Real-Time High Fidelity Inverse Tone Mapping for Low Dynamic Range Content

Francesco Banterle[1], Alan Chalmers[2], and Roberto Scopigno[1]

[1] Visual Computing Laboratory, ISTI-CNR, Italy
[2] Visualisation Group, WMG, University of Warwick, UK

Abstract
In this paper, we present a novel parallel implementation of a high fidelity inverse tone mapping operator. Our method makes use of point based graphics to accelerate density estimation, and multi-core CPUs for extracting light sources. We show that our method can achieve real-time performance on a lower-end graphics card, with minimum loss of quality.

Categories and Subject Descriptors (according to ACM CCS): I.4.1 [Image Processing and Computer Vision]: Enhancement—I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations

1. Introduction

High Dynamic Range (HDR) image capturing is now becoming more common-place, including now being standard on DSLRs, compact cameras and even mobile devices. Although many problems have been solved for static HDR image, such as moving objects, compression etc., many challenges still remain, especially for HDR video. There are a few prototypes of HDR video-cameras such as the HDRv by SpheronVR [CBB*09], the AMP-HDR system by Contrast Optical [TKTS11], the RED HDRx (www.red.com), etc. However, their costs make them unaffordable except for a few specific high-standard applications, such as cinematography.

Despite the growth in HDR content, the vast majority of legacy footage is Low Dynamic Range (LDR). This paper addresses the problem of expanding or inverse/reverse tone mapping (iTM/rTM) LDR videos in real-time at High Definition (HD) resolution (1920×1080) by exploiting multi-core CPUs and graphics hardware. This enables the LDR content to be played on an available HDR display, providing an enhanced viewing experience.

2. Related Work

Recently, researchers have focused on developing new models for extending the dynamic range of LDR content in order for it to be used in applications such as Image Based Lighting (IBL) or for viewing on HDR displays.

Inverse Tone Mapping Operators (iTMOs) can be classified based on image processing techniques they use. For example, global operators [MAF*09] (the same expansion function for all pixels), or local operators [RTS*07] (the expansion function varies depending on the content). Reinhard et al. [RWP*10] and Banterle et al. [BADC11] provide good overviews on this topic.

In this work, we propose a novel implementation of the Banterle et al.'s method [BLDC08] (BiTMO). This method does not work in real-time, but it can be applied automatically to videos [BLDC08] and it can generate content that is close to the reference HDR content in terms of perception [BLD*09].

2.1. The iTMO method used

In this section, we present more in depth the Banterle et al.'s operator, see its pipeline in Figure 1.

Luminance Boosting. The first step is to linearise the signal (using the camera response function or removing gamma). The linearised image is then expanded using the inverse of photographic operator [RSSF02], which provides a controllable non-linear expansion curve. This is defined as

$$L_{\mathrm{w}}(\mathbf{x}) = L_{\mathrm{white}}^2 \beta \left(L_{\mathrm{d}}(\mathbf{x}) - 1 + \sqrt{\left(1 - L_{\mathrm{d}}(\mathbf{x})\right)^2 + \frac{4}{L_{\mathrm{white}}^2} L_{\mathrm{d}}(\mathbf{x})} \right) \quad (1)$$

DOI: 10.2312/conf/EG2013/short/041-044

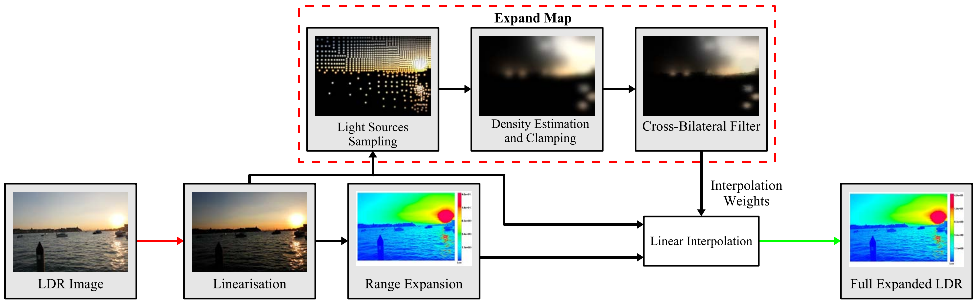

Figure 1: *The full pipeline of the Banterle et al.'s operator [BLDC08].*

where $L_{\mathrm{w}}(\mathbf{x})$ is the expanded luminance at pixel $\mathbf{x}$, $L_{\mathrm{d}}(\mathbf{x})$ is the input linearised LDR luminance, L_{white} determines the stretching of the curve, and β is a parameter and determines the desired maximum luminance output.

