186.119 Projektpraktikum mit Bakkalaureatsarbeit (2006/07)
Ralf Habel (Tutor)

Realtime HDR Rendering

Christian Luksch

MatNr. 0525392

Institute of Computer Graphics and Algorithms
TU Vienna

Abstract

High dynamic range rendering in realtime graphics has increased the visual quality of realtime scenes essentially. There are several techniques of HDR-Rendering. This work summarizes the theoretical background and results of previous works. A ready-to-use HDR rendering library has been developed that integrates seamlessly into the open source graphics engine Ogre3D [Ogr]. Further a comparison that show the advantages and disadvantages of different techniques in a realtime rendered scene has been made.

1. Introduction

HDR-Rendering follows another way to create images. The algorithms tries to imitate the physical behavior of light and human vision. Color and luminance values are calculated in high dynamic range, approaching to real-world luminance values, measured in Candela per square meter. Instead of 8-bit color depth the whole calculation has to be done in floating-point values. The dynamic range is the ratio of the highest and lowest value of these data. In natural environments this range is very high and to visualise these range, often a logarithmic scale is used. The term HDR is not clearly defined, but speaking of HDR usually means to cover more than about 4 orders of magnitude.
HDR images cannot be displayed directly on normal display hardware. To display these data a tone mapping operator maps them back to low dynamic range, also called device color space. Most techniques use a logarithmic based mapping to simulate human vision. The goal is to find a display representation which yields the same visual sensation as viewing a scene in the real world.
In the library we handle global tone mapping operators due to realtime rendering constraints. A global tone mapping operator determinates a mapping function for all pixels, e.g. it takes the average luminance of the scene to scale all values. Another approach is a local tone mapping operator which calculates a mapping value for each pixel from the surrounding pixels. Grzegorz [KMS05] shows a realtime implementation of this approach mainly based on the theoretical background of Erik Reinhard [RSSF02].
To improve the realism one can use techniques which imitates the behaviour of human perception, e.g. a temporal adaptation can be used to simulate the reaction of human vision in changing luminance. Also a glare and star effect that imitates the recording of light in photographically equipment can be added. The whole process of HDR-Rendering is computationally expensive. To handle this in realtime at least a Shader 2.0 graphics device that supports floating-point render targets is required.
The major part of this work was to implement a library and to test different tone mappers and HDR rendering techniques. It is currently part of the Ogre3D Add-ons library collection [Ogr]. It can be used in any Ogre3D application with minor exceptions. The library can be controlled by several parameters to configure the processing from the main application. Internal it uses the Ogre3D compositor framework and creates a post-processing effect, which handles all the HDR processing and adds it to the compositor chain. The HDR effect can also be combined with other post-processing effects e.g. Motion-Blur, Depth-Of-View, ...

2. Human perception

First we take a closer look at human perception to understand the processes on witch the implemented models are based on.
Humans are capable of seeing a huge range of intensities,

from daylight level to night luminances. The cells of the Human Visual System have a much more limited response range, about 4 orders of magnitude. They cope the large luminance range by a process called adaptation. A raw adaptation takes place in the first seconds, but the final adaptation takes several minutes. At this an additional distinction has to be made by either adopting to dark or to light intensities and it is also depending on the actual level of illumination. [FPSG96a]

Human perception is not equal at all light intensities, because of the different sensitivities of the two different kinds of photoreceptors, the rods and cones.

Rods are responsible for perception of luminance differences, but have no capability to distinguish different colors. They are sensitive to low luminance values, in higher orders they lose their capability. The range where predominantly rods are responsible for vision is called scotopic vision [FPSG96b].

The second kind of photoreceptors are cones, which are responsible for color vision. In detail there are three different types, each sensitive at a certain wavelength of light, approximately red, green and blue. They are concentrated in the center of the retina called fovea and only work in the well lighted luminance range. The range where cones are responsible for vision is called photopic vision. The overlapping region between the scotopic and photopic range is called mesopic [FPSG96b].

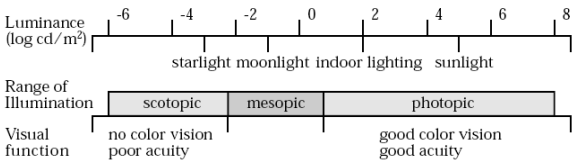

Figure 1: *The range of luminances of a natural environment and associated visual parameters. [FPSG96b]*

Another phenomena of human vision is loss of visual acuity at scotopic levels of illumination where barely rough shapes can be perceived. Further there are changes in perception of color at different light intensities [FPSG96b] and the phenomena of color constancy where the perceived color remains relatively constant under varying illumination conditions [col07].

3. The scene

To demonstrate and test the tone mappers, a small scene with three objects: a simple sphere, a knot and the ogre head has been created. They have an Image-Based Lighting material [Deb02] with a static convoluted environment map [KM99]. The background is a sky-sphere textured with a 32-bit floating point HDR cube map in DirectX's *.dds*-Format [dds07]. Additionally, the demo includes a GUI to control all parameters of the library.

Figure 2 shows a screenshot of the demo application.

