Graphics Hardware 2007
Timo Aila and Mark Segal (Editors)

A Real-Time FPGA-Based Architecture for a Reinhard-like Tone Mapping Operator

F. Hassan and J. E. Carletta

Department of Electrical and Computer Engineering, The University of Akron, USA

Abstract

This paper presents a field-programmable gate array-based hardware architecture for a Reinhard-like tone mapping operator. Modifications to the original Reinhard operator were done to ensure that the operator is amenable to implementation in hardware. The architecture is described in VHDL and has been synthesized using Altera Quartus tools. It achieves an operating frequency consistent with a video rate of 60 frames per second for a frame of 1024×768 pixels. The quality of the implementation is measured using peak signal-to-noise ratios on testbench images.

Categories and Subject Descriptors (according to ACM CCS): I.4.10 [Computing Methodologies]: Image Processing and Computer Vision—Image Representation; I.3.1 [Computing Methodologies]: Computer Graphics—Hardware Architecture.

1. Introduction

Novel camera sensors can capture scenes of much wider dynamic range than standard display devices can display. Tone mapping operators are used to bridge this mismatch. It is highly desirable to be able to do this processing in real-time, and embedded within the camera or display. However, published tone mapping operators have been implemented mostly on workstations and applied to stored images, without regard to resource and timing constraints. The work described in this paper modifies a well-known local tone mapping operator so that it can be implemented in real time on an embedded system with high throughput and low cost. Various platforms can be considered for implementation, including general purpose microprocessors, digital signal processors, field programmable gate arrays (FPGAs), and graphics cards. Of these platforms, FPGAs are the most compelling for embedded, high throughput image processing systems. Commercially available FPGA devices permit low cost implementations that are easily updated, much more easily embedded than those using graphics cards, and significantly faster than those based on processors. Despite the importance of local tone mapping operators, there has been no FPGA implementation of a state-of-the-art system for local tone mapping of images of high dynamic range.

The remainder of this paper is organized as follows. In Section 2 we review published tone mapping operators. We then give some background on the Reinhard operator that will be implemented in this paper in Section 3. Our proposed architecture for a Reinhard-like operator will be described in detail in Section 4. Details about the hardware synthesis of the system are given in Section 5. In Section 6 we show the simulation results of the system using a set of images from the Debevec library [Debevec and Malik 1997]. Finally, we give conclusions in Section 7.

2. Review of tone mapping operators

Tone mapping operators can be classified according to whether their operators are global or local. Common global tone mapping operators include those in [Tumblin and Rushmeier 1993; Ward 1994; Larson et al. 1997; Durand and Dorsey 2000; Pattanaik et al. 2000; Drago et al. 2003]. Because global tone mapping operators do the same transformation on every pixel, they lend themselves well to parallel implementations based on lookup tables. However, it is not possible for a single transformation based on global parameters to properly eliminate the effects of illumination in images for which illumination varies locally. [Chui et al. 1993] were the first to show that global tone mapping operators are not always sufficient.

In contrast, local tone mapping operators use transforms that vary locally with the neighborhood of the pixel. [Jobson et al. 1997a; Jobson et al. 1997b] normalized each pixel with an estimate of illumination surrounding it. They suggested the use of the mean of three different scales of the Gaussian surrounds of the pixel as an estimate of illumination. The design and development of a digital signal processor implementation of this method using one Gaussian surround was presented in [Hines et al. 2004], and achieved a frame rate of 20 frames per second for a 256×256 grayscale image. [Reinhard et al. 2002] select the best illumination estimate around the pixel from a Gaussian pyramid of the image. The authors in [Goodnight et al. 2003] mapped the Reinhard operator to the pixel processor on a programmable graphics hardware board. They were able to process 256×256 gray scale images at a frame rate of 30 frames per second. Another real-time implementation of the Reinhard operator was suggested in [Krawczyk et al. 2005]. The performance of the implementation was measured on a Pentium4 2 GHz processor and NVIDIA GeForce 6800GT graphics card. Using four scales of Gaussian pyramid; they were able to process only 14 frames per second for a 1024×768 image. Other local tone mapping operators include iterative methods [Tumblin and Turk 1999; Fattal et al. 2002], nonlinear filters [Durand and Dorsey 2002; Choudhury and Tumblin 2001; Ledda et al. 2004] and image appearance models [Pattanaik et al. 1998; Fairchild and Johson 2002; Johson and Fairchild 2003]. However, there is no published evidence of real time implementation of such methods.