Expand Map Computation. As the first step, light source samples are extracted using a sampling algorithm, e.g. median-cut. Then, a smooth field, Λ, is generated using the density estimation as

$$\Lambda(\mathbf{x}, r_s) = \frac{1}{|\Omega| V(\Omega)} \sum_{p \in \mathscr{P}} K\left(\frac{\|\mathbf{x} - \mathbf{y}_p\|}{r_{\max}}\right) \Psi_p \qquad (2)$$

where $\mathbf{x}$ is the position of the current pixel to evaluate, Ω is the set of samples inside the sphere $(\mathbf{x}, r_s)$, V is the volume of Ω, Ψ_p is the power of the p-th sample, and K is a smoothing kernel. During sampling a few isolated samples can be generated due to the presence of large low luminance areas. These can produce isolated outliers, i.e. areas that do not need to be expanded. This issue can be solved by *clamping* which sets void the density estimation, if $\|\Omega\| < ns_{\min}$ where $ns_{\min}$ is a threshold.

Composition. Once the Λ is computed and filtered using the cross-bilateral filter, the expanded values, $L_{\mathrm{w}}(\mathbf{x})$, and linearised LDR luminance values, $L_{\mathrm{d}}(\mathbf{x})$, are linearly interpolated obtaining the final luminance values using $\Lambda(\mathbf{x})$ as weights.

3. Algorithm

The main idea of the algorithm proposed in this paper is to split computations of BiTMO between a multi-core CPU and a GPU. We achieved this by leaving the pure image processing computations to the GPU such as: linear interpolation, range expansion, linearisation, density estimation, and bilateral filter. The remaining computations, the extraction of light sources, and the flow control, we kept on the CPU.

Figure 2 shows the work-flow how we re-organised the operation in the BiTMO pipeline Figure 1 for parallel execution. The generation of each image/frame has two CPU threads. The first one, the top thread in Figure 2, simply calls GPU image processing functions and the density estimation. The second thread, the bottom thread in Figure 2, computes the median-cut sampling algorithm and stores the results in the *light source samples buffer*. This buffer is the input for the density estimation, and samples are stored as vertex buffers ready to be used by the GPU. This thread is running asynchronously to the first one and it is typically at least 5-8 frames ahead in order to buffer the results. In addition, there is an extra thread that feeds the *video frames buffer*, loading frames of the video from the hard drive.

We decided to use the classic graphics pipeline for our novel implementation of BiTMO. The main reason for doing so is that most of the operations can be mapped on this model in a straightforward way.

3.1. Image Processing Operations

Most of the operations of the iTMO are straightforward to implement on the GPU through writing a shader which evaluates a point-wise function. These operations include:

- *linearisation*: which is the application of a gamma function, in the best case, or an evaluation of a quintic polynomial applied to the current colour value.
- *range expansion*: which is the evaluation of a simple non-linear equation, see Equation 1.
- *linear interpolation*: which is a linear interpolation between the linearised image and the expanded one using the expand map as interpolation weight.

Note that the *range expansion* and *linear interpolation* can be performed in a single pass without the need to calculate them separately.

The most computational demanding image processing step of the algorithm is the cross bilateral filter, for reducing artifacts of the expand map around edges. This filter has a high complexity, $\mathscr{O}(k^2)$, where k is the size of the kernel in pixels. We used an approximation algorithm [BCCS12] in order to speed-up this test. This approximation is GPU-friendly, efficient, and it does not require a large amount of extra memory for filtering. In our experiments, we used as

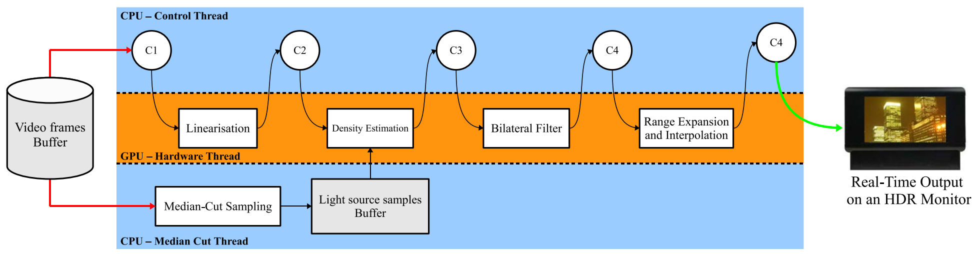

Figure 2: *The computation flow showing the co-operation between CPU and GPU threads for the iTMO. The first thread (top) is the CPU control of the program. The second thread (middle) is a thread which launches GPU operation in Direct3D. The third thread (bottom) computes median-cut sampling algorithm. In the case of videos, the samples are computed for 5-8 frames ahead and they are stored in a buffer.*

filter parameters $\sigma_s = 4$ (the spatial parameter) and $\sigma_r = 0.2$ (the range parameter). These parameters were chosen after pilot experiments; they allow to transfer strong edges.

