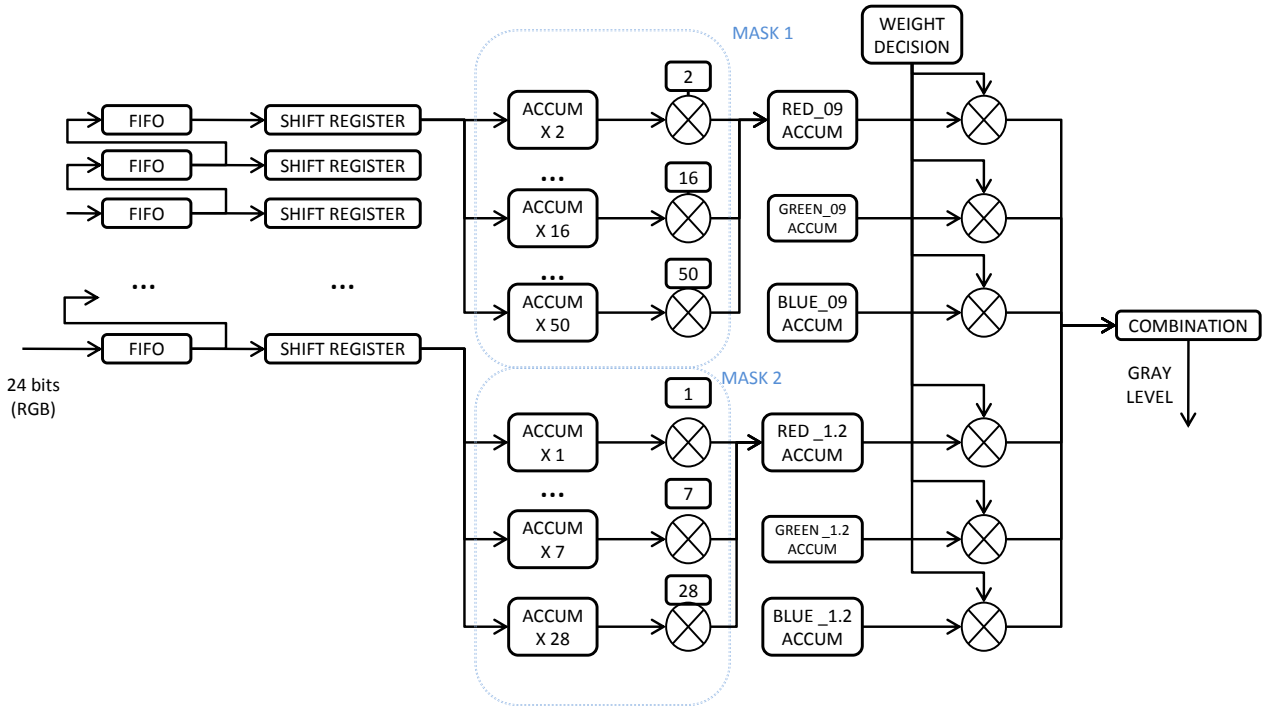

Figure 5 Convolution diagram for the FPGA.

processing stages. Figure 6a shows a dark image [22] in which main features of the scene cannot be appreciated, only the window can be distinguished. Figure 6b shows the output from the retina-like filtering, as the whole image is too dark, the retina output mainly enhances the edges of the main features. Figure 6c shows the output brightness equalization without taking into account the retina output. The final output image is depicted in Figure 6d. As we can observe in this image, main features of the scene can be distinguished clearly whereas the glares and the too bright areas which appear in Figure 6c have been mitigated. Moreover, the colors of the picture do not appear distorted.

Figures 7 and 8 show a comparative of the output of our operator, without taking into account the retina-like processing, and the output from well-known TMOs and contrast enhancement algorithms known as Drago et al. [5], Mantiuk et al. [23], and Reinhard and Devlin [24] operators and Multiscale Retinex [8]. As we have explained previously, our operator has two main stages, one comprising a dynamic range adaptation and contrast enhancement and a second one for glare mitigation and edge enhancement. The later is specially designed for low-vision-affected people; therefore, the output obtained from the combination of both parts is not directly comparable with the output from other TMO operators. For this reason, we have set aside the output from the retina-like processing in this comparative.