Figure 2: *Screenshot of the demo application*

Convoluted Environment Maps extend the environment mapping [env07] technique and include diffuse and phong features to enable the rendering of non-metallic objects. The material simplifies a Bi-Directional Reflectance Distribution Function (BRDF), which describes the reflectance of a material from all points of view over the whole hemisphere, considering the light irradiance over the whole hemisphere. [Mon06] BRDFs are a huge amount of data, but they can be simplified to a good approximation. It is achieved by convoluting the environent map using a filter kernel that approximates the BRDF. This creates a lookup-texture for the diffuse light and another for the phong reflectance. The specular reflection is looked-up in the environment map like in common environment mapping. Therefore it does not support self reflection and only considers a static environment, which does not allow reflection of dynamic objects in a scene. The problem of no reflections of other objects, could be accomplished by updating and convoluting the environment map in realtime, but because the filter kernels are quite large it would radically reduce the performance.

Figure 3 shows the environment map, original from [Deb02], and the in course of the work of Monitzer [Mon06] created phong and diffuse map, both generated with the tool CubeMapGen [Cub].

In the material-shader the diffuse term is looked up with the surface normal, the phong and specular term is looked up with the reflected view vector. Each gets multiplied with an adjustable factor. Together they are used to calculate the surface color:

$$Color = MatColor \cdot (Diffuse + Phong + Specular) \quad (1)$$

With this formula, a range of materials, from diffuse to fully reflective can be created.

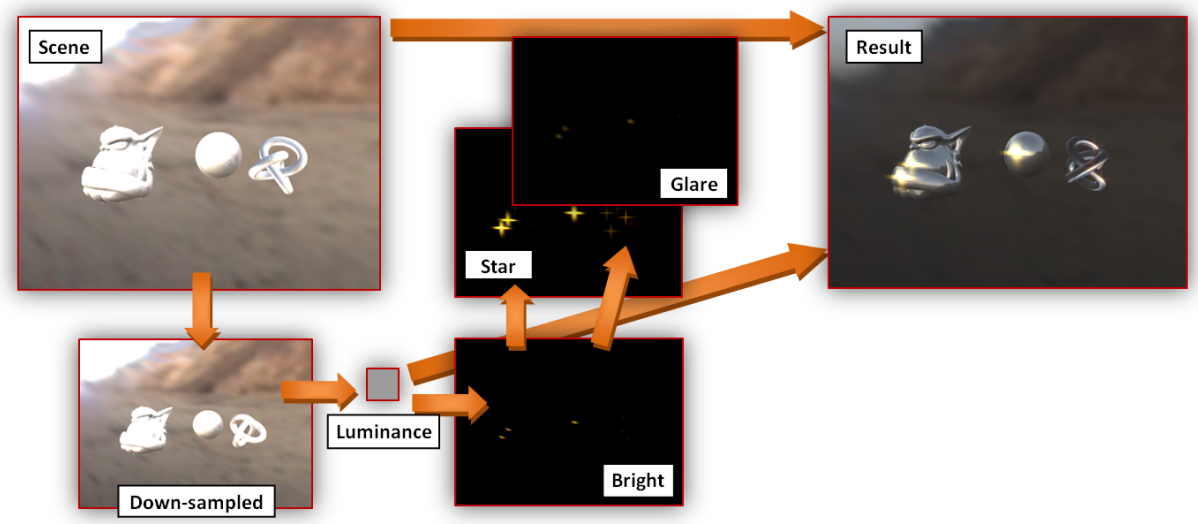

Figure 4: *HDR rendering processing chain*

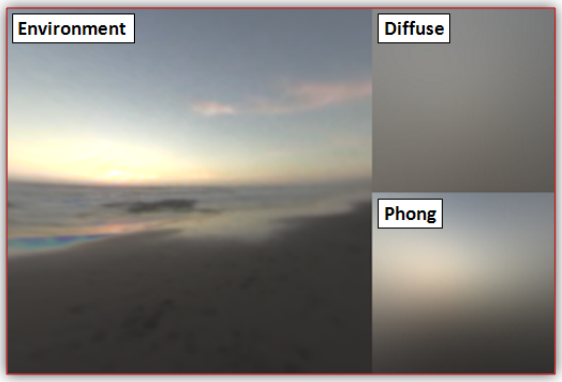

Figure 3: *One face of the environment map and the convoluted diffuse and phong map, created with CubeMapGen [Cub]. The original environment map is from [Deb02].*

4. Processing overview

The whole process of HDR rendering requires a lot of temporary render-targets. Therefore it comes to many render-target switches each frame. Furthermore the performance of the process rather depends on the memory bandwidth and the pixel fill-rate of the graphics device.
Figure 4 gives a general overview of the processing steps.
For HDR rendering, the following sequence has to be calculated:

1. Render the scene
2. Down-sample the scene
3. Extract the average luminance
4. Filter bright areas
5. Build the glare effect
6. Build the star effect
7. Apply final tone mapping and merge with effects

Additional we do the following two steps:

a) Adapt luminance
b) Calculate key

The next sections will explain each step in detail.

