CGI 2008 Conference Proceedings manuscript No.
(will be inserted by the editor)

Marcos Slomp · **Manuel M. Oliveira**

Real-Time Photographic Local Tone Reproduction Using Summed-Area Tables

Abstract High dynamic range (HDR) rendering is becoming an increasingly popular technique in computer graphics. Its challenge consists in mapping the resulting images' large range of intensities to the much narrower ones of the display devices in a way that preserves contrastive details. Local tone mapping operators effectively perform the required compression by adapting the luminance level of each pixel with respect to its local neighborhood. While they generate significantly better results when compared to global operators, their computational costs are considerably higher, which has prevented their use in real-time applications. This paper presents a real-time technique for photographic local tone reproduction that runs entirely on the GPU and is significantly faster than existing implementations that produce similar results. Our approach is based on the use of summed-area tables for accelerating the convolution of the local neighborhoods with a box filter and provides an attractive solution for HDR rendering applications that require high performance without compromising image quality.

Keywords local tone-mapping operator · HDR images · GPU technique · real-time rendering · summed-area tables

Fig. 1 Tone-mapped images using Reinhard et al.'s photographic local operator. The image on the left was generated with a software implementation using Gaussian filtering and 8 adaptation zones. The image on the right was obtained with our summed-area-table technique also using 8 adaptation zones. Note the similarity between them. For this 1024×1024 image, the software implementation runs at 0.19 fps on a 2.21 GHz PC with 2.0 GB of RAM. Our technique renders the same image at 102 fps on a GeForce 8800 GTX graphics card.

1 Introduction

High dynamic range (HDR) rendering is becoming an increasingly popular technique in computer graphics [3]. However, proper presentation of HDR content requires that the display devices support the dynamic range of the images. Otherwise, the resulting pictures will look either underexposed or overexposed. Currently, HDR displays [22, 24, 25] are still very expensive and have limited dynamic range. Compressing such large luminance ranges so that they fit the much narrower ones supported by low dynamic range (LDR) display devices, in a way that the resulting images are perceptually equivalent the original HDR scenes is known as *digital tone mapping* [22, 26].

Digital tone-mapping operators can be classified as *global* or *local* [22]. Global operators use the same parameter values to process all the pixels in the image [6, 7, 11, 17, 21, 23, 26, 27]. In contrast, local operators try to find an optimal luminance value for each pixel considering a variable-size neighborhood around the pixel [1,4,9,10,18,19,21,28]. For images with large dynamic ranges, local operators tend to produce significantly better results in terms of contrast preservation. This comes from the fact that the amount of compression is locally adapted to the different regions of the input images. For this reason, local tone-mapping operators are computationally more expensive than global ones. An in-depth discussion of the various tone-mapping algorithms can be found in [22].

The photographic local tone mapping operator proposed by Reinhard et al. [21] produces very plausible results, uses few parameters, and requires no user intervention. In order to compress an image's dynamic range, it requires comput-

Marcos Slomp
Instituto de Informática - UFRGS
E-mail: slomp@inf.ufrgs.br

Manuel M. Oliveira
Instituto de Informática - UFRGS
E-mail: oliveira@inf.ufrgs.br

ing the average luminance around each pixel at different scales. Reinhard et. al used a luminance-perception model presented by Blommaert and Martens [2], which is essentially a set of differences of Gaussian-filtered luminance images. Unfortunately, convolving an image with Gaussian kernels of various scales is a computationally expensive operation. Thus, in practice, interactive applications currently limit themselves to global operators. The exception is the work of Goodnight et al. [12], which efficiently implemented the 2D Gaussian convolution using separable 1D kernels on the GPU. However, despite all their optimization efforts, their technique could handle only a few adaptation zones while still running at interactive rates.

This paper presents a new GPU-based technique for performing Reinhard et al.'s photographic local tone reproduction in real time. Our approach uses a summed-area table [5] to efficiently evaluate an approximation of Blommaert and Martens's luminance-perception model [2], which is the most expensive part of the operator. Our technique produces results that are comparable to a software-based implementation of the local operator (Figure 1), but is significantly faster than previous GPU implementations of the algorithm. Our approach is based on the following key observations:

- Reinhard et al. [21] used normalized differences of Gaussian-filtered luminance images to identify the largest approximately isoluminant region around each pixel. This is performed by checking whether these normalized differences are smaller than a specified threshold. Note that the actual (approximately) isoluminant region around a pixel can have arbitrary shapes, whereas the used Gaussian kernels are rotationally symmetric and do not perfectly fit these regions. Thus, the algorithm computes an approximate scale for each region. Similar results can be obtained using normalized differences of box-filtered luminance images.
- Since a convolution with a box filter is just an average over a rectangular region, it can be efficiently implemented using a summed-area table [5], for a fraction of the cost of a Gaussian convolution. Compared to the approach described by Goodnight et al. [12], we achieve an average speedup of about eight times, while producing very similar results. Compared to Krawczyk et al.'s approach [16], our technique is about twice as fast and reduces the occurrence of halo artifacts.

