GPU based Implementation of Film Flicker Reduction Algorithms

Martn Piñeyro and Julieta Keldjian and Alvaro Pardo

mpineyro@ucu.edu.uy, apardo@ucu.edu.uy, jukeldji@ucu.edu.uy
Department of Electrical Engineering, School of Engineering and Technologies, Universidad Catolica del Uruguay

Abstract. In this work we propose an algorithm for film restoration aimed at reducing the flicker effect while preserving the original overall illumination of the film. We also present a comparative study of the performance of this algorithm implemented following a sequential approach on a CPU and following a parallel approach on a GPU using OpenCL.

1 Introduction

Visual flicker is one of the most common consequences of degradation in old films. It is the result of global intensity fluctuations between consecutive frames. In some cases flicker can be a local effect too, with regions of the frame experiencing local intensity variations between consecutive frames. Although it may seem a simple problem that could be addressed with traditional intensity normalization techniques, usually this approach is not able to remove the distortion completely.

When film is digitalized by capturing its projection with a digital camera flicker is introduced as a consequence of temporal mismatch between the capture and projection system (the acquisition of frames is unsynchronized with the projectors shutter). Aging is also an important cause of flicker. Not all the frames of a film suffer the same degradation along the time and therefore unintended variations in mean illumination may be encountered. When removing flicker through digital image processing the effects due to digitalization and aging have to be removed while preserving the flicker caused by the mechanical limitations of the equipment originally used to capture the content.

In order to avoid the introduction of new structures as a consequence of the restoration process usually the flicker reduction algorithms use histogram corrections that preserve the geometry of the frames. Also, since flicker is a temporal distortion affecting a sequence of frames, restoration processes aimed at reducing it need to consider multiple frames in order to capture the temporal intensity variations.

2 Previous work

The works in the literature of flicker reduction can be categorized in two types: the ones that apply local methods and the ones that apply global methods. In this work we concentrate in algorithms that apply global methods to modify the histogram of each frame of the film in order to smooth the intensity variations across time. The first approach for global intensity modification is based in the use of affine transformations. The corrected frame at time t, $\hat{I}(x;t)$, is transformed with the equation $\hat{I}(x;t) = a(t)I(x;t) + b(t)$ where parameters $a(t)$ and $b(t)$ are selected to match the input dynamic range to a desired one. Since flicker varies with time both parameters are time dependent. The values of $a(t)$ and $b(t)$ are selected to reduce the mean intensity variations within the dynamic range. The main difficulty with this solution is that the desired output dynamic range has to be manually given to the algorithm.

Another option for applying global changes to the frames is using histogram modifications (not only affine as in the first case). This allows more general intensity modifications between frames. In [5] the authors proposed matching the histogram of a given frame $I(x;t)$ to the mean histogram in a window centered at time t. In [3, 2] Delon analyzed the algorithm from [5] and proposed an improvement. The main observation made by Delon is the following: if the histograms of two frames that differ only in a constant are averaged, two unimodal histograms may produce a bimodal one. This simple observation shows the limitations of the algorithm proposed in [5]. To deal with this problem Delon proposed to average the inverse of the cumulative histograms.

For trasforming two images I_1 and I_2 to have the same histogram both images are transformed with continuous and strictly increasing functions $g_i : [0, 255] \to [0, N]$. Assuming continuous images the cumulative histograms are defined as: $H_i(q) = \int_0^q h_i(\lambda)d\lambda$, where $h_i(q)$ is the histogram of the image I_i. In [2] Delon shows that the following transformations produce images with the same cumulative histogram: $g_1 = \frac{H_1^{-1}+H_1^{-1}}{2} \circ H_2$, $g_2 = \frac{H_1^{-1}+H_1^{-1}}{2} \circ H_1$ In this way the final cumulative histogram of each image is: $H_i \circ g_i^{-1} = \left(\frac{H_1^{-1}+H_1^{-1}}{2}\right)^{-1}$. This method was also presented in the discrete case by Cox in [1].

In this work we use this idea but using a weighted average of frames within a temporal window following the ideas discussed in [3]. Given a set of N frames in a given window the inverse average is defined as:

$$H_a = \left(\sum_{i=1}^{N} w_i H_i^{-1}\right)^{-1}.$$

To remove the flicker each frame is transformed with a function $g_i = H_a \circ H_i$.

Using a weighted inverse average inside a window permits to smoothy correct the flicker while allowing variations (in the temporal axis). In that way global brightness variations that could be part of the original material and not the result of any degradation process are preserved. Using the average of all the frames of the scene is not recommended because it will destroy the original content of the scene forcing all frames to have equal histograms. With the proposed approach the temporal variations of the intensity are smoothed and discontinuities avoided while respecting the original variations of the film.