5. Rendering the scene

The processing begins by rendering the scene in a floating-point render-target of same size as the viewport. All materials of the scene have at least to be Shader 2.0, where all light calculations are done in high dynamic range. Otherwise the render-target will get invalid values, because of lower shader versions or fixed function pipeline do not support rendering to floating-point render-targets.
To improve the performance, the next step is to sample the scene down, into a temporary texture, to a quarter of its original size. With that texture all further calculations will be done. The lost of spatial accuracy does not matter for calculating the average luminance or building the glare and star effect. Every pixel of the down-sampled scene is calculated by building the arithmetic mean of the corresponding 4x4 block of the original scene.

6. Calculating the average luminance

The average luminance is needed to do the proper final tone mapping and it is also required to decide the bright areas

of the scene which are needed to build the glare and star effect. A suitable way to measure the luminacne is to build a logarithmic average of all pixel luminances in the scene [Rei02]:

$$\bar{L}_w = \exp\left(\frac{1}{N}\sum\left(\log\left(\delta + L_w(x,y)\right)\right)\right) \tag{2}$$

To calculate the luminance of a pixel one can take the arithmetic mean of red, green and blue, but in this implementation a weighted sum like in the YUV color model is used, because it covers the human perception of color more closely. It follows the equation:

$$L_w(x,y) = \overline{RGB}_{xy} \cdot \begin{pmatrix} 0.2126 \\ 0.7152 \\ 0.0722 \end{pmatrix} \tag{3}$$

The average luminance calculation of the scene can be done in four passes. The first pass starts with a texture with a size of 64x64. For each pixel of this texture the logarithmic average of a 3x3 pixel group from the down-sampled scene is calculated. The offset of the texture coordinates for the group is equivalent to one pixel of the down-sampled scene. However, this method can only acquire 192x192 pixels of the down-sampled scene, with is equivalent to an original resolution of 768x768. Due to the slightly higher usual original resolution, not all pixels from the scene are acquired, but they are enough to get an adequate result.
In the next passes the 64x64 texture gets sampled down. First to a size of 16x16, then 4x4 and finally to a 1x1 texture, where the logarithmic average luminance also gets mapped back to world luminance by the exponential function. The pyramidal character of this step can be seen in Figure 5.

Figure 5: *Calculating the luminance*

Theoretic this calculation could be done in one pass with a dynamic flow control by iterating all pixels in two nested loops on newer graphics hardware. However, one would lose the benefit of parallel processing with multiple shader units, but it may proof to be efficient on future hardware, saving one or two temporary steps.

7. Exposure control

By knowing the average scene luminance we can map the high dynamic range spectrum of the scene to a low dynamic range. A simple way to achieve that would be a linear mapping function:

$$L_{scaled} = a \cdot \frac{L_w}{\bar{L}_w} \tag{4}$$

The key value a adjusts the brightness of the final image. Figure 6 shows the result of different key-values.
Using a constant key leads to the problem that the final image always has his certain luminance even if there is darkest night or brightest sunlight. Furthermore if one define a certain key for a specific stetting it probably will not fit for another one. Two pictures of Figure 7 shows these problems. The work of Grzegorz [KMS05] comes with the solution to use a formula to calculate the key automatically. This function has to be adapted to the scene where one want to use it. For our demo scene we use this function:

$$Key = \max\left(0, 1.5 - \frac{1.5}{\bar{L}_w \cdot 0.1 + 1}\right) + 0.1 \tag{5}$$

It has been devoloped by testing with different key values in several luminance conditions. Figure 8 illustrates the behavior.

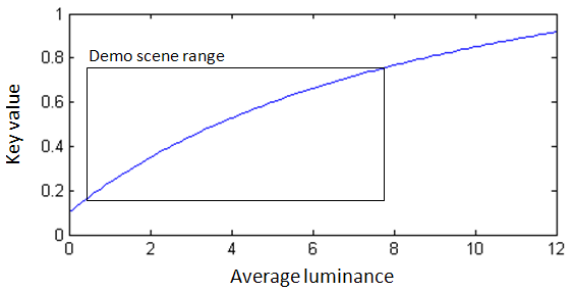

Figure 8: *Automatic key calculation*

The result is a key that produces a satisfactory final image in both setting of the camera in Figure 7.

8. Luminance adaptation

The human perception of luminance does not work as strict as the formula above, it adapts over time. The time course of adaption differs on wether we adopt to high or low intensities, and if we perceive the light using rods or cones [FPSG96a]. Rods are more sensitive to low light intensities and are rather responsible for peripheral vision. Cones are responsible for the perception of color and only work in relatively bright light.
In the library we adopt the model described in [KMS05]. To simulate this behavior the luminace gets temporary adapted