High dynamic range devices were suggested in [Ledda et al. 2003; Seetsen at al. 2004] as an evaluation tool for tone mapping operators. [Ledda et al. 2005] validated six frequently used global and local tone mapping operators against linearly mapped high dynamic range scenes displayed on a novel high dynamic range device. The Reinhard operator performed best for gray scale images. This result has prompted us to investigate the Reinhard operator.

3. The Reinhard local tone mapping operator

The Reinhard local tone mapping operator normalizes each pixel of an image according to the average luminance surrounding the pixel. The difficulty is in determining how large a surrounding area to use in calculating the average luminance; if too small a surround is used, it does not give an accurate estimate of the local illumination, but if too large a surround is used, the surround may encompass a change in brightness that will throw off the estimate. Halo effects are caused when too large a surround encompasses a sudden change in brightness levels, causing a pixel to be normalized incorrectly. Reinhard's solution is to consider successively larger surrounds. A smaller surround is eliminated from consideration if the relative difference between its estimate and the estimate of the next larger surround is no more than a set threshold. When calculating the illumination estimate for a surround, the Reinhard operator weights the pixels in a surround according to the Gaussian function:

$$g(i,j,s) = \frac{1}{\pi s^2} e^{-\frac{i^2+j^2}{s^2}} \qquad (1)$$

where i and j are the indices of the pixel, and s is the scale of the surround; this scale controls the rate of decay of the Gaussian around the pixel being processed. The pixel being processed has indices $(i, j) = (0, 0)$. The Reinhard operator uses nine surrounds; the first has $s = 0.35$, and each successive scale is 1.6 times larger than the previous one. When the estimate is calculated in the space domain via convolution, a window size must be chosen for each scale that is large enough to include the main area under the Gaussian. The first Gaussian lies in a 3×3 window, and the Gaussians get progressively wider, with the ninth in a 60×60 window. Together, the nine windows form a Gaussian pyramid.

4. Proposed architecture

The block diagram of the proposed architecture of the Reinhard-like operator is shown in Figure 1. The input to the system is a pixel stream for a video frame. A Gaussian pyramid is used to estimate the illumination local to the pixel. A logarithmic average is also computed for the global image to provide an estimate of the global illumination. Finally, the input pixel is normalized by a weighted sum of the local and global illumination estimates. We developed hardware for each subsystem that approximates the original operator well, but is simple enough for real-time embedded processing. We next consider hardware-friendly modifications to the Gaussian pyramid, log average and normalization blocks of the Reinhard operator.

4.1. Approximating the Gaussian pyramid

The original Gaussian filters are computationally complex; nine windows are used, and the required filter coefficients, with their exponential relationship, are in general difficult to implement in fixed-point mathematics. We use fewer scales, and replace the coefficients with ones that serve the same purpose but are much more hardware-friendly. We use four square windows of size 8×8, 14×14, 28×28 and 56×56 pixels, respectively. Figure 2 shows the 3D plot of these windows; the values shown in the plot are proportional to the logarithm of the corresponding weight in the window. Expressions for the pixel weights in the lower left quadrant are given, where $\lceil\ \rceil$ denotes the ceiling function; the other three quadrants can be obtained via symmetry.

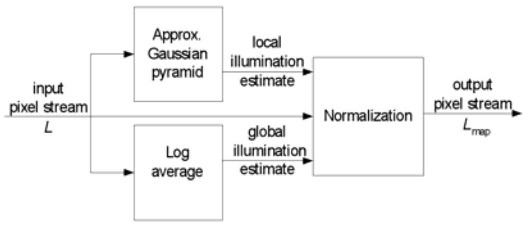


Figure 1: *Block diagram of the proposed architecture.*

As in a Gaussian filter, larger weights are given to the pixels in the center of the window; however, we approximate the steep exponential drop-offs of the two smaller windows by implementing filters with coefficients that are with powers-of-four and powers-of-two, respectively; the less steep drop-offs of the larger windows are approximated by using filter coefficients that are powers-of-two repeated two and four times, respectively. This special relationship of the weights we have chosen ensures that computation of the averages can be done with simple accumulator structures.

