Tone Mapping in VR Environments

A. Hausner and M. Stamminger

Computer Graphics Group, University of Erlangen-Nuremberg, Germany

Abstract

Tone mapping is the process of mapping high dynamic range images to the low dynamic range of current displays. So far, tone mapping research was focused on static images, but with powerful graphics hardware available today, real time tone mapping becomes possible. In this paper we describe tone mapping within a VR environment. The tone mapping operator utilizes information about the user's line of sight obtained by a tracking system. Adaptation is optimized for the current view point and costly image operations are restricted to the central view field. Since all computations are completely performed by graphics hardware, we achieve interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques

1. Introduction and Previous Work

Recent improvements in graphics hardware allow to shift more and more complex operations to the graphics processor. The fixed pipeline stages are replaced by programmable shader units, so that general computations can be done on the graphics card. In latest graphics hardware all these units and data paths can work with floating-point precision which allows us to deal with high dynamic range data (HDR) [DM97].

Current display devices can only display a low dynamic range of intensities, whereas real world environments typically exhibit a much higher dynamic range. *Tone mappers* map light intensities of high dynamic range to the displayable intensities, with the goal to maintain as much contrast and detail as possible. There are many different tone mapping operators available so far. Generally, they can be classified as global and local operators. Global operators gather information from all pixels of the image and apply a single mapping function to all pixels. Due to their global nature, these operators cannot always preserve local contrast. Local operators operate on a per-pixel basis and apply a locally varying mapping. For a detailed discussion please refer to [KDP02], which give an extensive survey of available tone mapping operators.

The problem of tone mapping also appears in interactive rendering, and in particular also in virtual reality (VR) environments. In this paper we describe a variant of Reinhard's tone mapping operator [ERF02], which is tailored for VR environments. Usually, in a VR environment, the position of the observer is followed by a tracking system. The knowledge of the viewer's position is used to adapt the virtual camera, so that the user can move around virtual objects just by moving the head.

In this paper, we use this additional information from the tracking system to optimize the tone mapper. Since we know the user's head position and view direction, we can determine at which part of the screen the user looks. Because in VR environments we usually have large screens, the focused regions only covers part of the screen. We then can concentrate the tone mapping computations to this focused region, which saves time and optimizes the result for the user's current line of sight.

We have choosen as basis Reinhard's operator for several reasons. It is an excellent operator developed from a practical approach producing very good results. The operator combines a simple global function and a complex local function and is therefore well suited for a GPU implementation. Goodnight at al. [NGH03] have shown, that this operator can be implemented on graphics hardware for the rendering of high dynamic range images.

2. Review of Reinhard's Operator

The idea of Reinhard's operator is based on practical photography where photographers use the zone system to determine

© The Eurographics Association 2004.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

whether all details they capture are preserved on the final print. To avoid loss of detail in local regions, Reinhard extends his *simple operator* with a technique that is quite similar to the printing technique dodging-and-burning [Ada83].

Given is the world luminance $L_w(x,y)$ of each pixel (x,y) in the scene. First, the log-average luminance $\overline{L}_w$ is computed:

$$\overline{L}_w = \exp\left(\frac{1}{N}\sum \log(\delta + L_w(x,y))\right) \tag{1}$$

where N is the number of pixels in the image and δ a small safety parameter for black pixels. $\overline{L}_w$ is then used to scale the luminance L_w to the middle grey zone:

$$L(x,y) = \frac{a}{\overline{L}_w} L_w(x,y) \tag{2}$$

where $L(x,y)$ is the scaled luminance and a is the key-value to indicate whether an image is subjectively light ("high key") or dark ("low key"). The standard value is 0.18 for normal-key scenes, but the user can vary a down to 0.09 and 0.045 or up to 0.36, 0.72 or 1.0 if the scene makes it necessary.

Next, a simple global operator is applied to obtain the display luminance:

$$L_d(x,y) = \frac{L(x,y)}{1 + L(x,y)} \tag{3}$$

brings all luminances to a displayable range of $[0:1]$.