The remaining of this paper is organized as follows: Section 2 discusses some related work and reviews the concept of summed-area tables [5]. Section 3 reviews Reihard et al.'s photographic local tone-mapping operator, which is the basis of our implementation. Section 4 presents the details of our technique. Section 5 discusses some of our results, and Section 6 summarizes the paper.

2 Related Work

Goodnight et al. [12, 13] investigated the implementation of some local tone-mapping algorithms in programmable graphics hardware. They observed that Fattal et al.'s operator is difficult to implement in order to achieve interactive rates, even on modern GPUs [13]. They also described implementations for both the global and the local photographic operators of Reinhard et al. [12, 21], and showed how the technique could be modified to support time-dependent luminance adaptation. For the local operator, Goodnight et al. efficiently implemented the 2D Gaussian convolution using a two-pass 1D approach. Their GPU implementation was written using the ATI 9800 Pro assembler [12]. According to their reported results, for images with 512x512 pixels or more, their technique only runs at interactive rates when using two adaptation zones (*i.e.*, two levels of the Gaussian-filtered luminance images). The use of only two adaptation zones tends to cause the loss of some important details in the resulting images.

In order to reduce the cost of the convolution in Goodnight et al.'s approach, Krawczyk et al. [16] implemented a simplified version of Goodnight et al.'s technique that consisted of downsampling the luminance images to $1/4$, $1/16$, and $1/64$ of their original dimensions and performing the convolution using smaller approximate Gaussian kernels. The blurred images were then upsampled and used to compute the normalized differences of Gaussian-filtered images. Although this strategy can significantly speedup the algorithm, it has some inherent limitations: the downsampling blurrs high-contrast edges and the upsampling that follows the convolution adds some extra blurring. The occurrence of excessive blurring across the high contrast edges tends to introduce clearly noticeable halo artifacts. Although all local tone-mapping operators are prone to this kind of artifacts, good operators try to minimize their occurrence. Due to current GPU restrictions, Krawczyk et al. perform both the downsampling and the upsampling using shaders. The need to store a downsampled texture together with two upsampled and a temporary result texture leads to some large memory requirements, which is about $2\times$ bigger than the memory consumption of our technique.

Regarding previous uses of box filters with tone-mapping operators, Pattanaik and Yee [18] used a circular approximately box-shaped kernel with fixed radius (7 pixels) in their bilateral-filtering algorithm.

2.1 Summed-Area Tables

A *summed-area table* (SAT) [5] is a cumulative tabular data structure S that stores a 2D prefix sum of another table T. More formally, each cell S_{ij} of S stores the sum of all elements T_{pq} from T such that $p \leq i$ and $q \leq j$ (Equation 1).

$$S_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} T_{ij} \tag{1}$$

Figure 2 (right) shows a SAT computed for the table shown to its left. SATs provide an efficient way to apply a spatial averaging filter (box filter) to any rectangular area of an image in constant time, requiring only four fetches (Figure 3).

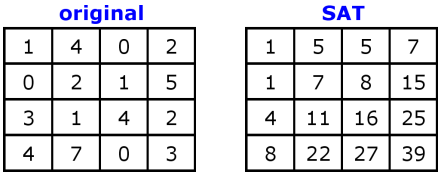


Fig. 2 Each element Sij of a summed-area table (right) is is obtained by summing the elements of the original table T (left) according to Equation 1.

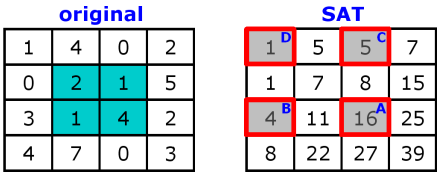


Fig. 3 Computing the average of an arbitrry rectangular area (highlighted) in the original table. First, fetch the four cells (A, B, C, D) delimiting the desired area (right). The average is then obtained by applying Equations 2 and 3 to the retrieved values.

According to Equations 2 and 3, for the example shown in Figure 3, one has $Sum = 16-4-5+1 = 8$ and $Average = 8/(2*2) = 2$.

$$Sum = A - B - C + D \tag{2}$$

$$Average = \frac{Sum}{width * height} \tag{3}$$

Summed-area tables can be efficiently generated in parallel using graphics hardware [14] by recursive doubling [8]. The tables are stored into textures and the algorithm consists of two stages: one for horizontal and another for vertical accumulation. The technique is based on ping-pong rendering and the process is illustrated in Figure 4. At each stage, $log_k(w) + log_k(h)$ steps are required, where w and h are the image width and height, respectively, and k is the number of cells that are accumulated at each step. Any cell outside the limits of the table is interpreted as a zero value cell. This can be implemented by rendering a black edge in the texture (clamp to edge) or by setting a black border color to it (clamp to border). Hensley et al. [14] pointed out that the optimal number of accumulations per step (k) depends on a balance between the number of rendering passes and the computational cost at each pass. Such a balance is closely related to the render-target switch overhead of the intended hardware platform, and to the texture cache accuracy.