In Figure 7a, we can observe the same underexposed image than in Figure 6[22]. Figure 7b-g shows the original image enhanced with different TMO algorithms. In Figure 7b, we can observe perfectly all the elements of the image, actually the proposed operator not only brighten dark region, but also keep details of the landscape out of the window, whereas the output from the other TMOs presents the window overexposed (see Figure 7d-g), and, in some cases, the whole image appears too bright (see Figure 7e, g). Figure 8 shows another comparative example. In this case, the original image presents better signal-to-noise ratio (SNR) than in the previous case, according to Table 1. As we can observe, the proposed algorithm enhances the whole image without saturating the bright regions and preserves the overall level of illumination in medium values, that way all the details can be appreciated without presenting disturbing glares.

Table 1 summarizes the SNR of each of the images presented in Figures 7 and 8.

The value of the SNR has been calculated according to Equation (19):

$$\mathrm{SNR} = 20\log\left(\frac{\mu}{\sigma}\right) \tag{19}$$

Where μ is the average value of the image and σ is the standard deviation. We are working with color

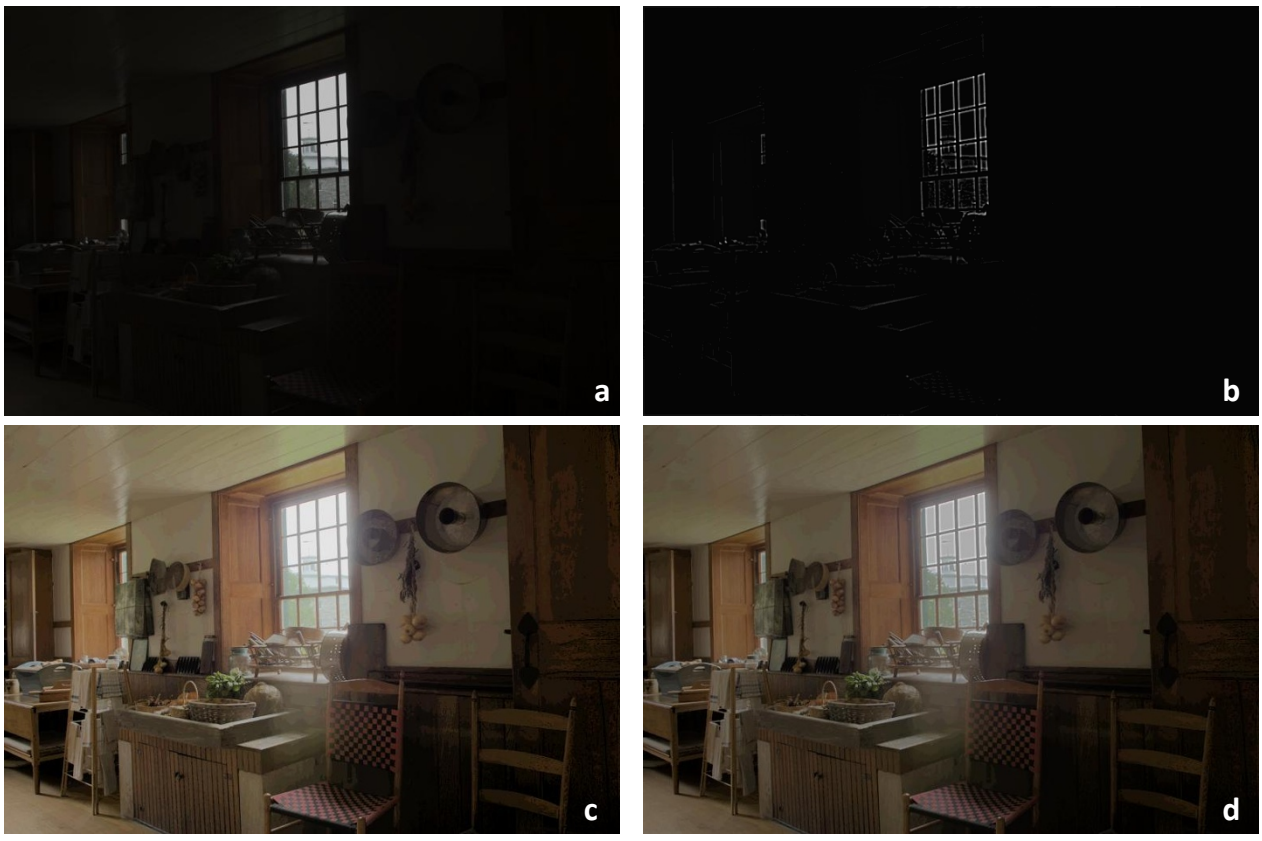

Figure 6 Outputs from the different stages of the proposed operator: (a) original image [22], (b) output from the retina-like filtering, (c) output from the contrast-enhancement processing, and (d) final image.

images so we show in Table 1 the average value of the SNR calculated for each channel.