This simple operator produces good results, but in bright regions detail is lost. To counteract this effect, Reinhard introduces *automatic dodging-and-burning*, a technique that has quite the same effect as the same-named printing technique: On some portions of the image more light is added, on others light is kept back. Dodging-and-burning is applied to regions where no sudden contrast changes occur.

The following center-surround function gives a measurement of local contrast changes at a certain scale. This function is defined by

$$CS(x,y,s_i) = \frac{V(x,y,s_i) - V(x,y,s_{i+1})}{2^{\phi} a/s_i^2 + V(x,y,s_i)} \tag{4}$$

where V_i is a gaussian convolution at a certain scale s_i:

$$V(x,y,s_i) = L(x,y) \otimes R(x,y,s_i) \tag{5}$$

$$R(x,y,s_i) = \frac{1}{\pi s_i^2} \exp\left(-\frac{x^2+y^2}{s_i^2}\right) \tag{6}$$

Now for every pixel, the size of the surrounding region without significant contrast changes is computed. To this end, blurred images with increasing filter size are generated, starting with $s_0 = 0.35$ and $s_{i+1} = 1.6 \times s_i$. For every pixel (x,y), we store the last filter size $s_{loc}(x,y)$, for which the local contrast is below a threshold, i.e. $|CS(x,y,s_i)| < \varepsilon$, $\varepsilon = 0.05$. ϕ is a sharpening parameter that enhances edges and is set to 8 by default. Reinhard observed best results with these values and our tests approved this. By increasing the sharpening parameter to higher values than eight we saw no visual difference. Decreasing ϕ dampens the effect of local sharpening.

Finally, the tone mapping operator of Equ. 3 is applied, but the scaled luminance in the denominator is replaced by $V(x,y,s_{loc}(x,y))$:

$$L_d(x,y) = \frac{L(x,y)}{1 + V(x,y,s_{loc}(x,y))} \tag{7}$$

which serves as a local dodging-and-burning operator for each pixel in the image.

3. A Tone Mapping Operator for VR

Our operator requires an HDR texture as input, so we first render the scene offscreen to a floating point P-Buffer. During rendering, we immediately compute the logarithmic luminance $\log L = \log(0.27R + 0.67G + 0.06B)$ of all pixels by a fragment program and also store these values in the result image.

The following computations only work on these luminance values, so we only need one-channel P-Buffers. With nVidia's extensions, a floating point buffer with one channel is only available for the R-Channel, so we have to store the luminance in the R-channel and swizzle the original RGB to GBA.

3.1. Global Scaling

We first compute the log-average luminance from Equ. 1. The log luminance values have been computed during rendering, so they only need to be averaged. Computing the average value of an image can be done by sequentially downsampling the image with a fragment program to an image size of one pixel. The same effect can be achieved faster using automatic mipmap generation of modern graphics hardware.

Although automatic mipmapping – introduced by the extension GL_GENERATE_MIPMAP_SGIS – is performed by the hardware it is still a time-consuming computation. For this reason we use a smaller P-Buffer of size 128^2 in this step.

Because mipmapping is not supported for floating point textures we scale the log luminance down to the range $[0:1]$ and use a standard pbuffer with 8 bits precision, which is sufficient for our purposes. After mipmapping we read back the single pixel of the coarsest mipmap level into system memory and perform another render pass that scales the luminance according to Equ. 2.

3.2. Local Scaling

In a VR environment usually head position *and* orientation are tracked. With this information we can determine the

© The Eurographics Association 2004.

screen position of the viewer's line of sight. We assume that the viewer's attention is concentrated to a region around that view point, and we call this region the *focus region*. Because the screens in typical VR environments are relatively large (with a field of view of 90 degrees or more), the focus region only covers a fraction of the original image.