In a subsequent work, Hensley et al. [15] used first, second and third order summed-area tables to filter HDR environment maps with box, Bartlett, and cubic filters, respectively. The filtered images were used to compute approximations for diffuse and Phong BRDFs in interactive rendering applications. An n-th order summed-area table (SAT^n) is obtained by recursively applying Equation 1 to its own output $(n-1)$ times. Note that the values stored in high-order SATs tend to grow very quickly, especially if the content of the original table T is HDR. This imposes some limitations on the size and dynamic range of the input images. In order to mitigate this and other loss of precision issues, Hensley et al. used an offset technique introduced by them in [14], which essentially consists in subtracting the mean value of the SAT from all its elements. In our approach, we compute a first-order SAT for HDR content on the fly. In our case, however, each cell of the original table T stores a scaled luminance value (Equation 5), which is essentially the ratio between the original pixel's luminance and the scene's average luminance. As a result, the values accumulated by our SATs tend to be a few orders of magnitude smaller than the ones stored by Hensley et al.'s technique [15] in a first-order SAT.

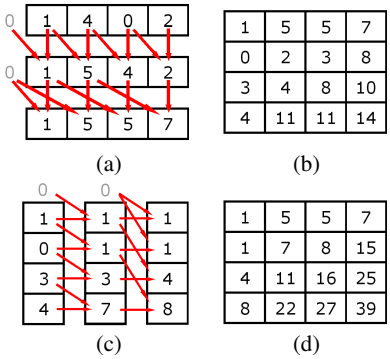


Fig. 4 SAT generation on the GPU. The tables are stored into textures and the SAT is generated using ping-pong rendering. (a) Horizontal accumulation steps applied to the first row of the table shown in Figure 2 (left). Applying this process to all rows will result in the intermediate table shown in (b), which will be used as input to the vertical stage. Similarly, (c) illustrates the vertical accumulation steps applied to the first column of the intermediate table. (d) The result of the vertical pass applied to all columns of the intermediate table is the SAT shown in Figure 2 (right).

3 The Photographic Local Operator

This section briefly reviews Reinhard et al.'s photographic local tone reproduction operator [21], which consists of the following steps: first, it computes the average log luminance of the input HDR image as

$$\widetilde{L} = \exp\left(\frac{1}{N}\sum_{x,y}\log\left(L(x,y)+\delta\right)\right) \tag{4}$$

where N is the number of pixels in the image, δ is a small value to avoid $\log(0)$, and $L(x,y)$ is the HDR luminance of the pixel with coordinates (x,y), computed from its RGB values:

$$L(x,y) = 0.30*R(x,y) + 0.59*G(x,y) + 0.11*B(x,y)$$

It then computes a scaled luminance $L_s(x,y)$ for each pixel as

$$L_s(x,y) = \frac{\alpha}{\widetilde{L}}L(x,y) \tag{5}$$

where α is the *key-value*, a parameter that determines if the image will favor dark or bright areas. Typically, a value of $\alpha = 0.18$ is used in automatic exposure control in cameras [12]. Reinhard [20] describes a procedure for estimating the value of the parameter α automatically.

Reinhard et al.'s global operator is defined by Equation 6, where the final luminance $L_{DG}(x,y)$ of each pixel $p(x,y)$ is obtained by dividing its scaled luminance $L_s(x,y)$ by $1 + L_s(x,y)$.

$$L_{DG}(x,y) = \frac{L_s(x,y)}{1+L_s(x,y)} \tag{6}$$

The local operator is obtained directly from Equation 6 by replacing $L_s(x,y)$ in the denominator with the weighted average $V(x,y,s)$ of the approximately isoluminant neighborhood of pixel $p(x,y)$. From the scaled luminance image L_s, a set of Gaussian-filtered images is generated, each at a different scale s:

$$V(x,y,s) = L_s(x,y) \otimes Gauss(x,y,s) \tag{7}$$

where $Gauss(x,y,s)$ is a rotationally-symmetric Gaussian kernel and $\otimes$ is the convolution operator. The hierarchy of filtered images defined by Equation 7 is used to compute normalized differences, according to the luminance-perception model proposed by Blommaert and Martens [2]:

$$W(x,y,s_i) = \frac{V(x,y,s_i) - V(x,y,s_{i+1})}{2^{\phi}\alpha/s^2 + V(x,y,s_i)} \tag{8}$$

Given some threshold ε, the technique looks for the largest scale s_{max} for which

$$|W(x,y,s_{max})| < \varepsilon \tag{9}$$

Such a scale represents the largest Gaussian-modulated neighborhood around pixel $p(x,y)$ for which no substantial luminance variation occurs. The parameter ϕ represents a sharpening factor. Default values used for parameters ε and ϕ are 0.05 and 8, respectively [21].