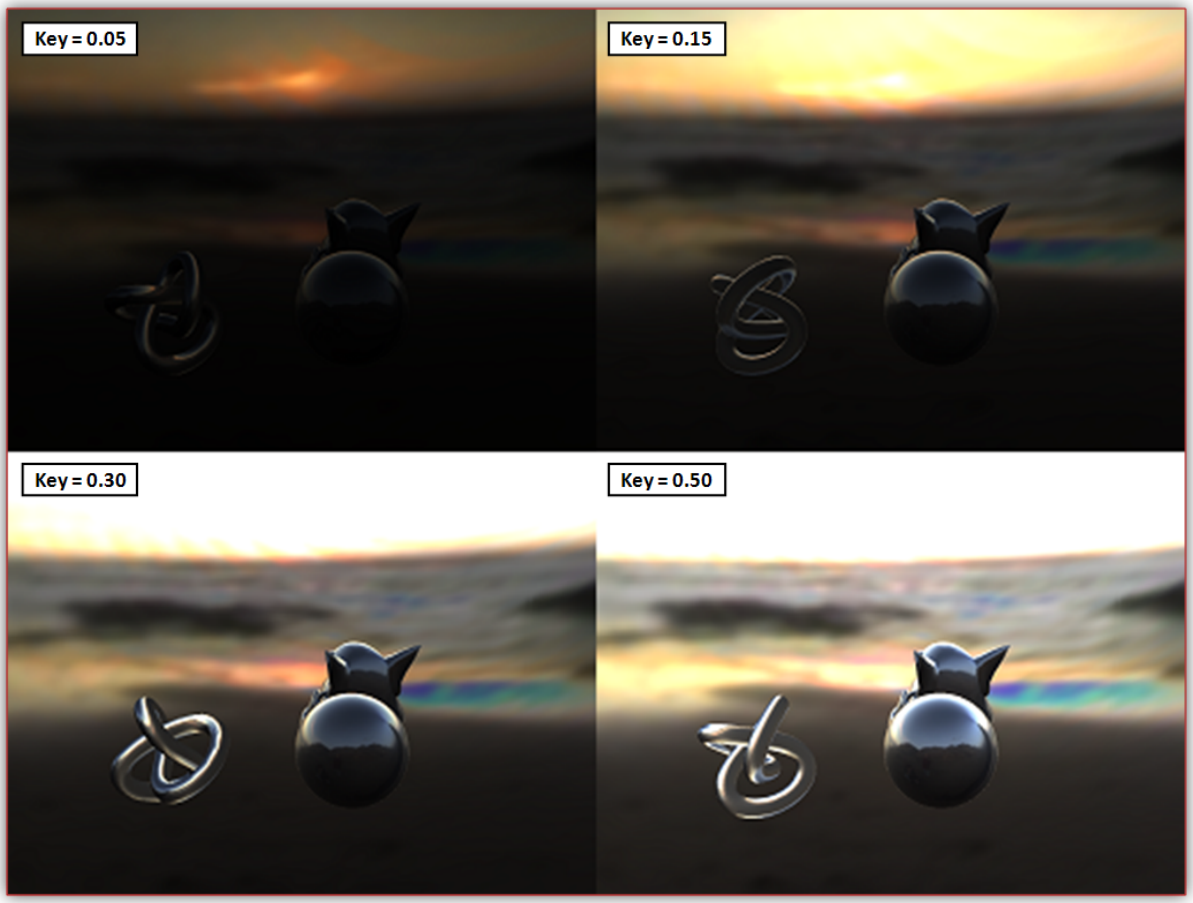

Figure 6: *Results with different key-values*

by using an exponential decal function:

$$\bar{L}_a^{new} = L_a + (\bar{L}_w - L_a) \cdot \left(1 - \exp\left(-\frac{T}{\tau}\right)\right) \quad (6)$$

where T is the elapsed time since the last frame. The adaptation constant τ is different for rods (0.4sec) and cones (0.1sec). The actual adaptation constant is calculated by interpolating between these two values depending on the actual scene luminance. Cones start to lose their sensitivity at $3.4\frac{cd}{m^2}$ and become completely insensitive at $0.03\frac{cd}{m^2}$. So the sensitivity σ of the rods can be calculated with:

$$\sigma = \frac{0.04}{0.04 + \bar{L}_w} \quad (7)$$

The interpolation formula to calculate the adaption constant is:

$$\tau = \sigma \cdot 0.4 + (1 - \sigma) \cdot 0.1 \quad (8)$$

Using this model requires to calculate with real world luminance values otherwise the adaptation is not correct. Instead one can take a simpler model with a constant adaptation.

9. The bright-pass filter

The bright-pass filter extracts the bright regions of the scene. This is the first step of building the glare and star effect. We build the bright-pass with the same resolution than the down-sampled scene, because for the star and glare effect a lower spatial resolution is enough. The bright-pass texture already is in device color space and has 8-bit color depth per channel. First the HDR color value is linearly mapped down to the final color value:

$$L_{scaled} = a \cdot \frac{L_w}{\bar{L}_w} \quad (9)$$

Then a threshold t, that determines the minimum color level for the bright region, is applied to the value and the result gets clamped by zero to suppress negative values.

$$L_{bright} = \max(L_{scaled} - t, 0) \quad (10)$$

Figure 7: *Showcase of the automatic key calculation in comparison to a constant key.*

Because of the linear mapping bright lights still might have relative high color values. In a final step the color values get mapped in the range $[0,1]$:

$$Bright = \frac{L_{bright}}{o + L_{bright}} \tag{11}$$

The offset o controls the separation between bright objects and the more intensive lights. For the demo scene a rather small offset $o = 1$ and threshold $t = 2.5$ are used, because the dynamic range of the scene is not that high and a strong effect should be produced. Figure 9 illustrates the result with these parameters.

