Interactive Local Tone Mapping Operator with the Support of Graphics Hardware

B. Roch*
Institute of Computer Graphics and Algorithms, Vienna University of Technology (Austria)
A. Artusi†
Department of Computer Science, Intercollege Nicosia (Cyprus)
D. Michael‡
Department of Computer Science, University of Cyprus
Y. Crisanthou§
Department of Computer Science, University of Cyprus
A. Chalmers¶
Warwick Digital Laboratory, University of Warwick, UK

Abstract

The flexibility of current graphics hardware is still not enough to ensure the full implementation of an original complex algorithm such as a local tone mapping operator, which maintains the same quality performances as its original CPU implementation. Significant changes are often needed to the original CPU implementation in order to overcome many of the limitations of the current graphics hardware. As a result of this we often have reduced quality reproduction, and the frame rate of the GPU implementation is not always acceptable for real-time applications. In this paper, we show how to change the CPU implementation of a state of the art local tone mapping operator for accelerating the computation process to real time frame rates. We also present a modification of the luminance local adaptation computation, showing a simple but not yet exploited property of the Gaussian filter, allowing us to maintain the same quality appearance of the original tone mapping operator. Finally we test the hardware implementation on NVIDIA graphics cards on several images and as well as a video. We compare our hardware implementation with the corresponding CPU implementation and previous work.

1 Introduction

The conversion from High Dynamic Range to traditional display luminance is known as tone-mapping (TM). TM is a very important last step in the (re-)production of realistic images and many operators have been proposed. However, the computational requirements of a complex tone mapping operator (TMO) is still such that it is not possible to achieve high quality results in real-time. Existing TMOs can be subdivided in two basic categories: global and local operators. Global TMOs apply the same operation to all pixels of the input image, while local operators take into consideration the local properties of individual pixels and use this information to preserve the local contrast reproduction. The graphics hardware, currently available, is becoming more and more flexible and suitable for general purpose programming, but there are still several limitations that restricts the possibility of implementing complex algorithms such as local TMOs. In fact a difficult aspect of GPU programming, as discussed in Goodnight et al. [Goodnight et al. 2003], is that it requires exceedingly careful optimisation in order to achieve the performance that is expected. Several factors contribute to this problem, such as: memory bandwidth, driver overhead, etc. Some of these problems can be reduced, but not completely avoided [Goodnight et al. 2003].

Recently, several high-level programming languages for GPUs were introduced, which help the programmer to speed up the programming phase, but on the other hand the limited number of assembly instructions (1024), still reduces the possibility to implement a sophisticated algorithm without significantly modifying it.

In this paper, we propose a hardware implementation of a state-of-the-art local TMO, showing how it is possible to overcome the limitations and drawbacks that still affect the direct implementation of a state of art TMO directly on the GPU. This implementation is able to deliver in real-time the results of the original TMO (CPU implementation), maintaining intact its quality reproduction. No trade-off between quality and speed is required. Additionally, a modification of the local luminance adaptation computation of the original TMO is presented.

The paper is organised as follow. Section 2 describes related work. Section 3 provides an overview of the hardware implementation. Section 4 shows the experimental results. Finally Section 5 concludes and suggests possible future work.

2 Related Work

The concept of TM was introduced by Tumblin and Rushmeier [Tumblin and Rushmeier 1993], in which they proposed a tone reproduction operator that preserves the apparent brightness of scene features. Subsequenly many TMOs have been proposed that can be classified as either global or local as discussed in the Section 1. All these TM methods concern accurate operators that attempt to reproduce individual visual effects at non-interactive rates.

Is not the purpose of this paper to give a complete overview of the state-of-art of the TMOs proposed in the literature. For a full overview of tone mapping see[Devlin et al. 2002]. In this Section we will concentrate on reviewing the work that attempts to develop a real time TMO.