From these comparisons of the results with different methods we can observe that the proposed algorithm provides an effective improvement of dark images and high-contrast images, without altering color information, preserving the details, and it does not brighten excessively the image. Moreover, it can enhance the image automatically according to the lighting conditions, without requiring the user to set complicated parameters.

According to the measures of the SNR presented in Table 1, our operator is able to increase the SNR with respect to the original image. Moreover, the values of the SNR provided by our algorithm are pretty similar to the values provided by the others TMOs, especially to the Drago operator. This operator is the one which provides more natural scenes and better detail reproduction in dark regions, according to a study performed by Yoshida et al. [25]. In this study, the authors conduct a psychophysical experiment based on a direct comparison between the appearances of real-world HDR images of these scenes displayed on a low dynamic range monitor employing seven well-known TMOs. The human subjects were asked to rate image naturalness, overall contrast, overall brightness, and detail reproduction in dark and bright image regions with respect to the corresponding real-world scene.

Moreover, as it can be observed from Figures 7 and 8, the proposed operator provides better detail reproduction in bright image regions (observe the window in Figure 7a, b).

At this point, we discuss the results obtained from tests regarding the performance of the system using the GPU and the FPGA-based platforms, and also related to the use of resources for the FPGA implementation, and the speed up obtained with respect to a non-parallel CPU implementation.

As we have mentioned before, the complete system have been implemented on a GPU NVIDIA ION2 and on an FPGA Xilinx Spartan 3. The results regarding area occupation and clock frequency for the FPGA implementation are summarized in Table 2.

Table 3 summarizes the performance in frames per second (fps) of the CPU (Matlab code running on a single core), GPU, and FPGA implementations of the new operator and the speed up obtained with respect to the