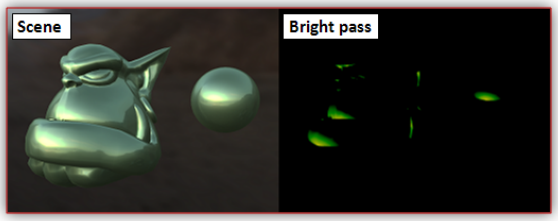

Figure 9: *Result of the bright pass*
Note: The left image is the final output of the tonemapper.

10. Glare effect

The glare effect is caused by bright intensities light which exhibits a blooming effect that takes over neighboring regions. It is produced by the scattering of light within the human eye or photography equipment. In the past, this effect has been simulated by rendering billboards on top of the lights, but this approach is rather static and can create some unaesthetic artifacts.

A more realistic way to accomplish this effect is to blur the scene and blend it to the final render target. To blur we use a

5x5 Gaussian filter-kernel F_{xy}, with $\sigma = 2$:

$$F_{xy} = \frac{1}{\pi \cdot \sigma^2} \cdot \exp(-\frac{x^2+y^2}{2 \cdot \sigma^2}) \quad (12)$$

Only the 13 center-pixels $|x| + |y| <= 2$ get sampled in the shader, because of the limitation of Shader 2.0. It is important to normalize the sum of the weights. Figure 10 shows the according distribution on one axis. The red border marks the range that is used and one can see that the weights there are still rather high. Usually the kernel size should be bigger, approximately $2 \cdot \downarrow (3\sigma) + 1$ that is 13x13, but that would overkill the graphic device just for a slight smoother result.

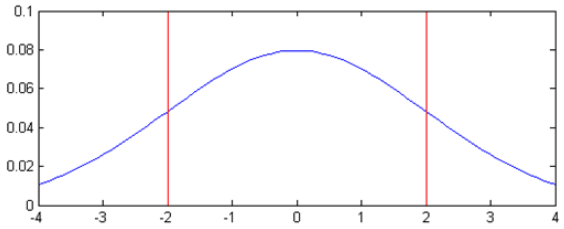

Figure 10: *Gauss distribution with* $\sigma = 2$

The weight values get each handled with their texture coordinate offsets $(\frac{x}{Tex_{xl}}, \frac{y}{Tex_{yl}}, F_{xy})$ as a $float3$ array to the shader. The source for blurring is the bright-pass output, because the glare effect is only caused by bright regions. The output size is the same as the bright-pass which is the same as the down-sampled scene; beside down-sampling is also a kind of blurring.
In the shader the sum of the 13 samples from the source texture with the offset multiplied by the weight is calculated per pixel. Figure 11 illustrates the result of an extracted 80x60 region.

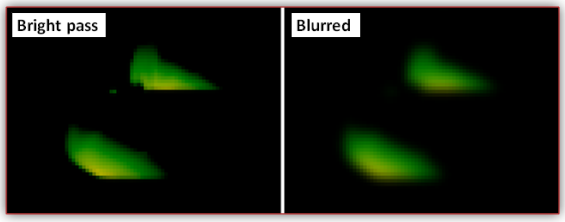

Figure 11: *bright pass convoluted with the 5x5 Gaussian filter-kernel*

To achieve a bigger filter-kernel, the blur filter can be repeated on the output.

11. Star effect

Another phenomena that occurs when filming bright light is that there is a star-like glow with streaks in various directions around the light. This is caused by microscopic bumps and scratches in the camera optic which produce internal reflections and refractions. This effect is also noticed when driving in a car and watching other cars headlights through the windshield. [SL04]
To achieve this effect we use a special blur filter. The number of passes needed to generate the star effect depends on how many and how big the generated streaks are.
The library builds a horizontal and a vertical streak, each in a separate render target, from the bright pass. We build an array with texture coordinate offsets and weights for the samples. The weights are defined from a normalized one-dimensional gauss distribution (Equation 13); with $\sigma = 2$, and the offset d for each pixel.

$$F_d = \frac{1}{\pi \cdot \sigma^2} \cdot \exp(-\frac{d^2}{2 \cdot \sigma^2}) \quad (13)$$

Shader 2.0 has a limit on texture samples, but the blurring can be repeated to increase the effect dimension.
Finally the vertical and horizontal blurs get merged by:

$$Star = \max(StarV, StarH) \quad (14)$$

Figure 12 illustrates the result of an extracted 80x60 region.

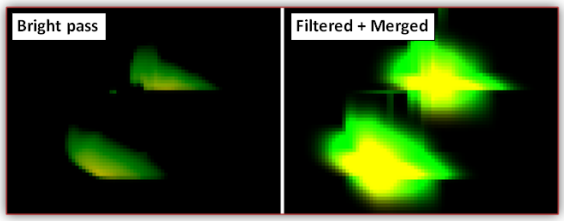

Figure 12: *Final star effect*

In the library only the number of passes, which control the size, and the strength, used when blending all together, of the star and glare effect can be configured. To see more characteristics of these effects one can take a look at the DirectX HDRLighting Sample [HDR]. It provides a small library where these effects are shown.

12. The final pass

In the last pass the scene gets mapped into low dynamic range by a specific tone mapping operator and the glare and star effect are added. So far the library supports five different global tone mappers: Linear mapping, the Reinhard's operator [RSSF02], the modified version of it, logarithmic mapping and the adaptive logarithmic tonemapper [DMAC03].

