Diplomarbeit

Real-Time Tone Mapping Using Selective Rendering

Ausgeführt am Institut für
Computergraphik und Algorithmen
der
Technischen Universität Wien

unter Anleitung von
Univ.Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer
und
Dipl.-Ing. Dr.techn. Alessandro Artusi

durch
Benjamin Roch
Krongasse 4/4
1050 Wien

Wien, Oktober 2007 _____

Abstract

High dynamic range imaging is an emerging technology, which tries to overcome the limitations of conventional photography and film equipment. Therewith it is possible to capture high contrast scenes without clamping effects. In order to present such scenes on conventional displays it is needed to have a method for somehow converting the high-contrast values into the range [0,1] preserving visual impression. This operation is called tone mapping.

Though most of the tone mapping operators are fast, they are not fast enough to be directly used in real-time applications. Implementations presented until now have a trade-off in terms of perceivable quality to gain this aim. This work presents an optimized operator, which is able to process data in real-time without loosing quality using selective rendering techniques.

Kurzfassung

In den letzten Jahren wurden Methoden entwickelt um Szenen mit hohem Dynamikumfang aufzunehmen. Diese Technik wird HDR (vom englischen 'High Dynamic Range') genannt. Die dabei enstehenden Bilder können aber nicht auf herkömmlichen Bildschirmen dargestellt werden. Dafür ist eine Skalierung notwendig, welche die Werte in den Bereich [0,1] abbildet und dabei den visuellen Eindruck der Szene bewahrt. Dieser Vorgang wird Tone Mapping genannt. Um Details einer Szene besser herauszuarbeiten ist sog. lokales Tone Mapping notwendig. In dieser Arbeit präsentieren wir Otimierungen um lokales Tone Mapping mit Hilfe des Grafikprozessors ohne erkennbare qualitative Abstriche in Echtzeit durchzuführen.

Contents

Acknowledgements

So Long ...
and Thanks for All the Shoes

NOFX

First of all I want to thank Alessandro Artusi for supervising my work and assisting me. He was a constant source of encouragement as well as a patient adviser. Furthermore I want to thank the Institute of Computer Graphics and Algorithms, especially Prof. Purgathofer for the provided support.

Thanks to Severin Ecker for fruitful discussions on graphics programming, coffee, testing, etc.

Thanks to my friends for taking care of my poker money and beer.

Thanks to my parents for giving me money to play poker and buy beer.

Especially I want to thank Johanna for brighten up my life and taking care of me - Dankeschön.

Chapter 1

Introduction

There is nothing new under the sun
but there are lots of old
things we don't know

Ambrose Bierce

This work is about accelerating local tone mapping by exploiting computational power of the graphics processing unit (GPU) and a technique called selective rendering which is based on psychophysical properties inherent to the human visual system.

Attempts to accelerate local tone mapping always had a trade-off in terms of quality. Most of the optimizations were done in resizing input data or reducing the number of passes.

This is not a problem as long as the size of the display stays small. When comparing the output of such real-time operators with the results of their reference implementation, differences, like missing details, can be recognized.

We propose another approach for accelerating local tone mapping on the GPU. We therefore use a model which is called saliency-based rendering.

Recent publications show, that this principle can be adapted for rendering algorithms which have a high amount of computations to be done on a per-pixel basis, e.g. ray-tracing. These processes can be optimized by selecting the salient regions and process them in high quality, while the rest of the (non-salient) pixels are processed in low-quality. The saliency map is based on a first small preview rendering. This map is then used as an input for

the high-quality rendering pass, which is able to differentiate on a per-pixel basis.

In this work we first explain the basics of the human visual system, which are needed to understand the problem of contrast compression, effects caused by the visual system and perceptual-based rendering.

HDR images and movies have to be somehow taken or generated, which leads us to the problem of acquisition in the following section. An explanation about the tone mapping process especially in terms of real-time applicability and perceptual rendering is done afterwards.

An overview about related work will be presented in chapter 3. It is concerned with published articles, related to tone mapping and aforementioned topics. We will describe the implementations we used as a basis in more detail.

We then present the concepts of our implementation, showing a general approach of how to optimize local tone mapping operators without noticeable loss of quality, followed by a presentation of results we retrieved, comparing several configurations and their performance. A short summary, conclusion and possible future work closes the work.