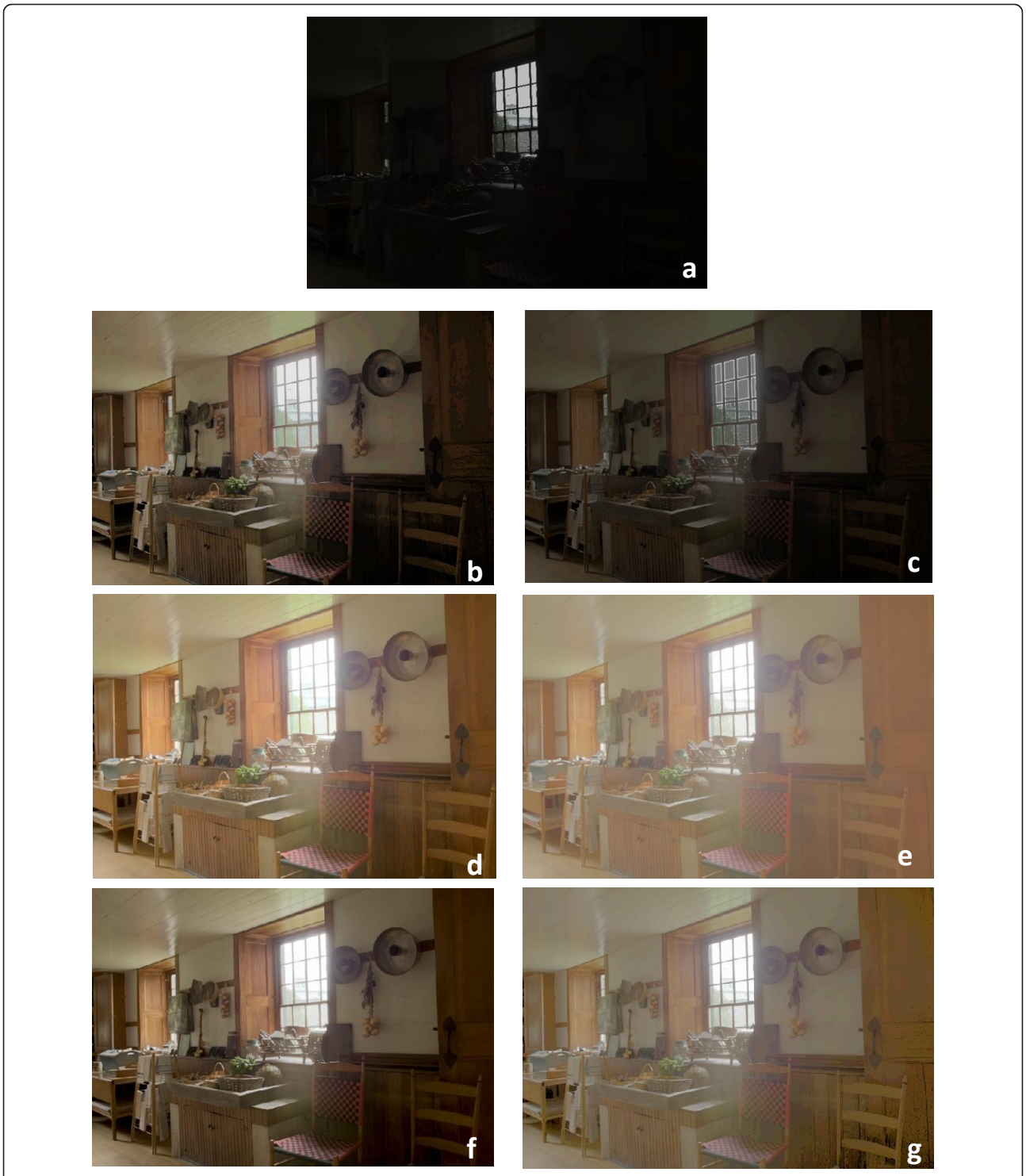

Figure 7 Comparison of the proposed algorithm with different image enhancement methods. **(a)** Original image from [22], **(b)** result of the proposed algorithm without the retina-like filtering, **(c)** result of the proposed algorithm with the retina-like filtering, **(d)** result of the Drago operator, **(e)** result of the Reinhard operator, **(f)** result of the Mantiuk operator, **(g)** result of the Retinex operator.

Figure 8 Comparison of the proposed algorithm with different image enhancement methods. **(a)** Original image from [22], **(b)** result of the proposed algorithm without the retina-like filtering, **(c)** result of the proposed algorithm with the retina-like filtering, **(d)** result of the Drago operator, **(e)** result of the Reinhard operator, **(f)** result of the Mantiuk operator, **(g)** result of the Retinex operator.

Table 1 SNR obtained with different TMOs

	Original (dB)	Proposed operator without the retina-like output (dB)	Proposed operator with the retina-like output (dB)	Drago (dB)	Reinhard (dB)	Mantiuk (dB)	Retinex (dB)
Kitchen scene	-3.28	18.48	16.18	18.18	20.70	12.36	18.52
Car scene	9.53	20.77	18.52	21.01	18.3	13.96	23.2

CPU when working with RGB images with VGA resolution (640 × 480). The CPU used to carry out this tests is the CPU Intel core i7 920 at 2.67 GHz. Both GPU and FPGA implementations reach real-time performance, over 25 fps, obtaining a minimum speed up of 7.5 with respect to the CPU even when using a high-end CPU.

The FPGA performs at a major frame rate than the GPU. This is mainly due to the large delay required to transfer the frame from the CPU memory to the GPU global memory and vice versa (10 ms). Nevertheless further improvement can be achieved by performing the transferences between the CPU and the GPU asynchronously, concurrently with computation.

However, the GPU works in floating point precision, whereas the FPGA uses fixed point since it has no native support for floating point arithmetic. Also the GPU computes the histogram with 256 intensity levels instead of the 64 levels employed by the FPGA, which is limited by routing constrains.

To measure the accuracy of both approaches we have calculated the peak-signal-to-noise ratio (PSNR), of the output image obtained with both, GPU and FPGA, systems with respect to the one obtained with the CPU, according to Equation (20). The resulting image computed with the FPGA obtains a PSNR of 30 dB, whereas with the GPU the value of the PSNR is infinite since the output image is identical to the one obtained with the CPU. The FPGA obtains a lower value for the PSNR as a result of the different algorithmic simplifications that had to be adopted, and the use of fixed point arithmetic.

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\mathrm{MAX}_I^2}{\frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)]^2}\right) \quad (20)$$

Table 2 Area and speed for the whole system on a Spartan 3 XC3S2000

Parameter	Value
Slices	16545 (80%)
LUTs	30086 (73%)
RAMB	39 (97%)
F_{max}	40.25 MHz
MULTs	26 (65%)
BUFGMUXs	7 (87%)
DCMs	2 (50%)

where I stands for the resulting image obtained with the CPU, K is the resulting image obtained with the GPU or the FPGA, m and n are de dimensions of the image and MAX_I is the maximum value that a pixel can reach (255 in our case).