The display luminance $L_{DL}(x,y)$ of pixel $p(x,y)$ obtained with the local operator is then defined as

$$L_{DL}(x,y) = \frac{L_s(x,y)}{1+V(x,y,s_{max})} \tag{10}$$

The final $RGB_{DL}(x,y)$ color channels for pixel $p(x,y)$ are computed from its corresponding display luminance $L_{DL}(x,y)$ using Equation 11. The pixel coordinates are omitted here to simplify the expression:

$$RGB_{DL} = \left(\left(\frac{R}{L}\right)^{\gamma} L_{DL}, \left(\frac{G}{L}\right)^{\gamma} L_{DL}, \left(\frac{B}{L}\right)^{\gamma} L_{DL}\right) \tag{11}$$

where γ performs gamma correction and typically ranges from 0.4 to 0.8 [12]. R, G, B, and L are respectively the red, green, blue, and luminance channels of pixel $p(x,y)$ in the input HDR image.

4 The Local Photographic Operator Using SAT

Following the observations described at the end of Section 1, we use normalized differences of box-filtered luminance images to estimate the scale of the approximate isoluminant neighborhood of each pixel. Thus, our GPU implementation of Reinhard et al.'s local tone-mapping operator follows the same steps presented in Section 3, with two minor modifications: (i) it evaluates Equation 7 using box filters instead of Gaussian ones, and (ii) it uses a different threshold value ε_B instead of ε in Equation 9.

We use a summed-area table to compute the convolutions at all scales in constant time and rewrite Equation 7 as

$$V(x,y,s) = L_s(x,y) \otimes Box(x,y,s) \tag{12}$$

The need for a different threshold value ε_B follows from the different shapes of the two kernels. The weights associated with a Gaussian kernel get smaller as a function of the distance to the kernel's center, thus reducing the severity of halo artifacts across high-contrast edges. A box filter, on the other hand, weights the contributions of all pixels in the neighborhood equally. Thus, the tone-mapped images produced with the use of box filters are more prone to noticeable halos than the ones produced with Gaussian filters of the same scale. We reduce the occurrence of halos in the resulting images by reducing the threshold used to compare against the normalized luminance differences (Equation 9). According to our experience, a threshold value of $\varepsilon_B = 0.025$ reduces the noticeable halos to the same level as the one perceived with the threshold $\varepsilon = 0.05$ for the Gaussian case.

4.1 Implementation Details

The pipeline of our technique starts by rendering an HDR scene into a floating-point texture I (either RGB or RGBA). It then proceeds with the following steps:

1. For each pixel $p(x,y)$ in I, compute $log(L(x,y))$ and store it in a new rectangular floating-point texture I_{logL} at (x,y);
2. Compute the average log luminance (Equation 4) of I_{logL}. This is performed by generating a mip-map pyramid for I_{logL}, retrieving the value stored at the 1×1 level, and using it to evaluate the exp function. Since the input textures are often not powers of two, the mip-map pyramid is built inside a shader using $log_2(max(width, height))$ passes;
3. Generate the scaled luminance map according to Equation 5. This map will be used as input to compute the SAT;
4. Generate the SAT using the algorithm described in [14];
5. Finally, we use the resulting SAT to evaluate Equations 12, 8, and 9 at various scales, until the proper s_{max} scale is found. The luminance of the individual pixels are then compressed using Equation 10, and their corresponding display RGB values are obtained using Equation 11.

In all examples shown in this paper, we used a 16-bit per-channel representation. According to our experience, the performance penalty when using full-precision (32-bit) luminance textures is minimal (about 2 fps on a GeForce 6800). Figure 5 shows a side-by-side comparison of pairs of filtered luminance images at the eight scales used by the algorithm. In order to make the comparison more challenging, we have chosen an image with lots of small high-frequency details. For each pair, the image on the left was obtained using a Gaussian filter, while the one on the right was produced with a box filter using a SAT. Since such images are exact representations of the results produced by Equations 7 and 12, respectively, they provide a good indication of how similar our results will be compared to the ones produced by Reinhard et al.'s Gaussian local operator.

Figure 6 shows a visualization of the maximum scale s_{max} computed for each pixel of the image, based on the normalized differences (Equations 8 and 9) of the filtered images shown in Figure 5. Brighter pixels indicate bigger scales. The image on the left corresponds to the Gaussian filter, while the one on the right corresponds to the box filter. For this example, the same threshold $\varepsilon = 0.05$ was used for both cases. A visual inspection of these two images reveals that the one corresponding to the box filter is slightly darker, indicating the use of smaller neighborhoods. The resulting tone-mapped images obtained with the use of the two techniques are shown in Figure 7.

If the box kernel is only partially inside the SAT (a common situation when applying the convolution to pixels in the border of the scaled luminance image), we compute the average of the valid pixels only.