Related to this work, we have a patent pending with Application No. $GB0709392.5$, see [ARCC].

Chapter 2

Background

What I give form to in daylight
is only one percent of what
I have seen in darkness

M. C. Escher

2.1 Human Visual Perception

Light moving through space can be seen as particles, called photons or waves having a certain wavelength. The so called visible spectrum of light lies between 400 to 700 nanometers (nm). The human eye interprets the spectrum of visible light, depending on its wavelength.

Color is just a representation of light-waves in our brain or as Sir Isaac Newton formulates: "Indeed (light) rays, properly expressed, are not colored". Beside other representations, color can be described in terms of chromaticity and luminance. Chromaticity stands for the "colorfulness", while luminance represents the gray scale-level or perceived brightness of a certain area. Given as absolute unit, luminance is often measured in cd/m^2.

Fig. 2.1 depicts the basic structure of the eye. The light enters at the cornea and is projected through the lens and the vitreous humour onto the retina. The retina houses two types of optical sensors - rods (about 100 millions) and cones (about 7 millions). Their ability to adapt to light are

different, 0.4 sec for the rods and 0.1 sec for the cones. Rods and cones have their own ideal conditions of operation: while the cones best operate at daylight and dim light, they start to loose their sensitivity at $3.4\frac{cd}{m^2}$ and become completely insensitive at $0.03\frac{cd}{m^2}$, where the rods are dominating, see fig. 2.2. The central place of the retina, called macula, houses the majority of cones. This area is responsible for sharp and detailed vision.

There are three types of cones, each having a distinct response curve [Poya] and [Poyb]: the S, M and L[1]. The combination of the three response curves makes up the impression of color. The three types are not equally distributed, which explains the differences perceiving the three color channels, see fig. 2.3.

The rods are the second type of visual sensors in our eyes. They are responsible for night vision. Rods are not able to recognize colors, which is the reason why we are not able to distinguish colors in dark areas. They also have short-comings related to recognizing details.

When changing from a dark room to sunlit surroundings, our eyes need some time to adapt to the outdoor scene, as well as vice versa. Imagine sitting in a dark room, just lit by some external tungsten light and watching TV. When we concentrate on the TV set, everything else around the image gets near black. When we look away from the display and look around the room, things will be noticeable after some time - while the TV set seems now to be very bright. This adaption process makes it possible for us to visually perceive our surroundings at varying lighting conditions.

Contrast describes the ratio between the darkest and the lightest intensity-value. See (table 2.1) for examples of different medias. All of them have a relatively low contrast compared to the sun or the human visual system in common. The limitations inherent to conventional imaging equipment lead to a degradation of details, caused by clamping and/or linear scaling (adaption) of the values to fit in the display-able range.

Painters are confronted with the same problem since beginning of their art - how to depict high contrast scenes on paper, which has only a fraction

[1]S, M, L represent short (blue), mid (green) and long wavelength (red)