3.2. Point-Based Density Estimation

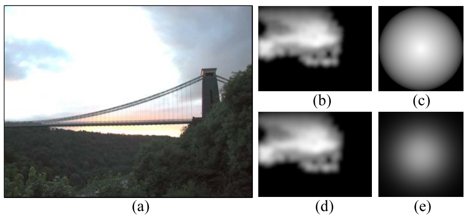

Figure 3: *An example of density estimation on a GPU using Cone and Gaussian kernel applied to Bristol Bridge image (image is courtesy of Greg Ward): a) The input LDR image. b) Density estimation using a Cone kernel. c) The Cone kernel with $\alpha = 1$ used in b) in the domain $[-1,1] \times [-1,1]$. d) Density estimation using a Gaussian kernel. e) The Gaussian kernel with $\sigma = 1$ used in d) in the domain $[-3,3] \times [-3,3]$.*

The density estimation on the CPU is typically computed in two steps. Firstly, for each evaluation point samples are gathered using a spatial query. Samples can be stored in a kD-tree or other spatial data structures. Secondly, Equation 2 is evaluated using the gathered samples.

To write a GPU spatial data structure is not a difficult task. However, a much simpler approach is to exploit the rasteriser power of drawing primitives in a fast way, especially for simple primitives such as points. Therefore, we redesigned the density estimation. The idea is to render a textured point for each light source sample, where the size of the point is equal to the radius r_s. This covers all pixels under the influence of that sample. The texture used for these points is a discretised smoothing kernel used in Equation 2, which allows to perform filtering, see Figure 3. Furthermore, the accumulation of values, when two points are overlapped, is achieved by disabling the Z-buffer and enabling the alpha blending. Finally, the implementation on a GPU is straightforward, because there are only two steps to compute: load samples into a vertex buffer, and render points from the vertex buffer using a short shader.

3.2.1. Extension to Videos

The method described before is designed for 2D still images, but the extension to videos requires only a few modifications. The first step is to add samples from backward and forward frames to the vertex buffer used for rendering. Furthermore, each sample in the vertex buffer has an attribute $t \in [-5,5]$, which identifies the frame of the sample, where $t = 0$ is the current frame. The second step is to discretise the 3D smoothing kernels. The idea is to store each slice in time of the 3D kernel into a 3D texture. When textured points are rendered, the access to the correct time slice of the 3D texture is achieved using the extra attribute t added to each sample. Note that the CPU version needs all frames to create the 3D-tree used for the spatial query. The GPU implementation needs only a few backward and forwards frames, up to 5, which makes it suitable for real-time streaming of content with buffering.

3.2.2. Light Sources Samples Clamping

On the CPU, to discard the density evaluation, if *clamping* is needed, is straightforward. The spatial radius query for gathering samples returns the number of samples in the volume. However, this process can be less intuitive on the GPU using the graphics pipeline. One solution would be to count samples using the alpha channel. Therefore, *clamping* needs a second pass for discarding evaluations. This solution slows down performance, because we need extra memory and a second pass for checking the counting. A more efficient solution is *precomputed clamping* which is directly performed on samples. For each sample $\mathbf{x}$, $\Omega = \left\{\mathbf{y} : \|\mathbf{x}-\mathbf{y}\| < 2r_s\right\}$ is computed. Then, the number of samples in Ω is tested if there is enough for a density estimation in its area, $|\Omega| \geq ns_{\text{min}}$.

4. Results

We evaluated our novel GPU implementation against a CPU version of BiTMO for still images and videos. While the CPU version was implemented in C++ and optimized using

OpenMP, the GPU one was implemented in C++ using Direct3D9c for accessing the GPU capabilities and Boost for threading management. Both versions were timed on an Intel Extreme Quad core at 2.4 Ghz (only three threads were used), with 2Gb of main memory, and a GeForce 8800GTX with 768 Mb of memory under Windows XP-32 SP3.

Resolution	CPU	GPU	CPU^v	GPU^v
360×240	510	10.49	990	11.92
720×480	1850	14.30	3560	16.40
1280×720	7730	18.58	11680	21.01
1920×1080	26930	32.69	29270	35.16

Table 1: *The timing results of the performance of the CPU and GPU algorithm for static images and videos (^{v}) in milliseconds (ms).*

The CPU and GPU versions were timed for still images and videos. The timing results are shown in Table 1. As can be seen the GPU version can achieve on average 28.44 fps (never dropping under 24 fps) at high definition (HD) resolution (1920×1080) while the CPU version can only achieve 1 fps at low resolution (360×240). The main bottleneck of this implementation is the low band for transferring frames from the main memory to the GPU's one. This can be an issue for images larger than HD resolution because less memory is available for buffering the video stream. However, the algorithm can expand content in real-time for DVDs and HD content which are, at the time of writing, the standard for digital entertainment. Note that the difference between the GPU version for still images and videos is quite small, on average 7-8fps. This is due to the fact that more samples in the density estimation are used, on average 8 times more for frame in the sequence in Table 1. Finally, our novel parallel implementation achieves large speed-ups, e.g. it is 809 times faster than CPU version using HD content.