The interactive solutions to the TM problem can be classified in two main categories: direct GPU implementation of the original TMO, and definition of a general acceleration platform. The first one refers to the implementation of the original CPU implementation of the TMOs directly on the GPU. This often requires a significant modification of the original CPU implementation. As a result of this we have reduced quality when compared with the output obtained with the original TMO. In addtion, as a final drawback, the time performances are rapidly decreasing as the resolution of the

*e-mail:phibre@gmx.net
†e-mail:artusi.a@intercollege.ac.cy
‡e-mail:despinam@cs.ucy.ac.cy
§e-mail:yiorgos@cs.ucy.ac.cy
¶e-mail:alan@compsci.bristol.ac.uk

input frame increases [Goodnight et al. 2003]. The second category, see Artusi et al. [Artusi et al. 2003], aims to develop a framework that can be applied to the current state of art TMOs in order to achieve interactive rates. The main advantage of this idea is that no modifications are required to the original TMOs, that are still implemented on the CPU.

Several authors, including Durand and Dorsey [Durand and Dorsey 2000][Durand and Dorsey 2002] and Ward et al. [Ward Larson et al. 1997], have proposed some acceleration methods in order to improve the computational performance of their TMOs. Several global TMOs which currently do achieve interactive rates, tightly coupled with current graphics hardware, have been proposed by Cohen et al. [Cohen et al. 2001] and Scheel et al. [Scheel et al. 2000]. Goodnight et al.[Goodnight et al. 2003] discussed the troublesome aspects of the GPU programming and presented a hardware implementation of the Reinhard et al. operator [Reinhard et al. 2002]. They also proposed a different algorithm for the photographic zone computation, in order to overcome the limitations of current graphics hardware. Krawczyk et al. [Krawczyk et al. 2005] presented the reproduction of perceptual effects within real-time tone mapping. Artusi et al. [Artusi et al. 2003] proposed a general framework usable only for global operators. They analysed the acceleration problem and discussed the hardware implementation of this framework, reducing the implementation complexity without modifying the rendering pipeline.

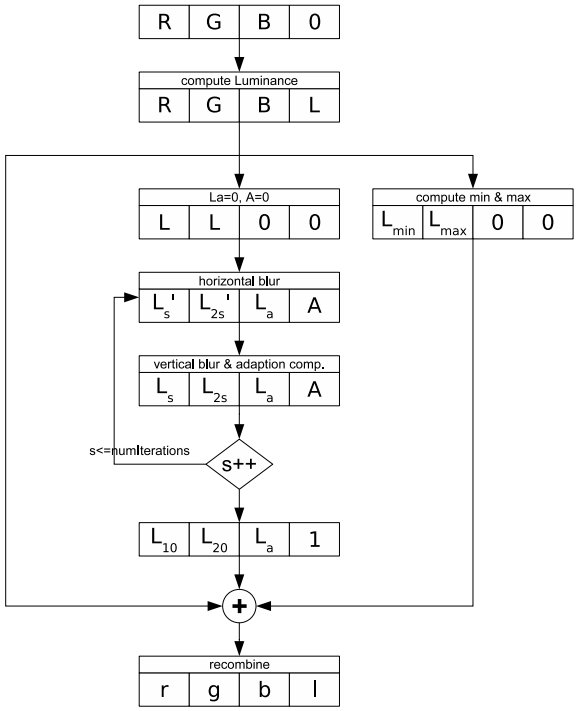

Figure 1: Per-pixel view of our implementation.

3 Implementation

Throughout this paper the uppercase notation of *RGB* (red,green, and blue triplet) and *LUM* (luminance) will represent high dynamic range quantities, while lower case notation means low dynamic range quantities. We have chosen the TMO proposed by [Ashikhmin 2002] for two reasons. First, it achieves visual appealing results. Second, it presents similar algorithmic structure to the previous work concerning HW implementation of TMOs. This will allow us to compare the ability of our implementation to achieve a better performance in terms of speed and quality, with previous work in a straightforward manner.Our implementation is depicted in Fig. 1.

The process roughly comprises the following stages:

- Convert *RGB* to *LUM*
- Compute L_a image with the neighborhood growing procedure
- Scale *LUM* with L_a, recombine with *RGB* and apply gamma correction to compute final *rgb* result

We use OpenGL and the OpenGL Shading Language to implement the previously mentioned stages. Framebuffer objects (FBOs) are used to provide a fast way for floating point textures, as well as floating point render-targets. Initialization of the structures is done once at application startup. Every stage uses one or more floating point textures as input and outputs the result to the double-buffered FBO resultbuffer. All computations are implemented as fragment shaders.