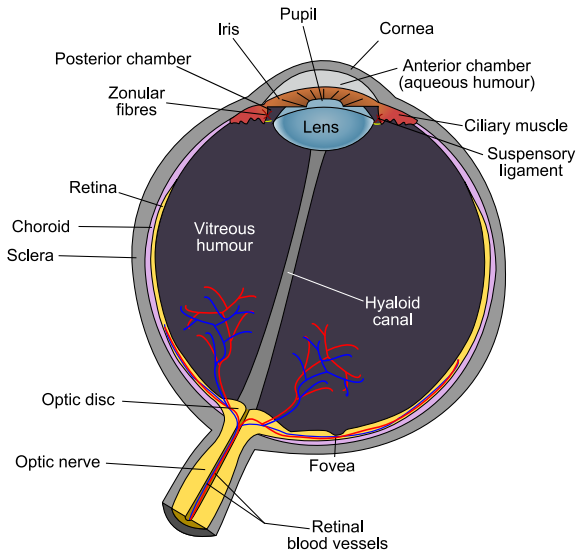

Figure 2.1: *The human eye, taken from [wik]*

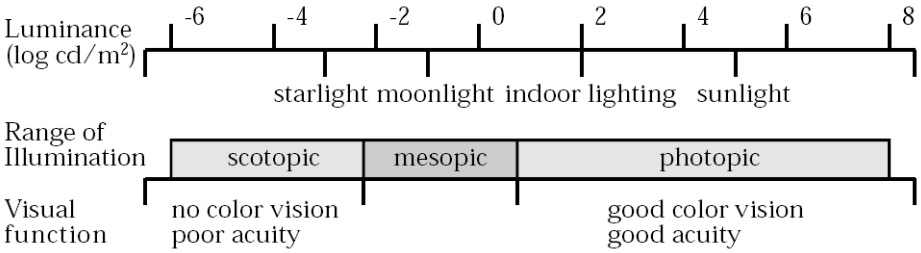

Figure 2.2: *Luminance ranges of the different visual modes, taken from [FPSG]*

	Contrast
Newspaper	1:30
Computer monitor	1:100
Analog Camera film	1:1000
Perceivable by human eye	1:10000

Table 2.1: *Typical contrast ratios*

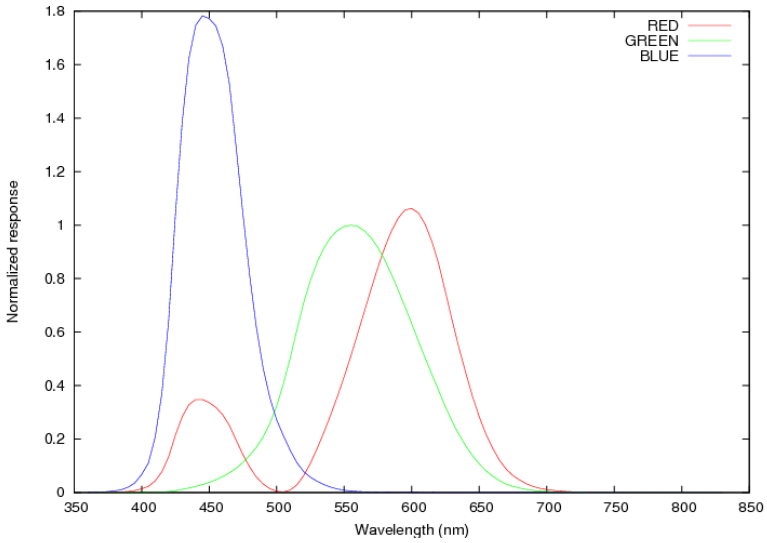

Figure 2.3: *Response curves of the three different types of cones. Data can be downloaded from [cvr]*

of the scene's dynamic range? They overcome the natural limits in contrast by clever arranging layers of paint - from coarse to fine, emphasizing local contrast and edges by strong opposing colors. They try to "compress" the contrast while preserving the natural appeal of the depicted scene. An example is Monet's "Impressions at Sunrise" (fig. 2.4). The colors of the surroundings seem to be very dull and dark. Monet emphasizes depth by blurring more and more. The sun and the reflections on the sea are in stark contrast to the rest of the scene. In terms of brightness as well as in terms of color. The opposing colors seem to further accent the sun.

One of the first attempts to retrieve a higher dynamic range in photography has been done by Sergei M. Prokudin-Gorskii at the beginning of the 20^{th} century. He designed his own camera and projection-system, which allowed colored photographs of high quality by using monochromatic film only. He captured every color channel separately by using red, green and blue filters. He then projected these three photographs combined using three aligned projectors, each equipped with a filter. By changing the light source of each projector he was able to balance the channels to achieve a colorful and vivid picture, which had higher contrast than comparable single film methods.