3 GPUs and OpenCL

Modern GPUs are very efficient in parallel processing of computer graphics data but also with any other type of data that can take advantage of the parallel nature of the GPUs. At the same time they are very competitive in terms of price. NVIDIA was the first company to introduce a general-purpose programming model with the release of CUDA and recently other companies joined efforts around the OpenCL standard. Although CUDA is widely extended OpenCL has the attractiveness of being a standard supported by many companies. In fact NVIDIA also supports OpenCL just like ATI which is another big player in the field.

One of the goals of this work was to explore the use of OpenCL for image processing. The research group has been working with CUDA so we took this project as a test case to evaluate the suitability of OpenCL for this kind of problem. The development was done using Windows 7, Visual Studio and OpenCL and tested in a ATI Radeon 5650 GPU.

4 Algorithm Implementation

Histogram Computation: The first stage of the algorithm computes the histogram of the N frames within the temporal window that will be considered in the flicker reduction process. For comparison purposes we implemented a CPU routine using C++ and another one based on the GPU using OpenCL. Since sequential computation of histograms presents no difficulties we will focus on the GPU-based routine.

An OpenCL application consists of two main parts: the host program and the kernels. Initially the host program defines a context (in this work the context consists of a CPU and a GPU). Then it defines a command queue in which commands issued by the CPU are scheduled for execution on the GPU. Subsequently, the host program loads a kernel file and creates a kernel object from the code in that file (in this case the set of instructions that calculate the histogram of an image). Finally, it transfers the arguments to the GPU (input image and empty array in which to return the calculated histogram) and enqueues the execution of the kernel. Once the execution finishes the host program reads the results back into a result buffer.

Each kernel execution processes in parallel every pixel within a subregion of the input image called work-group which dimensions are defined by the user before queuing the execution. The kernel gets the gray level of every pixel within a work-group and increases the value of the corresponding bins of the histogram. Notice that with a parallel approach as the one being described when two pixels of the same work-group have the same gray level, two threads will try to write to the same memory location simultaneously (the same bin of the histogram). To preserve data integrity in such cases it is necessary to use atomic operations, which implies a decrease in performance. See the following code:

```
if(x < image_width && y < image_height){
  color = read_imagef(inputImage, sampler, coordinates);
  atomic_inc(&histogram[color]);
}
```

Cumulative Histogram: Computing the cumulative histogram of an image consists in adding the values of all the bins of its histogram, hence it is inherently a sequential operation which isn't likely to be parallelized. For this reason this stage of the algorithm was implemented on the CPU. This routine takes as argument a structure containing the histograms of the N frames of the window being considered for restoration and returns another structure containing the N corresponding cumulative histograms.

Weighted average of cumulative histograms: We define a symmetric time window of N frames centered at the frame that is being processed (input frame) and compute the weighted average of the cumulative histograms within this window. The weight assigned to each frame of the window is determined through a normal distribution centered in the input frame. This way the coefficient corresponding to each cumulative histogram considered in the weighted average is calculated in terms of its proximity in time respect to the input frame using weights: $w_i = \exp(-\frac{(i-N/2)^2}{2\sigma^2})$

The standard deviation must be defined in terms of the number of frames that comprise time window. During the tests performed in this study it was found that using $\sigma^2 = 2N$ allows flicker reduction while still respecting the intensity variations that are part of the film's content.

When the first frame of the sequence is being processed the system does not have information from previous frames, therefore the initial weighted average considers only subsequent frames. Once the first frame has been restored previous frames become available and the routine progressively incorporates their cumulative histograms to the computation of the weighted average. The opposite case is presented while reaching the last frame of the sequence.

Midway equalization: Once the average cumulative histogram is computed the algorithm proceeds to equalize the histogram of the input frame. For this we implemented the method proposed in [4]. The routine defines a transformation that assigns to each pixel of the input image the gray level of the element of the average cumulative histogram which has the same rank as the considered pixel.

First the routine determines the rank of each gray level in the input image using its cumulative histogram (H_{input}): $H_{input}(\lambda_i) = k_i(rank)$

Then it calculates the gray level that corresponds to that rank through the inverse of the weighted average cumulative histogram: $H_a^{-1}(k_i) = \lambda_i'$

```
for(i=0; i<256; i++){
  //find the rank of each gray level of the input image
  rank = input_cummulative_histogram[i];
  //find the gray level that has the same rank
  r = 0;
  while(input_cummulative_histogram [r] < rank){
   r++;
  }
  //write the corespondent gray level to the transformation array
  transform[i] = r;
}
```

Image Transformation: The routine that applies the transformation to the input image was implemented on an OpenCL kernel. The routine takes as arguments the input image, an array describing the transformation and a blank image to which the result will be written to. Each execution of the kernel reads in parallel the gray level of a section of 16 x 16 pixels (one work-group) and writes the new gray level to the output image (according to the transformation described by the array).

```
if(x < image_width && y < image_height){
  input_color = read_imagef(input_image, sampler, coordinates);
  output_color = transform[input_color];
}
write_imageuf(output_image, coordinates, output_color);
```

5 Results and Discussion

In order to quantify the effectiveness of the flicker reduction algorithm implemented in this work several films with different degrees of flicker were processed and their mean gray level was compared before and after processing. The performance of CPU based and GPU based implementations of the algorithm was also measured during these tests.