We demonstrate the idea using a one-dimensional window of length fourteen and even symmetry. The one-dimensional window is given by the weights:

$$w[k] = \begin{cases} \frac{1}{2^6} 2^{k-1} & \text{for} \quad k = 1, \ldots, 7 \\ \frac{1}{2^6} 2^{14-k} & \text{for} \quad k = 8, \ldots, 14. \end{cases} \tag{2}$$

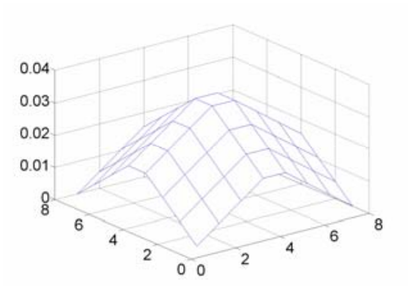
An output pixel $y[n]$ can be expressed in terms of the input pixel stream P as:

$$y[n] = \frac{\sum_{k=1}^{7} P[n-8+k] \times 2^{k-1} + \sum_{k=8}^{14} P[n-8+k] \times 2^{14-k}}{2^6} \tag{3}$$

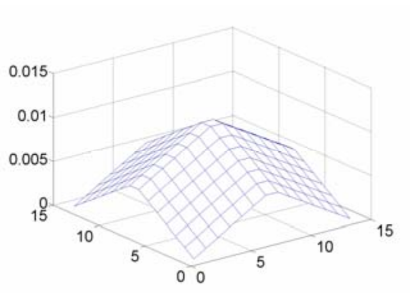
Note that in the first half of the window, each weight is twice the one immediately preceding it. This half of the window can be considered a rising geometric series with a gain of two. Similarly, the second half of the window is a falling geometric series, also with a gain of two, so that the overall gain of the window is four.

We can take advantage of the structure of the window in order to compute the window output using very simple accumulator-based hardware. One accumulator is devoted to the rising geometric series that is the first summation in (3); a second accumulator tracks the falling geometric series that is the second summation in the equation. The output of the window can then be found by simply adding the two accumulators together and dividing by the gain factor of four.

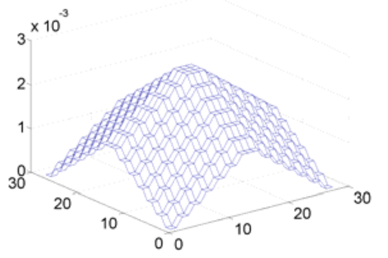
As the window slides one pixel further along the data, the accumulators are updated in the following simple way. For the first accumulator, the incoming pixel has the largest weight, of 2^6. All pixels already accounted for in the



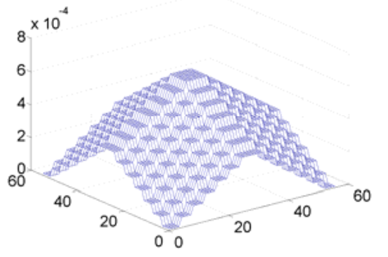
$$w_{i,j} = 2^{2i-8} \times 2^{2j-8}$$



$$w_{i,j} = 2^{i-7} \times 2^{j-7}$$



$$w_{i,j} = 2^{\left\lceil \frac{i}{2} \right\rceil - 7} \times 2^{\left\lceil \frac{j}{2} \right\rceil - 7}$$



$$w_{i,j} = 2^{\left\lceil \frac{i}{4} \right\rceil - 7} \times 2^{\left\lceil \frac{j}{4} \right\rceil - 7}$$

Figure 2: *The four windows for the four-scale approximate Gaussian pyramid.*

accumulator have a weight of one-half of whatever the weight was before sliding the window. Therefore, we can simply halve the entire accumulator, subtract the pixel on the extreme left hand side of the window weighted by 2^{-1}, and add the pixel in the middle of the window weighted by

2^6. The second accumulator is responsible for the falling geometric series. Here, the incoming pixel has the smallest weight, 2^0, and weights are doubled each time the window slides. Therefore, we can simply double the entire accumulator, add the newest pixel, and subtract the pixel in the middle of the window weighted by 2^7. While the accumulator updates described so far apply only to windows that lie completely within the borders of the image, the updates can be easily modified using stacks to accomplish symmetric extension at the image borders while still inputting each image pixel only once.