Nowadays, automatic exposure-meter systems of photo cameras allow to

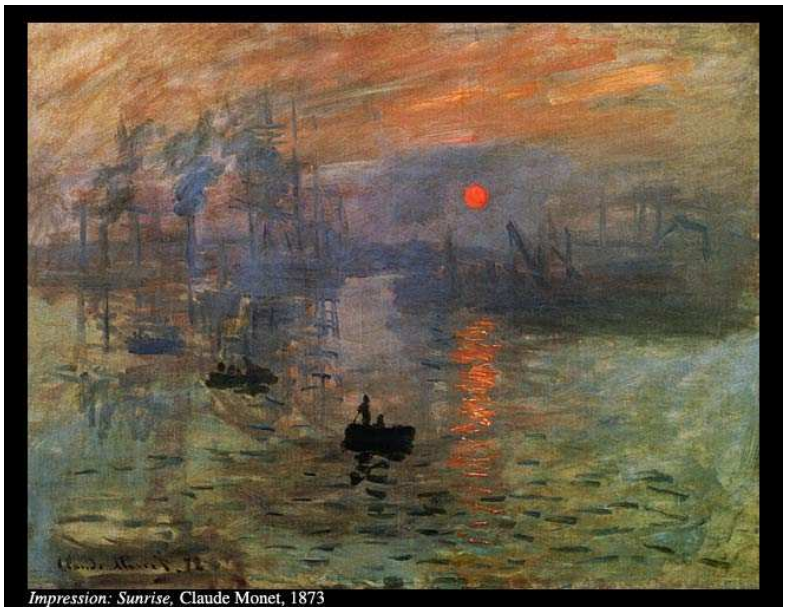

Figure 2.4: *Monet's "Impressions at sunrise" shows the effect of combining complement colors to revive the effect of strong contrast. The sun really seems to "burn" compared to the rest of the scene. Other impressions are also taken into account, like the blurring due to fog on the left side of the painting*

select the right "frame" of contrast which the image has to be taken with. Because of limited contrast inherent to the film material, it does represent only an amount of the real scenes contrast. The camera photographing Buzz Aldrin on the moon (fig.2.5) referred to the sun's light intensity, which is reflected by the moon surface. This caused the stars to be completely invisible.

The solution is to take more than one picture of the same scene, each with different exposure time, e.g. -2EV, 0EV, and +2EV[2]. This is called exposure bracketing - the sequence of produced images is called exposure latitude, see fig. 2.6. The photograph with the lowest exposure time will be very dark, only bright details will be noticeable. Going to the image with the highest exposure time, everything will be very bright, light elements will produce a streaking effect while dark details are are made visible.

Ansel Adams formulated and perfected this method as the zone system

[2]EV means Exposure Value. Zero EV is the exposure time, measured by the camera's sensor. Negative numbers mean underexposure, positive overexposure.

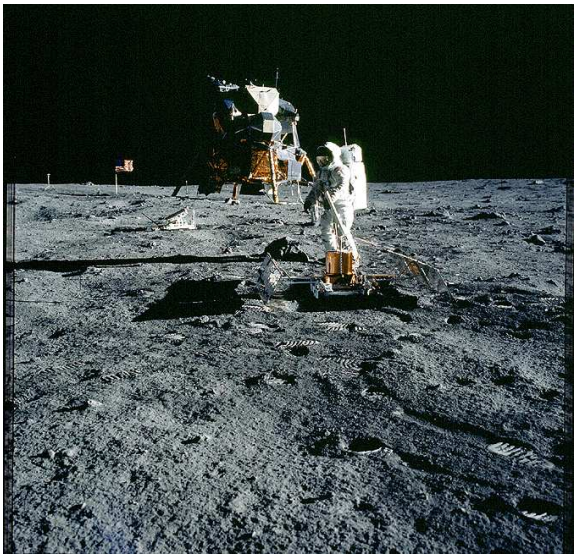

Figure 2.5: *Buzz Aldrin on the moon, conducting a seismic experiment - Where are the stars? Image is taken from [nas]*

[Adaed]. The images are then combined with a technique called "dodging and burning": During film development, regions are masked out while the rest of the film is further processed. Parts of the photograph can be emphasized or suppressed, details can be worked out, accentuating the image more. The result is a photograph, which corresponds more to the human visual system, which adapts to local contrast.

Figure 2.6: *Exposure latitude*

Figure 2.7: *Scene depicted with conventional and HDR method*

2.2 HDR Data Acquisition

The introduction of high dynamic range imaging (HDRI) to computer graphics stems from the need to model the influence of "real" light on artificial objects in a natural way [GC84]. Therefore physical based values are used, which usually exceed the displayable range of $[0.0, 1.0]$.