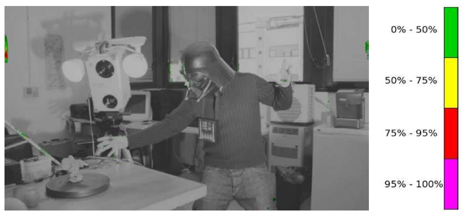

Figure 4: *An example of HDR-VDP comparison between frames of an HD video generated using the CPU version of the algorithm and our novel method. The HDR-VDP error is less than* 1%.

Finally, we tested the quality of this approximation compared to the original method. We employed HDR-VDP [MDMS05] a popular metric for assessing images against a reference image in HDR. From our experiments, we found out that on average the HDR-VDP is less than 1% for $P(X) = 0.75$ (the percentage of pixels in the image that the human-eye can spot as different in the whole image with a probability of 75%), see Figure 4.

5. Conclusions and Future Work

We proposed a novel parallel implementation of the Banterle et al. [BLDC08] algorithm for producing high fidelity inverse tone mapped content for still images and videos. We modified the original algorithm to efficiently exploit the performance of GPUs. Despite these modifications, the quality has not significantly changed compared to the original algorithm. The new method works in real-time for HD content even on old graphics hardware. Performance on this hardware obtained large speed-ups compared to the CPU implementation.

References

[BADC11] BANTERLE F., ARTUSI A., DEBATTISTA K., CHALMERS A.: *Advanced High Dynamic Range Imaging: Theory and Practice*, first edition ed. AK Peters, Ltd, 2011.

[BCCS12] BANTERLE F., CORSINI M., CIGNONI P., SCOPIGNO R.: A Low-Memory, Straightforward and Fast Bilateral Filter Through Subsampling in Spatial Domain. *Computer Graphics Forum 31*, 1 (2012).

[BLD*09] BANTERLE F., LEDDA P., DEBATTISTA K., ARTUSI A., BLOJ M., CHALMERS A.: A psychophysical evaluation of inverse tone mapping techniques. *Computer Graphics Forum 28*, 1 (March 2009), 13–25.

[BLDC08] BANTERLE F., LEDDA P., DEBATTISTA K., CHALMERS A.: Expanding low dynamic range videos for high dynamic range applications. In *SCCG '08: Proceedings of the 4th Spring Conference on Computer Graphics* (New York, NY, USA, 2008), ACM, pp. 349–356.

[CBB*09] CHALMERS A., BONNET G., BANTERLE F., DUBLA P., DEBATTISTA K., ARTUSI A., MOIR C.: High-dynamic-range video solution. In *ACM SIGGRAPH ASIA 2009 Art Gallery & Emerging Technologies: Adaptation* (New York, NY, USA, 2009), SIGGRAPH ASIA '09, ACM, pp. 71–71.

[MAF*09] MASIA B., AGUSTIN S., FLEMING R. W., SORKINE O., GUTIERREZ D.: Evaluation of reverse tone mapping through varying exposure conditions. *ACM Trans. Graph. 28*, 5 (2009), 1–8.

[MDMS05] MANTIUK R., DALY S., MYSZKOWSKI K., SEIDEL H.-P.: Predicting visible differences in high dynamic range images - model and its calibration. In *Human Vision and Electronic Imaging X, IST SPIE's 17th Annual Symposium on Electronic Imaging* (2005), Rogowitz B. E., Pappas T. N., Daly S. J., (Eds.), vol. 5666, pp. 204–214.

[RSSF02] REINHARD E., STARK M., SHIRLEY P., FERWERDA J.: Photographic tone reproduction for digital images. *ACM Trans. Graph. 21*, 3 (2002), 267–276.

[RTS*07] REMPEL A. G., TRENTACOSTE M., SEETZEN H., YOUNG H. D., HEIDRICH W., WHITEHEAD L., WARD G.: Ldr2hdr: on-the-fly reverse tone mapping of legacy video and photographs. *ACM Trans. Graph. 26*, 3 (2007), 39.

[RWP*10] REINHARD E., WARD G., PATTANAIK S., DEBEVEC P., HEIDRICH W., MYSZKOWSKI K.: *High Dynamic Range Imaging, Second Edition: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[TKTS11] TOCCI M., KISER C., TOCCI N., SEN P.: A Versatile HDR Video Production System. *ACM Transactions on Graphics 30*, 4 (2011).