3.1 Computation of luminance, minimum and maximum

The first step consists of converting the *RGB* input data to *LUM*. The resulting luminance value is stored together with the *RGB* triplet as (R,G,B,LUM) in the resultbuffer. Afterwards, an image pyramid is used to compute minimum and maximum values of *LUM*. We create the pyramid by using the original input image as the base (level=0). For each level we bisect the before headed level until it reaches a size of 1x1 pixels. That way 4 pixels of level k are projected onto 1 pixel of level $k+1$, storing the min and max in the red and green channel. We end up with one pixel, holding the minimum and maximum value of the *LUM* input image, see Fig.1.

To smooth the high changes in brightness between the frames, we do a simple interpolation using eq. 1. The new_{MinMax} holds the minimum and maximum values of the current frame, while ps_{MinMax} represents the interpolated values from the last frame. Linear scaling is applied to compute the new min and max luminance values (int_{MinMax}) of the current frame.

$$int_{MinMax} = pf_{MinMax} + (new_{MinMax} - pf_{MinMax})/scale \qquad (1)$$

The scale value has been chosen experimentally as 8.0.

3.2 Neighborhood growing procedure

In the original CPU implementation [Ashikhmin 2002], a Gaussian pyramid was chosen to compute the levels of Gaussian blur. We have chosen a slightly different approach, based upon applying the same gaussian filter kernels recursively. We utilize 2 one dimensional filter kernels: (1,2,1) for s and (1,4,6,4,1) for $2s$. The resultbuffer, which stores the interim results at each iteration is organised in the following way: to keep up with the computation and store all important results, 4 channels are needed per pixel. The configuration of the quadruple is as follows: (L_s,L_{2s},L_a,α). L_s and L_{2s} represent the *LUM* value, filtered with radius s and $2s$. The L_a is the current adaptation value, and the α builds a switch: When l_c

exceeds threshold, α is set to 1.0 and L_a stays fixed for the pixel, for the rest of the procedure.

In the first pass of each iteration we blur the result image of the last iteration horizontally - this interim results are called L_s' and L_{2s}'. After swapping the resultbuffer it is bound as a texture and the process of 1D-blurring is repeated again (now the vertical version of the filter kernels). The results are stored in L_s and L_{2s}. These values are used in the same pass to compute the l_c of the current iteration. After selecting the right L_a based upon the l_c, we obtain the pixel layout already described. We repeat the neighborhood growing procedure until s is equal to *numIterations*.

3.2.1 Recursive computation of the Gaussian pyramid

In this subsection we describe in more detail the recursive Gaussian filter technique used for the neighborhood growing procedure (Sec. 3.2). We will exploit one simple but useful property of the Gaussian filter, which help to accelerate the entire process: If we use the elements of Pascal's triangle as the filter kernel, the result of a Gaussian filter of general size $n \times s$ is equivalent to the result of a Gaussian filter of size s applied recursively n times.

Gaussian filters themselves are separable, i.e. that it is enough to just prove this for the one dimensional form of the filter.

i	Coefficients
0	1
1	1 1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
...	...

Every line with the index $i = 2 * s$ can be taken as a Gaussian 1D filter, with size s ((1), (1,2,1), ...).

s	$i = 2 * s$	Coefficients
0	0	1
1	2	1 2 1
2	4	1 4 6 4 1
3	6	1 6 15 20 15 6 1
...	...	...

We now want to prove that s-times execution of the recursive method with the base element (1,2,1) on a line of pixels is equal to a filter having index $2 * s$. First we compute the result by processing the pixel line twice with the base element (1,2,1)

$$\begin{aligned} A' &= ?+2A+B \\ B' &= A+2B+C \\ C' &= B+2C+D \\ &\dots \\ H' &= G+2H+I \\ I' &= H+2I+? \end{aligned}$$

which yields the following scanline: $A'B'C'D'E'F'G'H'I'$, while the second iteration looks like this:

$$\begin{aligned} A'' &= ?+2A'+B' \\ B'' &= A'+2B'+C' \\ C'' &= B'+2C'+D' \\ &\dots \\ H'' &= G'+2H'+I' \\ I'' &= H'+2I'+? \end{aligned}$$