13. The tone mapping operators

This section describes the implemented tone mapping operators and finally shows a comparison of the different results.

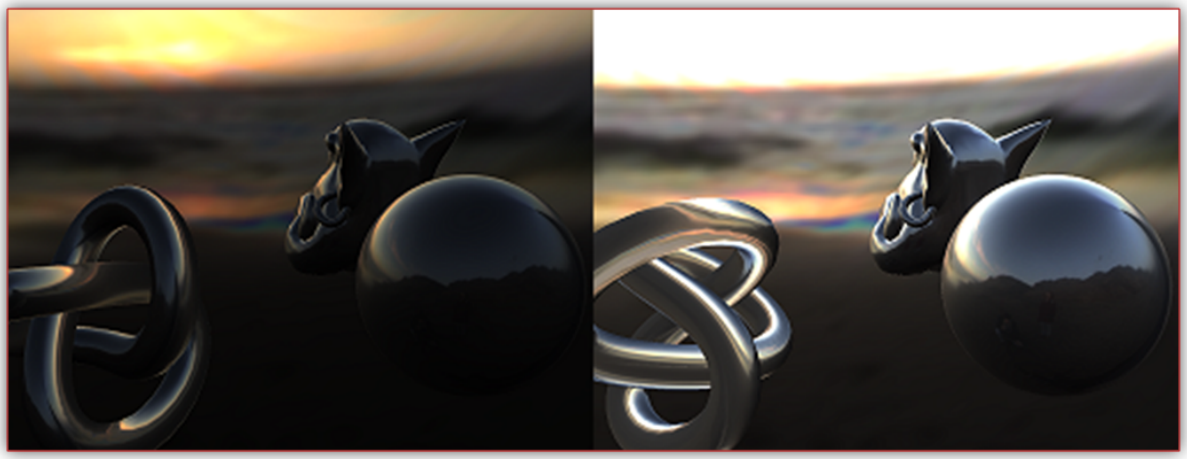

Figure 13: *Result of linear mapping*
The left image scales nearly the whole spectre in LDR, the right scales to an average luminance

13.1. Linear mapping

It is the simplest and fastest mapping. The high dynamic range is simply scaled to fit the LDR. Additionally the key a controls the brightness, but actually it is no real tone mapping.

$$Color = \frac{a \cdot L_w}{\bar{L}_w} \tag{15}$$

Conditional on the linear mapping only colors with luminance values close to the average scene luminance, scaled by the key, are clear and have high contrast. The left image in figure 13 has been created with a low key so that detail in the sky can be noticed, but the rest of the scene is rather dark, although the sky starts to overexpose. The right image has a higher key and shows more detail in most parts of the scene, but the sky and the reflections are just white.

13.2. The Reinhard's operator

The Reinhard's operator is named after Erik Reinhard who published this tone mapper in his work [RSSF02]. It uses a non linear mapping to display the whole dynamic range of an image.
The first step is to scale the HDR with a key and the logarithmic average scene luminance to LDR using Equation 15. Further it makes a compression of high luminance values with this formula:

$$Color = \frac{L_{scaled}}{1 + L_{scaled}} \tag{16}$$

This operator scales high luminance values by $\frac{1}{L}$ and low values by almost 1, so it maps all values in the range $[0,1]$. Figure 15 illustrates the mapping function.
The weakness of this operator is that the colors shift to grey and it never reaches pure white. In the left image of figure 14 these effects are visible: light colors have only barely contrast and are rather grey.

13.3. The modified Reinhard's operator

Reinhard further developed an improved version of his original operator to reduce the last mentioned weaknesses of the loss of color and the infinite mapping. [RSSF02]
Like the Reinhard's operator the first step is to map the scene colors to LDR using Equation 15. Now it uses an extended version of the Reinhard' operator (Equation 16). The new function is a blend between the Reinhard's operator and linear mapping:

$$Color = \frac{L_{scaled} \cdot \left(1 + \frac{L_{scaled}}{L_{white}^2}\right)}{1 + L_{scaled}} \tag{17}$$

This formula allows to decide at which luminance L_{white} the mapping should fade out to 1. The library uses a constant $L_{white} = 2.5$, it has enough contrast and the colors are still strong. The improvement over the common Reinhard's operator can be clearly seen in Figure 14.
Figure 15 illustrates the mapping function of the so far discussed tone mappers. One can see that the Reinhard's operator approximates 1 at infinity and therefore always maps all values to low dynamic range. The modified operator also covers most of the values in scenes with an average dynamic range and therefore produces a good result. However, in scenes with a very high dynamic range, details will be lost with the modified operator.