The two-dimensional square windows are implemented using two one-dimensional filters. The vertical filter calculates the average of the input image along the columns. The horizontal filter calculates the average of the image processed by the vertical filter along the rows. Because the pixels enter the system in a row-by-row order, it is not sufficient to use a single accumulator to calculate the output of the vertical filter. Instead, we use a set of accumulators, one for each column of the image. If, for example, the image contains 1024 pixels in a row, there is a set of 1024 accumulators, stored together in a single 1024-word memory. The accumulators in the set are updated one by one as the pixels enter the system.

The bottleneck for existing implementations of local tone mapping algorithms is memory access, so an appropriate memory organization for supplying data to the four differently sized windows is key to achieving real-time performance. Memory is also needed to supply a delayed input pixel stream, synchronized such that each input pixel appears simultaneously with its local illumination estimate. As shown in Figure 3, we divided the fifty-six rows of pixels needed for the update of the largest window into ten separate memories. Memory one holds fourteen rows of pixels and is used as a delay block as well as a stack. Memories two, three, four, and five hold seven, three, three, and one row of pixels respectively, and they are used as delay blocks. Memories six, seven, eight, nine and ten hold one, three, three, seven and fourteen rows respectively and they are used as delay blocks as well as stacks. All these memories are implemented using dual-port physical memories. The physical memories are sized up to hold a number of rows that is a power-of-two. One additional single-port physical memory of sixteen rows, not shown in the figure, holds a copy of the last rows of the image. The memory organization ensures that all pixels needed in a single time step are in different physical memories; these pixels include the four pixels at the extreme right sides of the four windows, the four pixels at the extreme left sides, the one pixel that lies in the (common) middle of the four windows, and the delayed version of the input pixel stream. Because they are in different physical memories, all pixels needed to update the accumulators in a given time step can be read simultaneously without conflict. FPGAs, which contain large numbers of moderately sized memory blocks, are an ideal platform for this sort of memory organization.

The local illumination estimate for a pixel $L_{\text{ave\_local}}$ is found by selecting the most suitable of the four windows. The algorithm used is similar to Reinhard's original selection criterion, and compares the relative difference between a window and the next larger window to a set threshold. The most computationally complex part of the algorithm is the division required for the relative difference. We do this division in a hardware-friendly way by converting the fixed-point value for the denominator into a floating-point value with a mantissa between 0.5 and 1. We then use an iteration based on the Newton-Raphson method to find the reciprocal, after retrieving an initial guess for the iteration from a look-up table that is indexed on a limited number of bits of the mantissa. The procedure is such that one look-up and one iteration are enough to achieve the required precision.

4.2. Approximating the log average luminance

The log average luminance subsystem for our real-time embedded version of the Reinhard operator calculates an estimate of the global luminance of the image. It does this by computing the sum of the base-2 logarithms of all the pixels in the image. The global illumination is estimated as 2 raised to this sum, and corresponds to the average of all the pixels. Figure 4 shows the block diagram of the log average luminance subsystem. It is divided into three main subblocks. The first takes the base-2 logarithm of the pixel stream, the second computes the average of the logarithms, and the third takes the inverse base-2 logarithm of the average.

An estimate of the base-2 logarithm of an integer x can be found from the number of leading zeros in the integer; the weight of the most significant '1' in x is the integer part of the base-2 logarithm, and the remaining bits in x determine the fraction part of the logarithm. For example, the integer 3481 has its most significant '1' in the eleventh position; its logarithm can be written as $\log_2(3481) = \log_2(2^{11}+1433) = 11+\log_2(1+1433/2^{11})$.