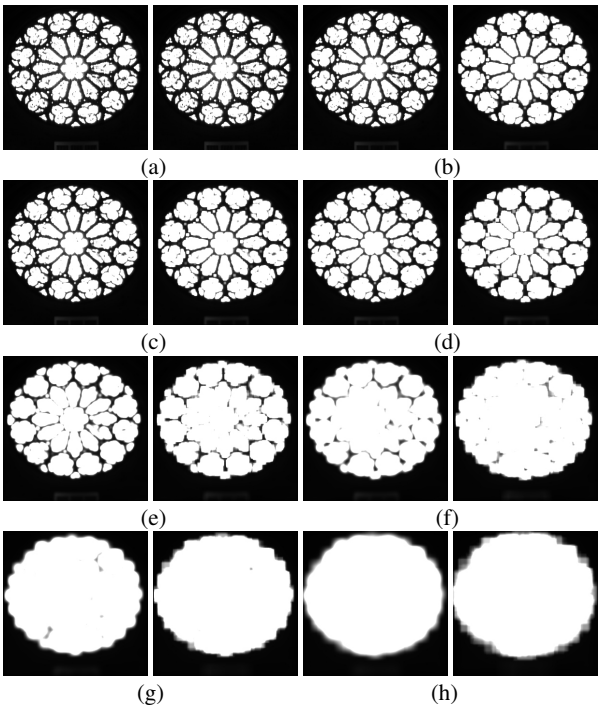


Fig. 5 Side-by-side comparison of pairs of filtered luminance images at several different scales. The images of the left of each pair were obtained using Gaussian filters (Equation 7), whereas the images on the right of each pair were produced with box filters (Equation 12). The sizes of the filter kernels (in pixels) used for the 8 adaptation zones are: (a) 1×1, (b) 3×3, (c) 5×5, (d) 7×7, (e) 11×11, (f) 17×17, (g) 25×25, and (h) 39×39.

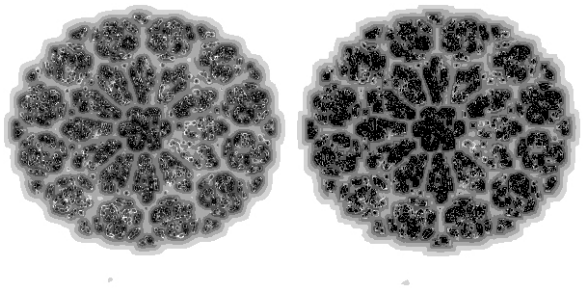


Fig. 6 Maximum scale s_{max} values computed for each pixel according to the filtered images shown in Figure 5. Brighter values indicate larger scales. Results obtained with: Gaussian filter (left) and box filter (right). The same threshold $\varepsilon = 0.05$ was used for both filters.

5 Results

We have implemented our SAT-based approach using Cg and C++ and used it to tone map a large number of HDR images. We have compared our results with the ones produced by a CPU implementation of Reinhard et al.'s local operator, as well as with our GPU implementation of the technique described in [16]. All performance measurements presented in the paper were obtained on a 2.21 GHz PC with 2 GB of memory and a GeForce 8800 GTX with 768 MB of memory and PCI-Express.

Table 1 compares the performance of our technique on two different GPUs (GeForce 6800 and GeForce 8800 GTX) for images with different dimensions and for different values of the parameter k, which defines the number of cells accumulated at each step of the SAT generation process (see Section 2.1). For the GeForce 8800 GTX, we provide the times (in seconds) for the SAT generation (SAT - step 4 of the algorithmic description presented in Section 4.1) and for performing the actual tone mapping (TM - step 5). From Table 1, one observes that best performance is achieved by using 4 accumulations per step ($k = 4$). Higher values of k reduce the number of shader passes, but increase the number of texture fetches at each step.

Krawczyk et al.'s technique [16] is much faster than Goodnight et al.'s approach [12]. Table 2 compares its performance with ours for images of different dimensions on a GeForce 8800 GTX, and shows that our technique is significantly faster than Krawczyk et al.'s [16].

We have also compared the quality of the results produced by both approaches against the ones obtained with a software implementation of Reinhard et al.'s local operator [21]. Figure 8 shows a tone-mapped image produced using this software implementation. The top row of Figure 9 shows the results produced by Krawczyk et al.'s (a and b) and by our approach (c and d) applied to the same input

Fig. 7 Comparison between the resulting tone-mapped images obtained with CPU-based Gaussian operator (left) and the GPU-based SAT operator (right).

		6800	8800 GTX		
		fps	fps	SAT	TM
256x256	k=2	83	322	0.0008	0.0003
	k=4	92	385	0.0005	0.0003
	k=8	79	390	0.0005	0.0003
512x512	k=2	28	244	0.0023	0.0011
	k=4	30	243	0.0016	0.0011
	k=8	25	228	0.0015	0.0011
1024x1024	k=2	5	102	0.0098	0.0043
	k=4	6	102	0.0061	0.0043
	k=8	4	80	0.0076	0.0043

Table 1 Performance of our technique on two different GPUs, for images with different dimensions, and for different values of the SAT generation parameter k. SAT represents the SAT generation time (steps 1 to 4 of the algorithmic description presented in Section 4.1). TM is the time for performing the actual tone mapping (step 5). These times are expressed in seconds.

dimensions	Krawczyk et al.	Our Technique
256x256	189 fps	385 fps
512x512	157 fps	243 fps
1024x1024	74 fps	102 fps

Table 2 Performance comparison between Krawczyk et al.'s technique and our technique on a GeForce 8800 GTX and using $k = 4$. Eight adaptation zones were used in both cases.