Figure 14: *Comparison of Reinhard's and modified Reinhard's operator.*

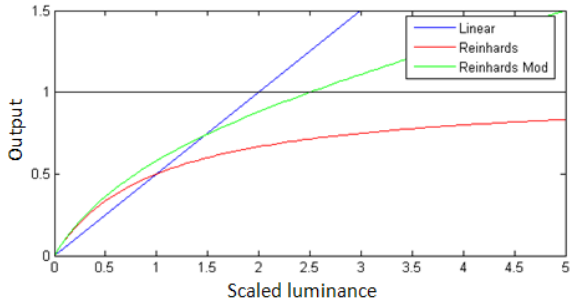

Figure 15: *Mapping functions of the tone mappers.*

13.4. Adaptive Logarithmic Mapping

Another method of tone mapping is adaptive logarithmic mapping [DMAC03]. It uses a logarithmic mapping, following the relation:

$$L_d = \frac{\log_x(L_w+1)}{\log_x(L_{max}+1)} \tag{18}$$

This mapping ensures that always all values will be mapped in range $[0,1]$, but generally the luminance compression is excessive and the feeling of contrast is lost. The approach of [DMAC03] is an adaptive adjustment of the logarithmic base depending on each pixel's radiance. They interpolate between $log_2(L_w)$ (good contrast) and $log_10(L_w)$ (good compression). Bases $x < 2$ or $x > 10$ produce too excessive results. The final equation of their work is:

$$Color = \frac{L_{dmax}\cdot 0.01}{\log_{10}(L_{wmax}+1)} \cdot \frac{\log(L_w+1)}{\log\left(2+\left(\left(\frac{L_w}{L_{wmax}}\right)^{\frac{\log(b)}{\log(0.5)}}\right)\cdot 8\right)} \tag{19}$$

where L_{dmax} is their key to adapt the output; L_w the pixels luminance; L_{wmax} the maximum luminance in the scene and with the parameter b, which should be between 0.5 and 1, the run of the interpolation is controlled. Figure 16 illustrates this mapping function using different values for b.

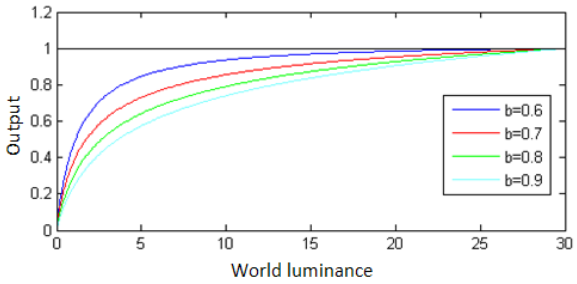

Figure 16: *Adaptive logarithmic mapping function with different values of b.*

To use this method in a realtime application it had to be modified a bit. The usage of the maximum scene luminance for scaling is a problem, because it produces heavy changes of the adaptation when moving in the scene. It has been tested to only calculate the maximum luminance of the 4x4 luminance pass, but this is still useless. So we also use the logarithmic average luminance with a key value a to adjust the mapping. Further we use a constant parameter $b = 0.7$; now the function is:

$$Color = \frac{a}{\log_{10}(\bar{L}_w+1)} \cdot \frac{\log(L_w+1)}{\log\left(2+\left(\left(\frac{L_w}{\bar{L}_w}\right)^{\frac{\log(0.7)}{\log(0.5)}}\right)\cdot 8\right)} \tag{20}$$

This method has been developed to map all values, like the Reinhard's operator. The additional parameter b gives a