Because the viewer mainly perceives the luminances in the focus region, we compute a local average luminance value for the focus region. From this we can obtain a local scaling factor as in Equ. 2. Instead of scaling the entire image with a single global factor, we scale the image with the local scaling factor in the focus region, and the global scaling factor outside the focus region with a smooth transition zone.

By this, we achieve a better representation of the relevant focus region. When the user looks into a dark (bright) region, this region is brightened (darkened) and detail becomes visible again.

3.3. Local Sharpening

We also apply the local sharpening operator of Reinhard. For the computation of Equ. 7, we must determine $s_{loc}(x,y)$ for every pixel and store the corresponding color value $V(x,y,s_{loc}(x,y))$ in a *local luminance image*.

We initialize this image with $L(x,y)$. Then, we generate filtered versions of the input image with increasing filter size s_i. The computation of the gaussian convolution is separated to a horizontal and vertical filter pass. This separation requires two operations with a 1D-filter, but is still much faster than a single 2D-filter operation.

Whenever the local contrast for a pixel (x,y) grows above a threshold, i.e. $|CS(x,y,s_i)| > \varepsilon$, we store the last available filtered luminance value $V(x,y,s_{i-1})$ in the local luminance image. Thus, each iteration requires three rendering passes, two for the filtering and one for the threshold test.

The local sharpening is very time consuming and too slow for interactive applications. To tune this operation we make use of the knowledge where the viewer is looking at. We concentrate the local sharpening computation to the focus region by building a pyramid: In the first two iterations we compute the local sharpening only in a quad of size 512^2, the third and fourth iteration in a quad of 256^2 and for every following iteration a quad with size 128^2, where each quad is centered around the focus regions.

With this approach a viewer sees his focus region with maximum detail. On current graphics boards, this optimization results in interactive frame rates at the cost of some details which are outside the viewer's viewing angle.

3.4. Stereo Rendering

Stereo viewing requires rendering the scene twice which halves the frame rate. However, in practice we can reuse the log-average luminance of Equ. 1 computed for one eye. Thus we avoid each mipmapping operation for global and local scaling. Our experiments have proven that the log-average luminance for the left and right eye differ only marginal.

4. Results

In our tests we were using an nVidia FX-series graphics card, that supports a standard floating point precision of 32 bits per channel. All benchmarks were done using an AMD Athlon XP 3200 with 512 MB RAM and a GeForce FX 5900 card. We have implemented both a Linux and Windows version of the application in OpenGL. The pixel buffers the tonemapper uses had a size of 1024^2.

fps	Windows	Linux
Global Scaling	37	22
Local Scaling	25	20
Local Sharpening	25	17
Local Scaling / Local Sharpening	19	15

Table 1: *Achieved frames rates rendering Paul Debevec's memorial church (the image was resized to 1024^2). The windows version profits from its render-to-texture capability.*

Table 4 shows our benchmark results rendering Paul Debevec's memorial church in a resolution of 1024^2. The Windows version profits from its render-to-texture support and is much faster than the Linux version which requires glCopyTexSubImage() instructions. Local sharpening is done in up to 7 iterations. We have found that further iterations do not increase the image quality much more, and are not worth the additional computation time. Doing even more iterations requires a gaussian filter with radius of at least 15 pixels (9th iteration). That means 31 texture lookups or 8 render passes (each 4 horizontally and vertically respectively).

Fig. 1 shows the tone mapping results of Paul Debevec's memorial church. Depending on the line of sight, details in the focus region become visible (left and center image). Local sharpening increases the contrast further (right image).

Screenshots of an interactive walkthrough session of a temple scene are shown in Fig. 2. If the viewer looks inside the temple, the local scaling and sharpening make details visible.

5. Conclusion

In this paper we have shown how Reinhard's tone mapping operator can be integrated to a VR environment. Our operator uses knowledge about the user's line of sight to optimize the result and the computation time.