High dynamic range data can be acquired differently - the most prominent methods are:

- Rendering with an HDR aware system
- Combining multiple exposures (exposure bracketing)
- Film scanning
- HDR capture devices

When creating realistic-looking artificial scenes with rendering systems, physical based lighting is crucial. Lighting simulations, rendered objects copied in real movies, etc. cannot be done plausibly without HDR output. One early example is the Radiance [War] rendering system, which also introduced on of the first HDR image file format [War92].

Most of the HDR images available and nearly all HDR images presented in this thesis, are taken using multiple exposures, called exposure bracketing. This system originally stems from analog photography and was described by Ansel Adams [Adaed]. The basic idea is similar to film scanning - we change

the exposure time of the camera, thereby creating an exposure latitude. It is best to use a tripod and a remote control to avoid shaking. Mostly, it is enough to take three photos of the scene. The exposure is then modified from short (-2 steps), over normal (0 steps) to long (+2 steps), capturing the scene with different lighting intensities. While the short exposure time emphasizes very bright details in the scene, long exposure works out the dark details. Those exposures are then combined with software like pfstools [hdr] into one single HDR image.

Compared to conventional digital media, analog film has a higher contrast. Scanning the film by using different light intensities, it is possible to capture the whole dynamic range on one digital HDR image. This method is comparable to exposure bracketing.

First native HDR capturing devices are commercially available. The surveillance camera SMaL by Cypress Semiconductor is an example. Since their large contrast differences between outdoor and indoor, door entrances are a preferred field for HDR enabled installments. Thereby it is possible to capture both sceneries at the same time without loss of information and need for further adjustments.

2.3 Tone Mapping

Tone mapping is the process of somehow scaling a high contrast scene to obtain a low dynamic range (low contrast) representation, usable for conventional displays. Otherwise the image is clamped to [0,1]. Depending on the domain the operator is using, we differentiate between three types [RWPD06]:

- Frequency-based
- Gradient-based
- Spatial-based

Since the spatial domain is best suited for implementing on the GPU, we don't further discuss the other two domains here and redirect the interested reader to [RWPD06], [LW02] and [TT99] for further informations.

The spatial domain operators can be divided into global and local operators, depending on the scaling they apply. Global operators tone-map the image using the same value for all pixels, while local operators select the scaling value, based upon the examined pixel's neighborhood. Since global operators only apply a unified scaling, they only need one rendering pass, which makes them favored for real-time applications. The easiest method is linearly scale (see fig. 2.9) all pixels of the image by the maximum luminance: $lum = LUM/LUM_{max}$. While this method is the fastest, it has two big limitations: most of the details in the image will vanish, especially in darker regions. The other problem is that it is not adequate to human perception by any means. The visual impression of the result will totally differ from the impression given by the real scene.

Since our work is concerned with accelerating local operators exclusively we will not further discuss global operators. From now on, we use the term "tone mapping operator" as an equivalent for spatial local tone mapping operator.

Nearly all of the tone mapping algorithms operate on the luminance representation of the image. Therefore the HDR RGB[3] image is first converted to luminance with eq. 2.1. The different coefficients are due to the varying response curves of the cones. This representation is then "compressed" by a tone-mapping method. The tone-mapped luminance is then used to bring the original RGB into the $[0, 1]$ range to obtain the low dynamic range (ldr) representation of the image, see alg.1.

$$LUM = 0.2126 * R + 0.7152 * G + 0.0722 * B \tag{2.1}$$

Therefore most of the local algorithms build up an image pyramid (e.g. Gaussian pyramid), consisting of layers of iteratively filtered versions of the original image. This pyramid is then used to extract important features by somehow comparing the levels against each other. Obviously, this process is computationally costly compared to global tone mapping.