HDR image. For both techniques, we used threshold values $\varepsilon = 0.05$ and $\varepsilon = 0.025$. Note the clearly visible halos around the sun and next to the mountains silhouette on Figures 9(a) and (b). The bottom row shows the corresponding per-pixel perceptual error computed for each image on the top row using the S-CIELAB color difference metric [29] and using the image shown in Figure 8 as reference. The S-CIELAB is a spatial extension to the CIE L*a*b* DeltaE color difference metric that incorporates the pattern-color sensitivity measurements by Poirson and Wandell [29]. In these error images, white means bigger errors. Note that for both threshold values our technique generated smaller errors, and produced better results for $\varepsilon = 0.025$.

Figures 1, 10 and 11 illustrate the use of our technique on several classical HDR images. Figure 1 shows the result produced by our approach ($k = 4$ and $\varepsilon = 0.05$) on a portion of the *Foggy Parking Lot* image and compares it with the result obtained with a software implementation of Reinhard et al.'s local operator. Note how similar the two results are. For this 1024×1024 image, our technique runs at 102 fps, whereas the software implementation runs at 0.19 fps.

According to our experience, although the use of the threshold ε between the values 0.05 and 0.025 can make some difference for certain images, such as the one shown in Figure 9, for most images they only produce negligible differences. Figure 10 shows the *Memorial* image generated using $\varepsilon = 0.05$. The images shown in Figure 11 were produced using $\varepsilon = 0.025$. For these examples, the corresponding results obtained with $\varepsilon = 0.05$ are almost identical.

Fig. 8 Tone-mapped image generated with a software implementation of Reinhard et al.'s algorithm using $\varepsilon = 0.05$.

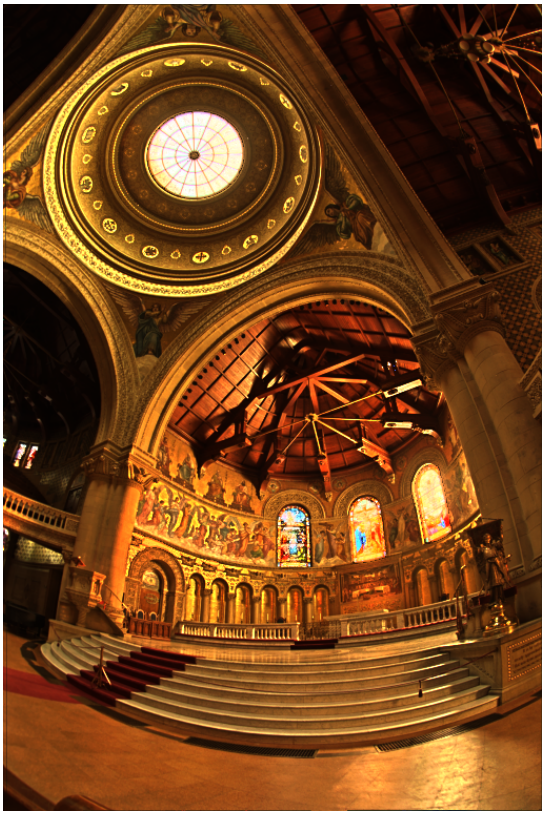
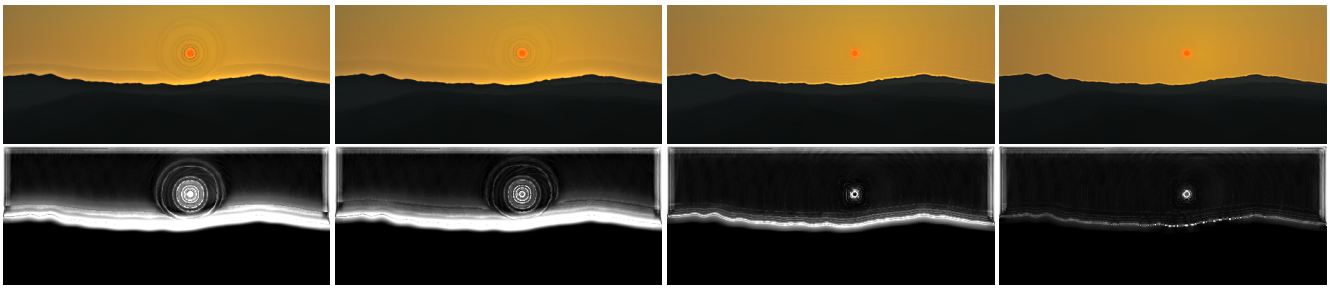
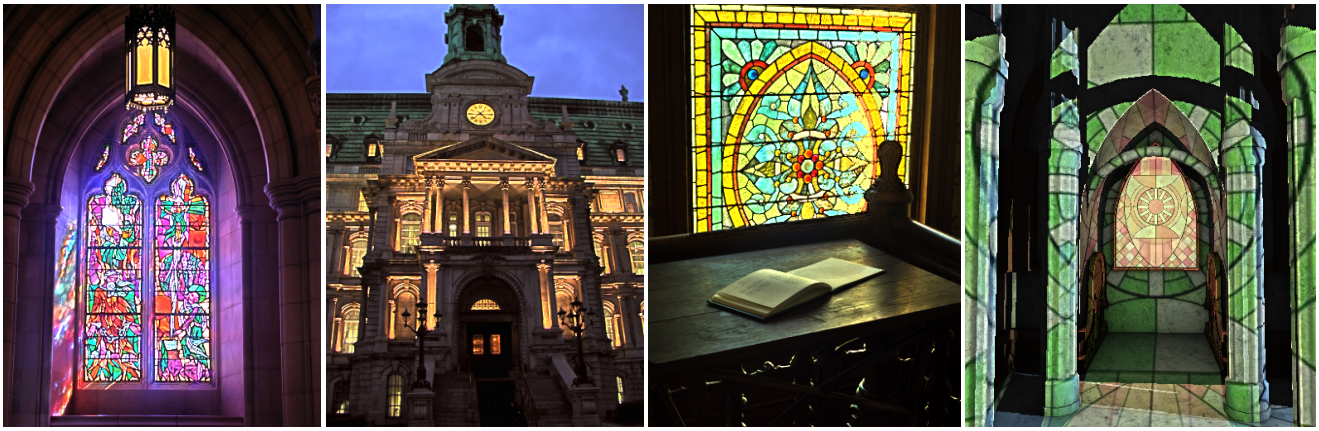