Recent announcements of nVidia's GeForce FX 6800 card say that this card is now capable of computing all graphics

© The Eurographics Association 2004.

with local scaling and local sharpening | with local scaling | with local scaling and local sharpening

Figure 1: *An HDR image displayed with our tone mapper. The user's line of sight is marked by a red dot. Note how the tone mapper brightens the dark region in the left image and dims the cupola in the center image. In the right image the contrast in the cupola is further increased by local sharpening.*

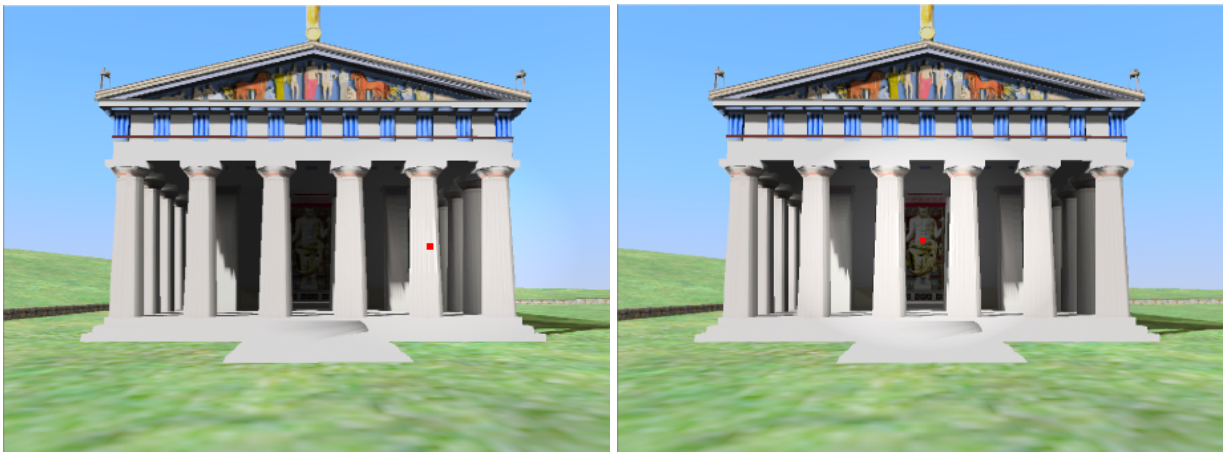

Figure 2: *Snapshots of a walkthrough of a 3D temple scene with line of sight marked in red. If the viewer looks inside the temple, details in the temple become visible (scene provided by "Foundation of the Hellenic World").*

operations with floating point precision and no clamping to [0:1] between the pipeline stages. We can then render 3D scenes with features like blending etc.

A possible improvements of the operator would be to implement a time-dependent model. With our operator so far it isn't possible to have a glare effect when looking suddenly from very dark regions to very bright regions. Our operator scales the luminance down at once. An adaption model that scales down the luminance down by time could mimic the behaviour of the human eyes.

References

[Ada83] ADAMS A.: *The Print*. Little, Brown and Company, 1983. 2

[DM97] DEBEVEC P. E., MALIK J.: Recovering high dynamic range radiance maps from photographs. *Computer Graphics Proceedings, Annual Conference Series* (Aug. 1997), pages 369–378. (Proceedings of SIGGRAPH 1997). 1

[ERF02] ERIK REINHARD MICHAEL STARK P. S., FERWERDA J.: Photographic tone reproduction for digital images. *ACM Transactions on Graphics 21*, 3 (July 2002), C219–C231. (Proceedings of SIGGRAPH 2002). 1

[KDP02] KATE DEVLIN ALAN CHAMBERS A. W., PURGATHOFER W.: Star: Tone reproduction and physically based spectral rendering. *Proceedings of Eurographics* (Sept. 2002), pages 101–123. 1

[NGH03] NOLAN GOODNIGHT RUI WANG C. W., HUMPHREYS G.: Interactive time-dependent tone mapping using programmable graphics hardware. *Eurographics Symposium on Rendering* (Sept. 2003), pages 1–13. 1

© The Eurographics Association 2004.