Performace results: Three films with different frame sizes were processed in order to compare the time it takes to compute the histograms and to apply the transformation using a sequential routine implemented in the CPU and a routine implemented in the GPU.

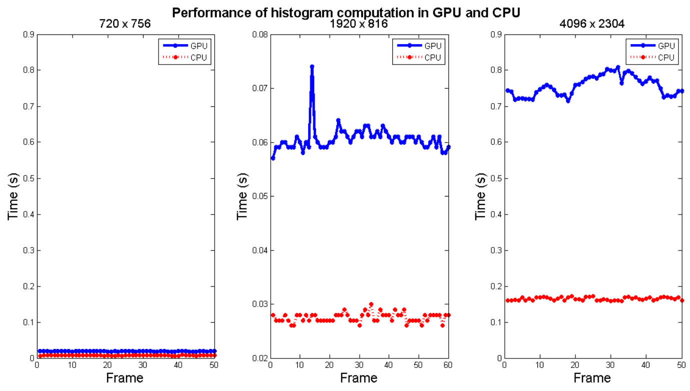

Fig. 1: Histogram computation performance.

The results presented in Figure 1 shows that in all three cases the computation of histograms runs faster on the CPU than on the GPU. Two possible reasons for the GPUs poor performance on the computation of image histograms are the following:

- In order to compute the histogram of an image on the GPU the input image and a buffer (where to write the histogram to) must be transferred from the CPU to the GPU. After the computation the result must be read from the GPU and this involves transferring the buffer back to the CPU.
- To preserve the integrity of data during simultaneous attempts to writing to the same memory location atomic operations must be used (preventing the execution of parallel instructions).

Figure 2 presents a comparison between the time it takes to apply the transformation to each frame of the sequence using a routine running on the GPU and a routine running on the CPU. These results show that for an image of 720×756 pixels both routines achieve similar performances. When processing larger frames (1920×816 and 4096×2304) the routine runs faster on the GPU than on the CPU.

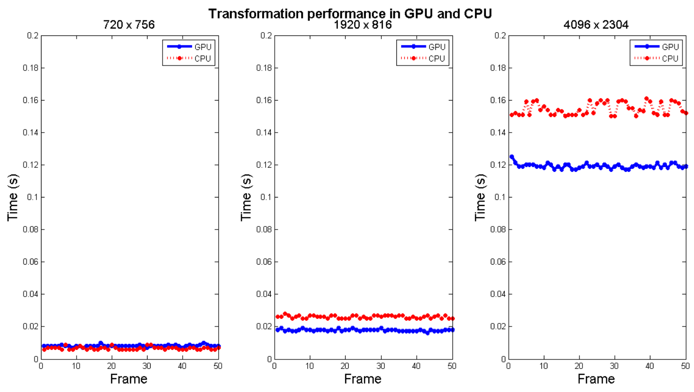

Fig. 2: Transformation computation performance.

As in the computation of histograms before applying the transformation in the GPU it is necessary to transfer the input data from the CPU to the CPU. In this case the parameters that must be transferred are the input image, an array in which the transformation is defined and a blank image on which the result will be written to. For small frames (less than 1 MP) the time it takes to the CPU and the GPU to apply the transformation is similar, thus the additional delay of the data transfer is not justified. For frames larger than 1 MP pixel the transformation runs faster enough on the GPU over the CPU to justify the computational cost of the data transfer and therefore the total processing time of the GPU is less than the total processing time of the CPU.

Flicker reduction: This section presents the results of three tests aimed at measuring the ability of the algorithm for the reduction of film flicker. The results arise from the comparison of the mean gray level of each frame of a flickering film before and after being processed. Figure 3 shows in blue the mean gray level of the first 100 frames of the films. In all three films the mean gray level of the original frames experience significant variations between consecutive frames and several discontinuities along the temporal axis indicating that the films have flicker.

In order to preserve the original content of the scene, it is desirable that the flicker reduction algorithm removes the discontinuities that cause the flicker effect while preserving the intensity variations that are part of the content of the film and not part of any degradation process.

Figure 3 shows that using an average of the cumulative histograms results in an almost constant mean gray level along the stream. This indicates that while this approach will reduce the flicker it will not preserve the original intensity variations of the original content of the film. On the other hand, the mean gray level values that result from using a weighted average of the cumulative histograms within a temporal window show that although rapid transitions and discontinuities were removed, slow transitions remain. This suggests that with this approach the global brightness variations of the original content are preserved.