[3]We denote high dynamic range (respectively low dynamic range) magnitudes in uppercase (respectively lowercase)

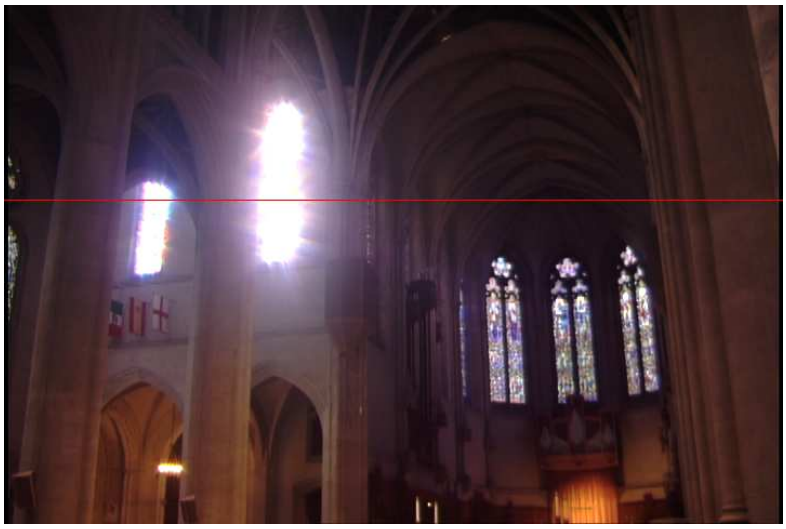

Figure 2.8: *Nave.hdr with selected scan-line (red)*

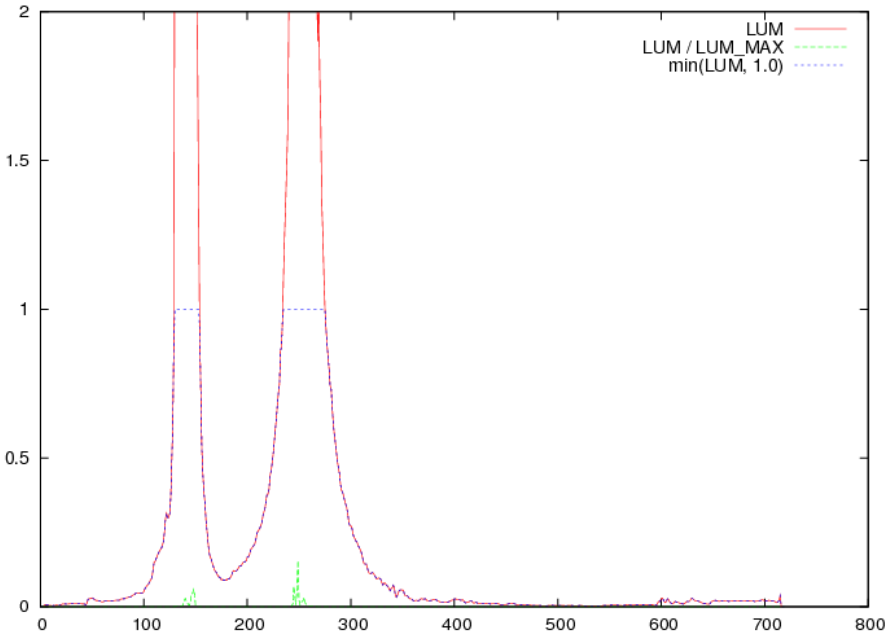

Figure 2.9: *LUM scan-line of nave.hdr compared with two naive methods of bringing HDR images into displayable range of [0.0,1.0]: clamping LUM (blue) and linear scaling LUM (green). Note that the red scan-line is also clamped at 2. Its maximum is at 653*

Algorithm 1 Generic local tone mapping

convert $RGB \Rightarrow LUM$
for each pixel **do**
  for s=1..NUMITERATIONS **do**
    examine neighborhood with radius s
    set pixel's adaption $L_{adaption}$ according to some criteria
  end for
end for
scale LUM by previously computed adaption map
recombine RGB to retrieve tone-mapped rgb

Figure 2.10: *Global vs. local tone mapping*

2.4 Visual Effects

Visual effects are evoked by the construction of the eye and extreme lighting conditions. We will describe here some of the visual effects, like glare, loss of acuity and scotopic vision.

Glare, the effect of veiling luminance around edges of strong light, is caused by scattering of bright light within the eye. Glare prevents the eye from perceiving the scene in a precise way. Instead it causes a blurred impression, especially at the edge of the light source's corona. This effect is also reproducible with a conventional camera, when capturing a scene against the sun (fig. 2.11).