Fig. 10 Tone-mapped version of the Memorial image obtained using our technique with $\varepsilon = 0.05$.



(a) (b) (c) (d)

Fig. 9 Comparison of the results produced by the GPU technique described in [16] and by our approach for a given HDR image. Top row: Krawczyk et al.'s technique using (a) $\varepsilon = 0.05$ and (b) $\varepsilon = 0.025$, and our technique using (c) $\varepsilon = 0.05$ and (d) $\varepsilon = 0.025$. Note the clearly visible halos around the sun and on the mountain silhouette in (a) and (b). Bottom row: per-pixel perceptual error computed using the S-CIELAB metric described in [29] and using the image shown in Figure 8 as reference. White means bigger errors. Reinhard et al. [21] noticed that $\varepsilon = 0.05$ produced better results for their method. The presented technique also adapts well with the same threshold, but we noticed that we can enhance the image quality using a lower value such as $\varepsilon = 0.025$.



(a) (b) (c) (d)

Fig. 11 Examples of tone-mapped images produced with our technique using $\varepsilon = 0.025$. (d) shows a chapel scene which is part of our accompanying demo.

6 Conclusion and Future Work

We presented an efficient GPU technique to implement an approximation to Reinhard et al.'s photographic local tone-mapping operator in real time. Our approach uses box filters instead of Gaussian ones, and uses summed-area tables to efficiently evaluate the required convolutions. By showing a side-by-side comparison of the filtered luminance maps generated with both filters at various scales and for an image containing small high-frequency details, we have provided some intuitive explanation of why our results are visually similar to the ones generated by Reinhard et al.'s original technique.

We have shown that our approach is significantly faster than previously known GPU implementations of the same operator, requires less memory and is also considerably simpler to implement. In terms of image quality, we have shown that our technique is less prone to halo artifacts than the technique described by Krawczyk et al. [16]. All in all, our technique provides a practical and attractive alternative for HDR-rendering applications that require high performance but do not want to compromise image quality with the use of global tone-mapping operators.

The results of our research open some interesting avenues for future exploration. For instance, SAT-based approximations of the luminance-perception model of Blommaert and Martens [2] could be also used by other performance-demanding applications.

Acknowledgements Marcos Slomp was supported by a CNPq-Brazil fellowship (Processo 130941/2006-4). Manuel M. Oliveira has a CNPq-Brazil fellowship (Processo 305613/2007-3). We are deeply thankful to Igor Kenne Braga for modeling the chapel used in our demo. We also want to thank Erik Reinhard, Rui Wang and Nolan Goodnight for providing valuable information about their methods, Xuemei Zhang for making his S-CIELAB MATLAB code available, Giovane R. Kuhn for his assistance with the S-CIELAB software, and Eduardo Costa for producing the video. NVIDIA kindly donated the GeForce 8800 GTX graphics card used in this work. Thanks also to the several researchers who made their HDR images publicaly avaible. Microsoft Brazil provided additional support.