Table 4 details the percentage of the total processing time employed in each of the tasks by the GPU and by the FPGA. In the case of the FPGA, the percentage of time is obtained for each module separately, when the whole system is working, all the tasks are being executed in a pipeline. On the other hand, the GPU employs only a 6% of the processing time in the histogram adjustment, since the histogram calculation is performed in parallel with the RGB to HSV conversion. Moreover, more than 30% of the time is employed in performing image transfers from CPU memory to GPU memory, so further improvement can be achieved performing the memory storage in parallel with the computation. Table 5 summarizes the power consumption, clock frequency, and weight for both systems.

According to the tables presented, we can observe that real-time performance (over 25 fps) is reached with both embedded solutions. Nevertheless the FPGA implementation is an order of magnitude more power efficient than the GPU, although it provides less accuracy in the computations and therefore output images with less PSNR.

On the other hand, the FPGA solution is less weight, whereas the GPU solution is more affordable since its use is widely extended. In the case of the GPU, a fixed architecture is provided and the goal is to obtain its maximum performance, whereas an FPGA design leaves more choices to the engineer. This flexibility of the FPGA comes at the cost of a much larger design time than the GPU and makes tuning the system more difficult than in the case of the GPU.

7. Conclusions

High-end GPUs and FPGAs are suitable for highly parallel complex algorithms such as pixel-wise processing.

Table 3 Performance of GPU and FPGA implementations, frame size 640 × 480

	CPU	GPU	FPGA
Performance (fps)	4	30	60
Speed up	-	7.5	15

Table 4 Percentage of time per frame spent on each processing stage

Processing stage	% time/frame	
	GPU	FPGA
Store image in device memory	15.6	32.54
RGB to HSV conversion	30.2	10.53
Histogram adjustment	6.64	18.37
Retina-like processing	14.22	9.13
Lineal combination	6.01	2.09
HSV to RGB conversion	12.31	27.34
Store image in main memory	15.02	-

Table 5 Comparison of the employed platforms in terms of power consumption, clock frequency, and weight

Platform	Power (W)	Proc. clock (MHz)	Mem. clock (MHz)	Weight (kg)
ASUS EEPC 1201PN [18]	12	450	750	1.45
SB video-processing mobile platform [20]	0.9	40	27	0.5

On the other hand, limited resources GPUs and FPGAs, with less computing capability and reduced power consumption, can compete with other embedded solutions in portable applications which also require image processing parallel computation to achieve real-time performance.

We have presented two implementations of a new TMO on a GPU NVIDIA ION2 integrated in a small size netbook and on a Spartan 3 FPGA-based platform reaching in both cases real-time performance when working with 640 × 480 RGB images. The FPGA implementation provides higher frame rates and less power consumption, whereas the GPU implementation provides more precision in the computation and therefore higher quality output images.

Since both implementations use portable and battery-operated platforms they can be used as low-vision aids specially aimed at visually impaired people, such as those affected by Retinitis Pigmentosa, who present several difficulties to manage themselves in environments where illumination is not uniform or in low illumination environments.

Acknowledgements
This study was supported by the Junta de Andalucía Project P06-TIC-02007, the Spanish National Grants AP2010-4133, RECVIS (TIN2008-06893-C03-02) and DINAM-VISION (DPI2007-61683), the project GENIL-PYR-2010-19 funded by CEI BioTIC GENIL CEB09-0010, and the Special Research Programme of the University of Granada.

Competing interests
The authors declare that they have no competing interests.