We can use the bits following the most significant one bit in a number of different ways to estimate the fraction part of the base-2 logarithm; we need to compute $\log_2(1+f)$, where f is a number between 0 and 1. One possibility suggested in [Hau et al. 2004] is to simply use the bits themselves directly as an approximation of the fraction part. We propose to use a fixed number of bits following the most significant '1' to look up the value of $\log2(1+f)$ in a table. The size of the lookup table and the error in the estimation depend on how many bits are used to address that table, and how many fraction bits are stored in the table. As an example, for an address size of eight and word size of eight, $\log_2(1+1433/2^{11}) = 0.761$, so that our estimate of $\log_2(3481)$ is 11.761. Hau's method gives 11.699. The actual value is 11.765. The proposed method gives a quantifiable trade-off between hardware cost and the percentage error at the output of the log average luminance subsystem.

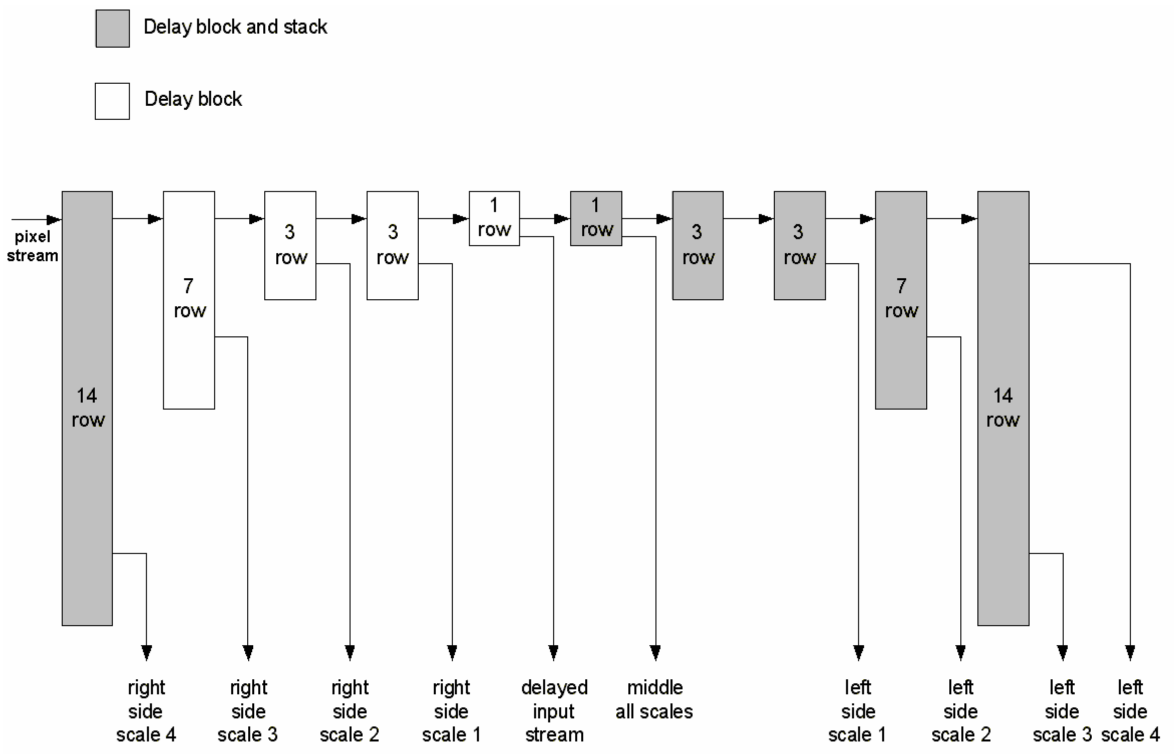


Figure 3: *Memory organization for a four-scale approximate Gaussian pyramid.*

The sum of the base-2 logarithms of all the pixels in the image is computed using an accumulator. The width of the accumulator depends on the resolution and dynamic range of the image. For example, a 1024×768 image with 13 bits of precision for the base-2 logarithm (five bits for the integer part and eight bits for the fraction part) requires an accumulator of 33 bits. The log average is found by dividing the accumulator by the number of pixels in the image, here N = 1024×768. We approximate the reciprocal of N with a four-term sum of powers-of-two as $2^{-20}\left(1+2^{-2}+2^{-4}+2^{-6}\right)$, and then accomplish the multiplication of the accumulator by the reciprocal with a series of shifts and additions. The result has thirteen bits of precision.