References

1. Ashikhmin, M.: A tone mapping algorithm for high contrast images. In: Proc. of EUROGRAPHICS Workshop on Rendering (EGRW'02), pp. 145–156 (2002)
2. Blommaert, F.J.J., Martens, J.B.: An object-oriented model for brightness perception. Spatial Vision **5**(1), 1541 (1990)
3. Blythe, D.: The Direct3D 10 system. ACM Trans. Graph. **25**(3), 724–734 (2006)
4. Chiu, K., Herf, M., Shirley, P., Swamy, S., Wang, C., Zimmerman, K.: Spatially nonuniform scaling functions for high contrast images. In: Proc. of Graphics Interface '93, pp. 245–253 (1993)
5. Crow, F.C.: Summed-area tables for texture mapping. In: Proc. of SIGGRAPH'84, pp. 207–212. ACM Press (1984)
6. Devlin, K., Reinhard, E.: Dynamic range reduction inspired by photoreceptor physiology. IEEE Trans. on Visualization and Computer Graphics **11**(1), 13–24 (2005)
7. Drago, F., Myszkowski, K., Annen, T., Chiba, N.: Adaptive logarithmic mapping for displaying high contrast scenes. In: Proc. of EUROGRAPHICS'03, *Computer Graphics Forum*, vol. 22, pp. 419–426 (2003)
8. Dubois, P., Rodrigue, G.: High Speed Computer and Algorithm Organization, chap. An analysis of the recursive doubling algorithm, pp. 299–307. Academic Press (1977)
9. Durand, F., Dorsey, J.: Fast bilateral filtering for the display of high-dynamic-range images. In: Proc. of SIGGRAPH'02, pp. 257–266. ACM Press (2002)
10. Fattal, R., Lischinski, D., Werman, M.: Gradient domain high dynamic range compression. In: Proc. of SIGGRAPH'02, pp. 249–256. ACM Press (2002)
11. Ferwerda, J.A., Pattanaik, S.N., Shirley, P., Greenberg, D.P.: A model of visual adaptation for realistic image synthesis. In: Proc. of SIGGRAPH'96, pp. 249–258. ACM Press (1996)
12. Goodnight, N., Wang, R., Woolley, C., Humphreys, G.: Interactive time-dependent tone mapping using programmable graphics hardware. In: Rendering Techniques, pp. 26–37 (2003)
13. Goodnight, N., Woolley, C., Lewin, G., Luebke, D., Humphreys, G.: A multigrid solver for boundary value problems using programmable graphics hardware. In: HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 102–111 (2003)
14. Hensley, J., Scheuermann, T., Coombe, G., Singh, M., Lastra, A.: Fast summed-area table generation and its applications. Computer Graphics Forum **24**(3), 547–555 (2005)
15. Hensley, J., Scheuermann, T., Singh, M., Lastra, A.: Fast HDR image-based lighting using summed-area tables. Tech. Rep. TR06-017, University of North Carolina at Chapel Hill (2006). URL ftp://ftp.cs.unc.edu/pub/publications/techreports/06-017.pdf
16. Krawczyk, G., Myszkowski, K., Seidel, H.P.: Perceptual effects in real-time tone mapping. In: Spring Conference on Computer Graphics, pp. 195–202. ACM (2005)
17. Larson, G.W., Rushmeier, H.E., Piatko, C.D.: A visibility matching tone reproduction operator for high dynamic range scenes. IEEE Trans. Vis. Comput. Graph. **3**(4), 291–306 (1997)
18. Pattanaik, S., Yee, H.: Adaptive gain control for high dynamic range image display. In: Spring Conference on Computer Graphics, pp. 24–27. ACM (2002)
19. Pattanaik, S.N., Ferwerda, J.A., Fairchild, M.D., Greenberg, D.P.: A multiscale model of adaptation and spatial vision for realistic image display. In: Proc. of SIGGRAPH'98, pp. 287–298. ACM Press (1998)
20. Reinhard, E.: Parameter estimation for photographic tone reproduction. J. Graph. Tools **7**(1), 45–52 (2002)
21. Reinhard, E., Stark, M., Shirley, P., Ferwerda, J.A.: Photographic tone reproduction for digital images. In: Proc. of SIGGRAPH'02, pp. 267–276. ACM Press (2002)
22. Reinhard, E., Ward, G., Pattanaik, S., Debevec, P.: High dynamic range imaging: acquisition, display, and image-based lighting. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2006)
23. Schlick, C.: Quantization techniques for the visualization of high dynamic range pictures. In: Proc. of EUROGRAPHICS Workshop on Rendering (EGRW'94), pp. 7–20 (1994)
24. Seetzen, H., Heidrich, W., Stuerzlinger, W., Ward, G., Whitehead, L., Trentacoste, M., Ghosh, A., Vorozcovs, A.: High dynamic range display systems. In: Proc. of SIGGRAPH'04, pp. 760–768. ACM Press (2004)
25. Seetzen, H., Whitehead, L.A., Ward, G.: A high dynamic range display using low and high resolution modulators. The Society for Information Display International Symposium **34**(1), 1450–1453 (2003)
26. Tumblin, J., Rushmeier, H.: Tone reproduction for realistic images. IEEE Computer Graphics and Applications **13**(6), 42–48 (1993)
27. Ward, G.: Graphics gems IV, chap. A contrast-based scalefactor for luminance display, pp. 415–421. Academic Press Graphics Gems Series. Academic Press Professional, Inc. (1994)
28. Yee, H., Pattanaik, S.: Segmentation and adaptive assimilation for detail-preserving display of high-dynamic range images. The Visual Computer **19**(7-8), 457–466 (2003)
29. Zhang, X., Wandell, B.: A spatial extension of CIELAB for digital color image reproduction. SID Journal **5**(1), 61–63 (1997)