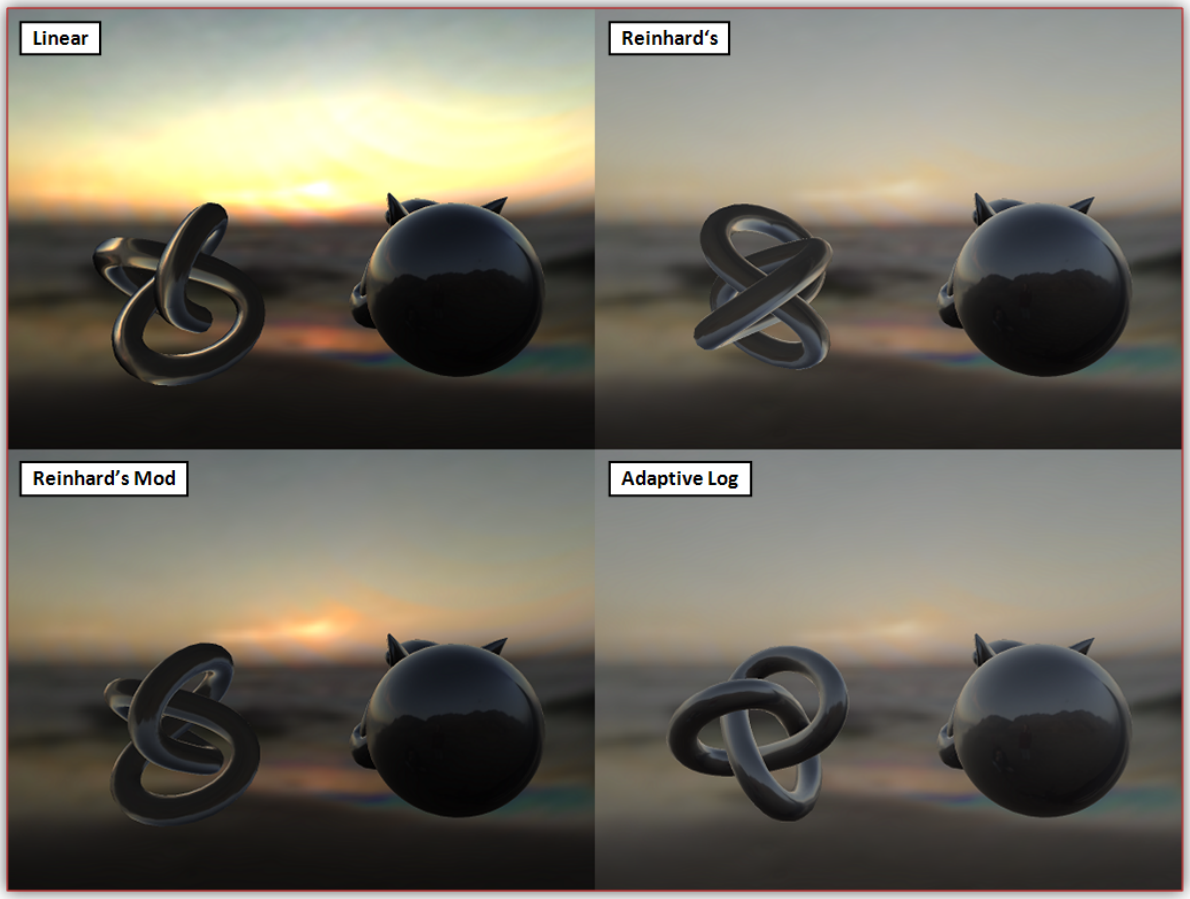

Figure 17: *Comparison of all tone mapping operators. Note: Glare and star effect are disabled.*

better way to control the mapping, but it makes it harder to configure and it can't be automated. However, in figure 17 one can see a slight advantage over the Reinhard's operator.

13.5. Comparison

Figure 17 compares all tone mapping operators. For these purposes, the glare and star effect as well as the luminance adaptation have been disabled, and the key has manually been set to a suitable value. The image shows that the Reinhard's operator and the adaptive logarithmic mapping have a good contrast and show all details of the scene. However, the image looks rather grey, but details can be seen quite good even in the low reflection on the objects. With the modified Reinhard's operator one lose a bit of the contrast, but the colors are more saturated. The sky in the image of the linear mapping is overexposed, while the other parts are quite dark.

14. Conclusion and future work

There is no general solution to tone mapping, a balance between performance and image quality must be assessed on a per scene basis. To choose a suitable tone mapper requires knowing how each tone mapper works, their differences, strengths and weaknesses.
A full featured HDR library for Ogre3D has been implemented and its features tested on different scenes, therewith a direct comparison was possible.
Because most of the tone mapping techniques have been developed for offline rendering, these techniques can only be partly applied in realtime rendering. The parameters for the tone mapping operators also have to be set to a constant or automatically estimated value, some works have not been developed for that use. Therefore the Reinhard's operator is a good choice. In comparison to adaptive logarithmic tone mapping the Reinhards's operator is not that complex with nearly the same result.
With more powerful graphics devices coming up, local tone mappers become applicable with acceptable performance in

realtime, and additionally improve the realism in realtime graphics. Complex models of human perception can be implemented to give a more natural impression when moving through a scene as well.

References

[col07] Color constancy, March 2007. http://en.wikipedia.org/wiki/Color_constancy.

[Cub] Cubemapgen. http://ati.amd.com/developer/cubemapgen/index.html.

[dds07] Dds file reference. http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp.

[Deb02] DEBEVEC P.: Image based lighting tutorial. In *IEEE Computer Graphics and Applications* (2002).

[DMAC03] DRAGO F., MYSZKOWSKI K., ANNEN T., CHIBA N.: Adaptive logarithmic mapping for displaying high contrast scenes, 2003.

[env07] Environment mapping, March 2007. http://en.wikipedia.org/wiki/Reflection_mapping.

[FPSG96a] FERWERDA J. A., PATTANAIK S. N., SHIRLEY P., GREENBERG D. P.: A model of visual adaptation for realistic image synthesis. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 249–258.

[FPSG96b] FERWERDA J. A., PATTANAIK S. N., SHIRLEY P., GREENBERG D. P.: A model of visual adaptation for realistic image synthesis. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 249–258.

[HDR] Directx hdrlighting sample. DirectX SDK (December 2006).

[KM99] KAUTZ J., MCCOOL M. D.: Interactive rendering with arbitrary BRDFs using separable approximations. pp. 253–253.

[KM00] KAUTZ J., MCCOOL M. D.: Approximation of glossy reflection with prefiltered environment maps. In *Graphics Interface* (2000), pp. 119–126.

[KMS05] KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Perceptual effects in real-time tone mapping. In *Spring Conference on Computer Graphics 2005* (Budmerice, Slovakia, 2005), Jüttler B., (Ed.), ACM.

[Luk] LUKSCH C.: Hdrlib. http://www.ogre3d.org/wiki/index.php/HDRlib.

[Mon06] MONITZER A.: Complex material in realtime graphics.

[Ogr] Ogre (object-oriented graphics rendering engine). http://www.ogre3d.org.

[Rei02] REINHARD E.: Parameter estimation for photographic tone reproduction. *J. Graph. Tools 7*, 1 (2002), 45–52.

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

[SL04] ST-LAURENT S.: *Shaders for Game Programmers and Artists*. May 2004.