The final step of the log average luminance is to compute the inverse base-2 logarithm of the average. Writing the average as an integer part x and a fraction part f, we have $2^{(x+f)} = (2^x)(2^f)$ where 2^x is a power-of-two and 2^f is a number between 1 and 2, which we further break into 1+g. A look-up table, addressed with eight bits of f and holding a word size of eight bits, is used to determine g, and the '1' is added. The result is then multiplied by 2^x using a barrel shifter. The final output of the log average luminance is computed by truncating the fraction bits of the output of the barrel shifter. This log average luminance is the global estimate of illumination $L_{\text{ave\_global}}$ for the pixel.

4.3. Normalizing the pixel

The normalization block takes the local average around the pixel and the weighted log average as inputs to normalize the pixel stream. We add these two quantities to get the normalization value of every pixel. The output of the system is given by:

$$L_{\text{map}} = \frac{L}{L_{\text{ave\_local}} + a * L_{\text{ave\_global}}} \tag{4}$$

where a is a weighting factor, $L_{\text{ave\_local}}$ is the local estimate of illumination around the pixel, $L_{\text{ave\_global}}$ is the global illumination estimate and L_{map} is the mapped luminance represented in floating point. The division method previously described is used again here.

At this point, the output pixel stream is represented in floating-point with eight bits of mantissa and five bits of exponent. The final step of the system is to convert the output pixel stream to fixed-point. For display purposes, the normalized pixel stream shoud have an eight-bit gray level between 0 and 255. We scale L_{map} to this range by first adding eight to the output pixel exponent, which is equivalent to multiplying the output pixel stream by 256. Then, we send the mantissa of the output pixel stream to a barrel shifter controlled by the exponent to convert the pixel to fixed-point, and saturate the fixed-point output to 255.

Table 1. *Summary of hardware synthesis report.*

Device	Stratix II EP2S90F1020C3
Total bits of memory	2,952,960 / 4,520,448
Total logic cells	17553 / 72,768
Max operating freq.	83.83 MHz

Table 2. *Summary of PSNR study, with values in dB.*

	constant weight	our system
memorial	30.9	34.4
rosette	25.1	28.5
groveC	29.4	33.5
groveD	29.8	33.6
vinesunset	41.4	42.6

5. Hardware synthesis details

The proposed architecture was described in VHDL, and synthesized using Altera's Quartus II v5.0 toolset. It was placed and routed on a Stratix II FPGA device. The architecture was sized in order to accommodate high resolution images of high dynamic range with 1024×768 pixels and 28 bits per pixel. It should be noted that memory requirements would be less for lower resolution images or images of smaller dynamic range.

Table 1 summarizes the synthesis report from Quartus. The simplicity of hardware is reflected in the clock speed achieved, and in the low utilization of logic cells. The implementation has used a significant percentage of the available embedded memory. It is clear that processing algorithms for high resolution images, in general, require significant amounts of memory. If they are to be implemented on a single chip, a specialized FPGA device with extended memory capabilities is required.

A typical video frame has horizontal blanking periods of 64 pixels, and a vertical blanking period of 32 rows. Given that we would like to achieve a video frame rate of 60 frames per second, and that there are (1024+64)*(768+32) or 870,400 pixels in the frame when we include the blanking periods, we need to be able to process 60*870,400 = 52.24 megapixels per second. Our architecture, which has a maximum operating frequency of 83.83 MHz, can accommodate this by taking in one pixel per clock.