6 Conclusions

This work proposes a way of preserving the original global intensity variations of a film when applying a flicker reduction algorithm based on the computation of the inverse weighted average of the cumulative histograms within a time window. The performance results obtained in this study lead to the conclusion that using OpenCL for computing image histograms on a GPU fails to achieve better performance than the computation of image histograms on a CPU. Regarding the transformation process that reduces the flicker it was observed that the extra time required to transfer data to and from GPU is not justified unless the frames being processed are larger than 1 MP. In such cases running histogram calculation on the CPU and applying the transformation on the GPU can achieve a better performance of the flicker reduction algorithm than if it runs entirely on the CPU.

References

1. I. Cox, S. Roy, and Sunita Hingorani. Dynamic histogram warping of image pairs for constant image brightness. In *ICIP*, pages 366–369, 1995.
2. J. Delon. Midway image equalization. *Journal of Mathematical Imaging and Vision*, 21(2):119–134, 2004.
3. J. Delon. Movie and video scale-time equalization ; application to flicker reduction. *IEEE Trans. Image Proc.*, 15(1):241–248, 2006.
4. J. Delon and A. Desolneux. Stabilization of flicker-like effects in image sequences through local contrast correction. *SIAM Journal on Img. Science*, 3(4):703–734.
5. V. Naranjo and A. Albiol. Flicker reduction in old films. In *ICIP*, pages 657–659, 2000.

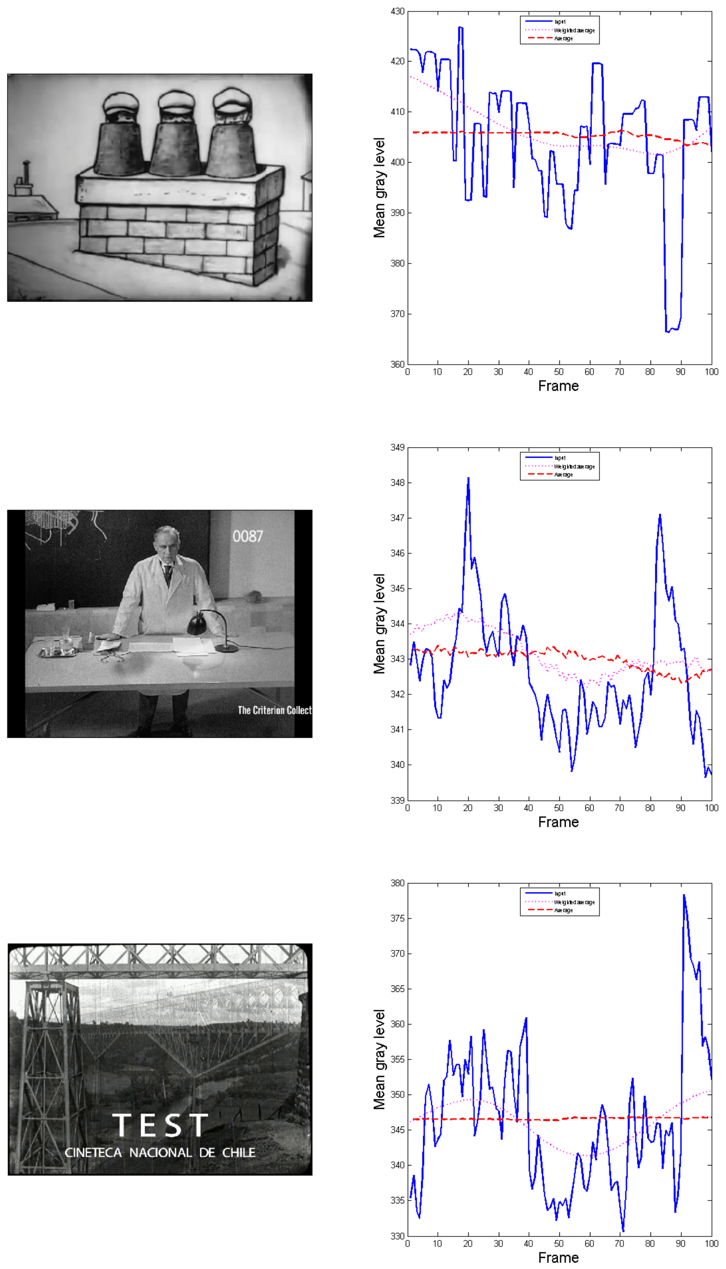

Fig. 3: Film flicker reduction. Left: Sample Frame. Right: original mean luminance (blue), restored mean luminance using average (red) and restored mean luminance usong wiegthed average (magenta).