which results in: $A''B''C''D''E''F''G''H''I''$. The ? symbolises the question of which value has to be taken, if the filter runs over the border of the image. We will discuss this question later, and concentrate just on the "complete" terms. Now we compute pixel values for a sample line of pixels: *ABCDEFGHI* with filter of $s = 2$ (i.e. $(1,4,6,4,1)$), resulting in

$$\begin{aligned} \tilde{A} &= ?+6A+4B+C \\ \tilde{B} &= ?+4A+6B+4C+D \\ \tilde{C} &= A+4B+6C+4D+E \\ \tilde{D} &= B+4C+6D+4E+F \\ &\dots \\ \tilde{I} &= G+4H+6I+? \end{aligned}$$

Which results in the pixel line $\tilde{A}\tilde{B}\tilde{C}\tilde{D}\tilde{E}\tilde{F}\tilde{G}\tilde{H}\tilde{I}$. We know that, starting with C'' (to avoid the question of border pixels) we have: $C'' = B'+2C'+D' = (A+2B+C)+2*(B+2C+D)+(C+2D+E) = A+4B+6C+4D+E = \tilde{C}$.

Because $s = 1$ provides the same result for both (we use (1,2,1) in both cases), and is also valid for s=2, we need to prove the way from $s \rightarrow s+1$ holds for all s. Because of the binomial coefficient, where we can construct the coefficients for every $(x+y)^n$ by looking at Pascal's triangle at index n, we know that every $2 * nth$ element is constructible by taking the n-th power of the base element $(x+y)^2$:

n	$(x+y)^{2n}$	Coefficients
0	1	1
1	$(x+y)^2$	1 2 1
2	$(x+y)^4$	1 4 6 4 1
...	...	...

The iterated application of the base element (1,2,1) means nothing else than the above, which proves the concept.
Though both methods are equal in a mathematical way, numerical errors can lead to very small differences. These differences are no problem for us, because we don't need the exact result of the equation, but only a criteria to decide if a value is much bigger than its neighborhood or nearly equal.

3.3 Recombination

We then tone map L_a with Eq.2 which uses the world capacity function, defined in eq.3. With the eq. 4 we 'infuse back' the fine details in the final tone mapped image as described in the original CPU implementation [Ashikhmin 2002]. This value is then used to scale each component of the *RGB* triplet by L_a/L_d, followed by a gamma correction, which is implemented as texture lookup. The whole computation needs one filter pyramid and one additional framebuffer object for the resultbuffer. All computations are executed on a per-pixel basis. The computation of the L_{minmax} needs $log(max(width, height) - 1))/log(2) + 1$ passes. The conversion from *RGB* to *LUM*, the computation of L_a, and the recombination

HDR Image	Size (pixels)
aeroporto.hdr	1024x705
lamp.hdr	400x300
memorial.hdr	512x768
nave.hdr	720x480
rosette.hdr	720x480
stilllife.hdr	1240x846

Table 1: Dimensions of the HDR images used for testing.

HDR Image	RMS % error	mean % error
aeroporto.hdr	0.053 %	0.008 %
lamp.hdr	1.063 %	0.023 %
memorial.hdr	0.041 %	0.025 %
nave.hdr	0.201 %	0.091 %
rosette.hdr	0.043 %	0.035 %
stilllife.hdr	0.074 %	0.051 %

Table 2: RMS and mean percent errors of our GPU implementation, of the local Ashikhmin operator, computed as described in the Eq. 5 and Eq. 6 respectively. These values are computed considering the CPU implementation of the operator as the accepted values.

together need $2+2*(numIterations)$ rendering passes. The LDMAX is a predefined constant, describing the maximum contrast of a display. It is therefore set to LDMAX=100.

$$TM(L_a) = LDMAX(C(L_a) - C(L_{min}))/((C(L_{max}) - C(L_{min})) \quad (2)$$

$$C(L) = \begin{cases} L/0.0014 & if L < 0.0034 \\ 2.4483 + \log(L/0.0034)/0.04027 & if 0.0034 \leq L < 1 \\ 16.5630 + (L-1)/0.4027 & if 1 \leq L < 7.2444 \\ 32.0693 + \log(L/7.2444)/0.0556 & otherwise \end{cases} \quad (3)$$

$$L_d(x,y) = (L(x,y)TM(L_a(x,y))/La(x,y) \quad (4)$$

4 Experimental Results

In this Section we describe the experimental results done for testing our GPU implementation of the local operator. In table 1 the images we used for testing are listed.