6. Experiments and results

To test the visual quality of the system we used a set of testbench high dynamic range images from the Debevec library. We obtain gray scale images by calculating a luminance value for each pixel as: $P= 0.27R+0.67G+0.06B$. We then transformed the images into a fixed-point representation with sixteen bits of fraction and twelve bits of integer, and simulated the images being processed by our proposed architecture of the Reinhard-like operator. The system is verified by comparing a fixed-point Matlab simulation with a simulation of the behavioral VHDL using Modelsim software. The only input parameter to the system is the weight of the global average a which affects the brightness of the output image. We used a fixed a=2 for all images except for Vinesunset where a=0.5. Figure 5 shows the set of images after processing by our system. Evaluating the output images visually, we see that our system gives comparable results to the original method. In particular, we do not see halo artifacts; for this set of images, these would manifest themselves as black or bright bands around the church windows and behind the trees in the natural images. Details in the dark areas can be seen, and edges look sharp. We have also conducted a study of the peak-signal-to-noise ratio (PSNR) contained by our hardware friendly approximation to the Gaussian pyramid. Our gold standard was a floating point version of the Reinhard-like operator that uses Gaussian surrounds with standard deviations of 2, 3.5, 7 and 14, respectively. Considering the processed image from the gold reference to be the signal, and the difference between the processed images from the gold reference and the proposed architecture to be the noise, the PSNR are given in Table 2. The size of these values gives us confidence that the approximation is reasonable. We conducted the same PSNR study on a similar architecture that uses constant weight pyramid rather than our hardware approximation of the Gaussian pyramid. The PSNR values for that architecture were on average 3dB lower, that is, using constant weight windows results in twice as much error as our approximation.

7. Conclusions and future work

The proposed architecture, with its simplicity and high operating frequency, is a promising method for real-time display of high dynamic range (HDR) images on standard LCD screens. While the example system implemented here is for 1024×768 gray scale images with 28 bits per pixel, the design can be easily parameterized to deal with images of different input dynamic ranges and displays of different resolutions. Our future goal is to display a HDR image with 32 bits per pixel on a traditional LCD at a resolution of 1280x1024 pixels and a rate of 60 frames per second. The current operating frequency is not quite fast enough to reach this target. We are also considering extension to color images.

Inclusion of more scales would improve the quality of the processed images; the proposed architecture has four scales, while the original Reinhard operator has nine. Implementation of additional scales is not a simple extension, because it will no longer be adequate to use only geometric series based on powers-of-two. Still, we believe that new bases can be chosen that will translate into simple, fast hardware.

References

BURT, P.J., AND ADELESON, E.H. 1983. A multiresolution spline with applications to image mosaics. ACM Transactions on Graphics 2, 217-236.

CHOUDHURY, P., AND TUMBLIN, J. 2003. The trilateral filter for high contrast images and meshes. Proceedings of the Eurographics Symposium on Rendering, 186-96.

CHUI, K., HERF, M., SHIRLEY, P., SWAMY, S., WANG, C. AND ZIMMERMAN, K. 1993. Spatially nonuniform scaling functions for high contrast images. Proceedings of Graphics Interface, 245-253.

DEBEVEC, P. E., AND MALIK, J. 1997. Recovering high dynamic range radiance maps from photographs. Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, 369-378.

DRAGO, F., MYSZKOWSKI, K., ANNEN, T., AND CHIBA, N. 2003. Adaptive logarithmic mapping for displaying high contrast scenes. Computer Graphics Forum, 24th Annual Conference of the Eurographics Association,419-426,

DURAND, F., AND DORSEY, J. 2000. Interactive tone mapping. Proceedings of the Eurographics Workshop on Rendering Techniques, 219-232.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high dynamic range images. Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, 257-266.

FAIRCHILD, M.D., AND JOHNSON, G.M. 2002. Meet iCAM: An image color appearance model. IS&T/SID 8th Color Imaging Conference, 33-38.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, 249-56.

GOODNIGHT, N., WANG, R., WOOLLEY, C., AND HUMPHREYS, G. 2003. Interactive time-dependent tone mapping using programmable graphics hardware. ACM International Conference Proceeding Series: 14th Eurographics Workshop on Rendering, 26-37.

HAU, T.N., LI, .T, AND VIJAYAN, K.A., 2004. A nonlinear technique for enhancement of color images: an architectural perspective for real-time applications. Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop, 124-129.