Received: 7 April 2011 Accepted: 15 February 2012
Published: 15 February 2012

References

1. Irawan P, Ferwerda JA, Marschner SR: **Perceptually based tone mapping of high dynamic range image streams.** *Paper presented at the Eurographics Symposium on Rendering* Konstanz, Germany; 2005, 231-242.
2. Hu K-J, Lu M-Y, Wang J-C, Hsu T-I, Chang T-T: **Using adaptive tone mapping to enhance edge-preserving color image automatically.** *EURASIP J Image Video Process* 2010, Article ID 137134, 11 (2010). doi:10.1155/2010/137134.
3. Cope B, Cheung P, Luk W, Howes L: **Performance comparison of graphics processors to reconfigurable logic: a case study.** *IEEE Trans Comput* 2010, **59**:433-448.
4. Pauwels K, Tomasi M, Alonso Diaz J, Ros E, Van Hulle MM: **A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features.** *IEEE Trans Comput* .
5. Drago F, Myszkowsky K, Annen T, Chiba N: **Adaptative logarithmic mapping for displaying high contrast scene.** *Paper presented at the Computer Graphics Forum* 2003, **22**:419-426.
6. Ward-Larsen GW, Rushmeier H, Piatko C: **A visibility matching tone reproduction operator for high dynamic range scenes.** *IEEE Trans Vis Comput Graph* 1997, **3(4)**:291-306.
7. Zhao H, Jin X, Shen J: **Real-time tone mapping for high-resolution HDR images.** *Paper presented at the 2008 International Conference on Cyberworlds* Hangzhou, China; 2008, 256-262.
8. Rahman Z, Jobson DJ, Woodell GA, Hines GD: **Image enhancement, image quality, and noise.** *Proc SPIE* 2005, **59070N, doi:10.1117/12.619460.**
9. Horiuchi T, Tominaga S: **HDR image quality enhancement based on spatially variant retinal response.** *EURASIP J Image Video Process* **2010, Article ID 438958, 11 (2010). doi:10.1155/2010/438958.**
10. Reza AM: **Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement.** *J VLSI Signal Process* 2004, **38**:35-44.
11. Pelayo FJ, Romero S, Morillas C, Martínez A, Ros E, Fernández E: **Translating image sequences into spike patterns for cortical neuro-stimulation.** *Neurocomputing* 2004, **58-60**:885-892.
12. Morillas C, Romero S, Martínez A, Pelayo F, Reyneri L, Bongard M, Fernández E: **A neuroengineering suite of computational tools for visual prostheses.** *Neurocomputing* 2007, **70(16-18)**:2817-2827.
13. Morillas C, Cobos JP, Pelayo FJ, Prieto A, Romero S: **VIS2SOUND on reconfigurable hardware.** *2008 International Conference on Reconfigurable Computing and FPGAs* Cancún, México; 2008, 205-210.
14. Sekuler R, Blake R: **Perception.** McGraw-Hill international editions; 1994.
15. 2011, GPU NVDIA ION 2. http://www.nvidia.com/object/picoatom_specifications.html.
16. NVIDIA Corporation: **NVIDIA CUDA C Programming Best Practices Guide 2.3.** 2009.
17. Ureña R, Martínez-Cañada P, Gómez-López JM, Morillas C, Pelayo F: **A portable low vision aid based on GPU.** *Paper presented at First International Conference on Pervasive and Embedded Computing and Communication Systems. PECCS 2011* Vilamoura, Portugal; 2011, 201-206.
18. 2011, Asus EEPC 1201 PN. http://www.asus.com/product.aspx?P_ID=N0JLbhfgdnpw5FaY.

19. Volkov V: **Prologue Quarterly Of The National Archives.** 2010, available at http://people.sc.fsu.edu/~gerlebacher/gpus/better_performance_at_lower_occupancy_gtc2010_volkov.pdf.
20. SB platform: 2011 [http://www.sevensols.com/index.php?seccion=262&subseccion=270].
21. Ridgeway D: **Designing Complex 2-Dimensional Convolution Filters.** The programmable Logic DataBook, Xilinx; 1994.
22. 2011, The HDR Photografic Survey, http://www.cis.rit.edu/fairchild/HDR.html.
23. Mantiuk R, Tomaszewska A, Heidrich W: **Color correction for tone mapping.** *Computer Graphics Forum* 2009, **28(2)**:193-202.
24. Reinhard E, Devlin K: **Dynamic range reduction inspired by photoreceptor physiology.** *IEEE Trans Visual Comput Graph* 2005, **11(1)**:13-24.
25. Yoshida A, Blanz V, Myszkowski K, Seidel HP: **Perceptual evaluation of tone mapping operators with real-world scenes.** *Paper presented at IS&T/SPIE's 17th Annual Symposium Electronic Imaging* San Jose, CA, USA; 2005, 192-203.

doi:10.1186/1687-5281-2012-1
Cite this article as: Ureña *et al.*: Real-time tone mapping on GPU and FPGA. *EURASIP Journal on Image and Video Processing* 2012 **2012**:1.