We demonstrate our GPU implementation in a real-time setting and on still images; integrating it in a rendering system that receives HDR frames as input.

The experiments were conducted on a PC with graphics card nVidia Go6800.

We will compare our GPU implementation with previous works such as Goodnight et al.[Goodnight et al. 2003] that implemented the Reinhard et al. [Reinhard et al. 2002] operator that uses a similar technique, for computing the local adaptation luminance as the Ashikhmin model.

We first tested the time performances of the GPU implementation in a real-time setting. In table 3 we show the results of this test (in fps). The approach of Goodnight et al. [Goodnight et al. 2003] used a more complex iterative system. In fact they used n/2 + 2 render passes for a $n \times n$ filter kernel. This difference in time performances decreases drastically when the resolution of the input frame and the number of zones are increasing; dropping down to non interactive performances. This is confirmed by a simple comparison of the fps achieved by our GPU implementation with the fps achieved by Goodnight et al.[Goodnight et al. 2003]. We can observe that real-time performances are achieved even with the frame resolution of 512×512 pixels, when in the case of Goodnight et al. [Goodnight et al. 2003] at this resolution the computation performances were already dropping down. In their case, in order to keep the computation performance at an acceptable value of fps, they needed to compromise between quality and speed. In our case this is not necessary as shown in table 3.

We can also observe how the overhead added by our GPU implementation to the rendering process is decreasing with the increase of the frame resolution. For high frame resolution the overhead is becoming imperceptible in term of fps. This suggests that for high frame resolutions no more work can be done on the part of the GPU of the state of art of the TMO, for improving the time performances of the real-time application.

The results of Table 3 are reported also in Fig. 2.

In the video submitted with this paper we smooth the high changes in the dynamic range of the sequence frames. It has been implemented as explained in section 3.1. Recall that the final comparison of the still images between GPU and CPU implementation of the TMO, for a correct comparison, is done without the simulation of the time dependency adaptation process.

Krawczyk et al. [Krawczyk et al. 2005] also claim in their work real-time performances, but in reality these are obtained only for low image resolution (320×240 pixels) and when the levels of the Gaussian pyramid and image resolution are increasing the frame rates is strongly decreasing to non interactive rates. Furthermore the time performances were obtained with approximated results, in term of quality, of the TMO.

In order to validate the hardware implementation, we conducted a quality comparison with the corresponding images obtained with the CPU implementation of the original algorithm. In fig. 3, we show the results obtained with the CPU implementation (Right), and the results obtained with the GPU implementation (Left).

In order to validate the quality comparison we computed the Root Mean Square (RMS) percent error between the CPU and the GPU implementations as:

$$error_{RMS\%} = \sqrt{\frac{1}{n}\Sigma[\frac{p_{cpu}(x,y) - p_{gpu}(x,y)}{p_{cpu}(x,y)}]^2} \quad (5)$$

where n is the number of pixel in the image, and $p(x,y)$ is the Luminance pixel value in the image. We also evaluate the mean percent error as:

$$error_{mean\%} = \frac{1}{n}\Sigma \mid \frac{p_{cpu}(x,y) - p_{gpu}(x,y)}{p_{cpu}(x,y)} \mid \quad (6)$$

Table 2 gives the error calculation for the images used in our tests. The values of the errors for the all images used in the tests are comparable if not better, in term of RMS and mean percent error, than obtained in Goodnight et al. [Goodnight et al. 2003].

5 Conclusion and Future Work

We have presented a hardware acceleration of a state of the art local TMO [Ashikhmin 2002]. We have shown a unexploited but use-

HDR Frame	Rendering + GPU TM [fps]	Rendering [fps]
256 x 256	39	60
512 x 512	28	29
1024 x 1024	8	9
2048 x 2048	7	9

Table 3: Results (in fps) of the GPU implementation varying the resolution of the HDR frame. In this table we compare the fps for the Rendering + GPU TM with the fps for the only Rendering without applying the GPU TM operator.