HINES, G.D., RAHMAN, Z., JOBSON, D.J., AND WOODELL, G.A. 2004. DSP implementation of the Retinex image enhancement algorithm. Proceedings of the SPIE Visual Information Processing XIII, 13-24.

JOBSON, D.J., RAHMAN, Z., AND WOODELL, G.A. 1997. Properties and performance of a center/surround Retinex. IEEE Transactions on Image Processing 6, 451-462.

JOBSON, D.J. RAHMAN, Z., AND WOODELL, G.A. 1997. A multiscale Retinex for bridging the gap between color images and the human observation of scenes. IEEE Transactions on Image Processing 6, 965-76.

JOHNSON, G.M., AND FAIRCHILD, M.D. 2003. Rendering HDR images. Proceedings of IS&T/SID 11th Color Imaging Conference, 36-41.

KRAWCZYK, G., MYSZKOWSKI, K., AND SEIDEL, H-P. 2005. Perceptual effects in real-time tone mapping. Proceedings of the 21st Spring Conference on Computer Graphics, 195-202.

LARSON, G.W., 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, 291-306.

LEDDA, P., CHALMERS, A., TROSCIANKO, T., AND SEETZEN, H. 2005. Evaluation of tone mapping operators using a high dynamic range display. ACM Transactions on Graphics, 24, 640-648.

LEDDA, P., SANTOS, L.P., AND CHALMERS, A. 2004. A local model of eye adaptation for high dynamic range images. Proceedings of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualization and Interaction in Africa, 151-160.

LEDDA, P., WARD, G., AND CHALMERS, A. 2003. A wide field, high dynamic range, stereographic viewer. Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, 237 – 244.

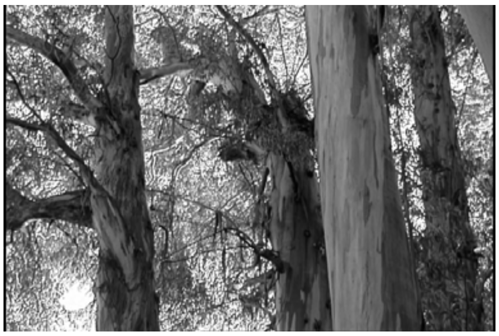
PATTANAIK, S.N., TUMBLIN, J., YEE, H., AND GREENBERG, D.P. 2000. Time-dependent visual adaptation for fast realistic image display. Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, 47-54.

PATTANAIK, S.N., FERWERDA, J.A., FAIRCHILD, M.D., AND GREENBERG, D.P. 1998. A multiscale model of adaptation and spatial vision for realistic image display. Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, 287-98.

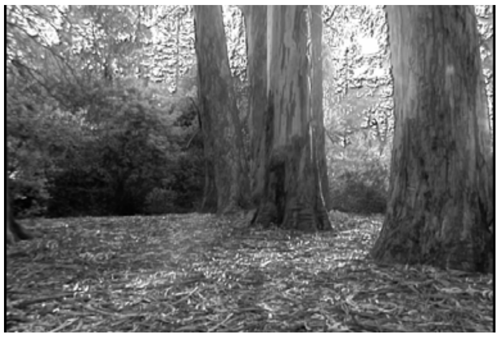
WARD, G. 1994. A contrast-based scalefactor for luminance Display. Graphics Gems IV, Academic Press, 415-21.

Memorial

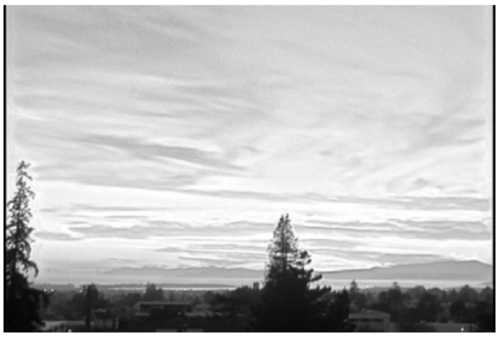
Rosette



groveD



groveC



Vinesunset

Figure 5. The set of images processed by our proposed architecture.