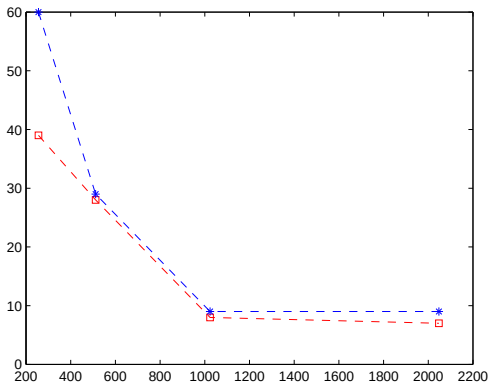

Figure 2: Graphical representation of Table 3.

ful property of the Gaussian filter and how to use it for modifying the Gaussian blurring technique used for computing the local luminance adaptation Despite the level of acceleration, the quality of the output image is maintained when compared with the output image obtained with the corresponding original CPU implementation. We also showed that the overhead introduced by our GPU implementation of the TMO to the rendering process is imperceptible, for high resolution frames, in term of fps.

These results shown how the main limitations of the current graphics hardware, as image quality reduced and speed strongly related to the resolution frame, are in part overcome.

We compare our results with two previous work [Goodnight et al. 2003] [Krawczyk et al. 2005], and showed ours has superior quality and speed.

Future acceleration of local tone mapping operators must be based on a different development concept that is based on an anlysis of the hardware limitations, reducing the necessary compromise for adapting the original software implementation on the latest hardware.

Acknowledgements

This research was partially supported by the Cyprus Research Promotion Foundation (RPF) within the scope of the IPE project - PLHRO/1104/21(ToneMap), "High Quality Rendition of Images for a Soldier Training VR System", and by the ERCIM "Alain Bensoussan" Fellowship Programme.

References

ARTUSI, A., BITTNER, J., WIMMER, M., AND WILKIE, A. 2003. Delivering interactivity to complex tone mapping operators. 38–44. Eurographics Symposyum on Rendering 2003.

ASHIKHMIN, M. 2002. A tone mapping algorithm for high contrast images. In *Rendering Techniques '02 (Proceedings of the 13th Eurographics Workshop on Rendering)*, 145–156.

COHEN, J., TCHOU, C., HAWKINS, T., AND DEBEVEC, P. 2001. Real-Time high dynamic range texture mapping. In *Rendering Techniques '01 (Proceedings of the 12th Eurographics Workshop on Rendering)*, 313–320.

DEVLIN, K., CHALMERS, A., WILKIE, A., AND PURGATHOFER, W. 2002. Star: Tone reproduction and physically based spectral rendering. 101–123. Eurographics 2002.

DURAND, F., AND DORSEY, J. 2000. Interactive tone mapping. In *Rendering Techniques '00 (Proceedings of the 11th Eurographics Workshop on Rendering)*, 219–230.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high dynamic range image. In *Proceedings of SIGGRAPH 2002*, 257–265.

GOODNIGHT, N., WANG, R., WOOLLEY, C., AND HUMPHREYS, G. 2003. Interactive time-dependent tone mapping using programmable graphics hardware. In *EGSR03, Eurographics Symposium on rendering 2003*, 26–37.

KRAWCZYK, G., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2005. Perceptual effects in real-time tone mapping. In *Spring Conference in Computer Graphics*, 195–202.

REINHARD, E., STARK, M., SHIRLEY, P., AND FERWERDA, J. 2002. Photographic tone reproduction for digital images. In *Proceedings of SIGGRAPH 2002*, 267–276.

SCHEEL, A., STAMMINGER, M., AND SEIDEL, H.-P. 2000. Tone reproduction for interactive walkthroughs. In *Computer Graphics Forum (Proceedings of Eurographics 2000)*, vol. 19(3), 301–312.

TUMBLIN, J., AND RUSHMEIER, H. E. 1993. Tone reproduction for realistic images. *IEEE Computer Graphics and Applications 13*, 6 (Nov.), 42–48.

WARD LARSON, G., RUSHMEIER, H., AND PIATKO, C. 1997. A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (Oct.), 291–306.

Figure 3: (Left) Images obtained with the GPU implementation, (Right) the original CPU implementation.