The transitional region from photopic to scotopic vision causes loss of acuity when perceived surroundings get darker. Due the inability of the rods to perform chromatic vision, differentiation between colors gets impossible. Because the rods compact the perceived wavelengths and blue is at the shorter wave-length end of the visible spectrum, there happens to be also a shifting to the blue spectrum, called blue shift.

Alternating lighting conditions and resulting changes in contrast force our eyes to somehow adapt to the new circumstances. Temporal luminance adaption is the mechanism used when our eyes adjust from one light situation to another one, which is noticeable lighter/darker.

The listed effects are the most notable. If tone mapping, especially real-time tone mapping is conducted without applying visual effects, the result is perceived as somehow incomplete.

2.5 Visual Attention

It is well known, that the human visual system does not process complex scenes at once. It does not operate in an ordered fashion, from the upper-left corner to the lower-right. The eyes scan the scene and conceive the most salient objects first by changing the point of interest very fast and alternating between the various objects in the scene. When entering a room with dim lighting, we will first encounter the windows, when the light conditions

Figure 2.11: *An example of glare*

outside are better. Looking at a painting, our visual system first analyzes interesting details, avoiding unimportant regions. The process of scanning the scene for interesting regions is proceeded automatically in a bottom-up fashion, not controlled by the individual.

The opposite of this operation is the top-down process which operates task-oriented. The individual is looking for a dedicated object. The individual knows, what has to be found, therefore the analysis happens on one's own volition. More elements of the human perception involved in the top-down process makes it slower compared to the bottom-up procedure. Both processes work hand in hand to guarantee, despite limited facilities of the HVS, the fast and correct cognition of our surroundings.

The fact, that the HVS identifies important regions, while ignoring other non-interesting ones can be used to vastly speed up pixel-based rendering algorithms, without loosing visual perceptible quality. This means that an observer, who studies two renderings of the same scene is not able (in reasonable time) to differentiate between those two images. Studies conducted in this part of visual perception accumulated into a model called saliency based rendering, which is further explained in chapter 3.

2.6 Motivation

With the emerge of HDR, affordable equipment will be available soon, which allows recording HDR stills and movies without any tweaks. By now, computer games, other interactive content and rendered movies make up a wide field for applying real time tone mapping techniques.

Most of the algorithms used are global algorithms, allowing a fast processing of data. For preserving small details, local operators are essential. It is therefore crucial to optimize local operators to be real-time-applicable. Often this means a trade-off in terms of quality.

In this thesis we want to present several optimizations in terms of speed for an existing local tone mapping operator, without noticeable degradation of quality and relative low memory consumption.

Chapter 3

Related Work

The secret to creativity is knowing how to hide your sources

Albert Einstein

Tone mapping is one of the newer topics of computer graphics. Nevertheless, a lot of algorithms had been invented in the last 20 years. We list here some representatives, describing their function in short. Since this work is not concerned with comparing different tone-mapping operators but accelerating them, we only describe two publications we used as a basis in detail, one local tone mapping operator and one concerned with visual effects.

3.1 Tone Mapping

3.1.1 LCIS

The *Low Curvature Image Simplifier* (LCIS) by Tumblin et al. [TT99] tries to imitate the process of painting. An artist draws from coarse to fine, detailing the picture in every iteration. This method allows very fine control of local contrast and preservation of details by different kinds of shading and gradients. LCIS tries to mimic this process in reverse. By iteratively removing details from the image, the algorithm selects regions which can be further compressed. Smoothing is done with edge preserving convolution to find regions and their boundaries. At the end only the coarse features are

compressed, infusing back the details afterwards. Its attempt to strongly preserve details leads to an over-emphasizing of edges and introduces grain into smooth regions. The need for adjusting 8 parameters and expensiveness in terms of computational power is another problem and makes this algorithm impractical for day to day use.

Fattal et al. [LW02] presented another gradient-based approach to solve the tone mapping problem. Their algorithm tries to preserve the perceived local intensity ratios. First the algorithm computes the log(LUM) and its gradients. The gradient is then attenuated by a function ϕ. The attenuation function is computed based upon a Gaussian pyramid, created from the gradients in a top down fashion. To avoid artifacts due to the multi-resolution edge detection scheme, they found out that it is important to not attenuate each gradient at the resolution it was found, but to propagate it down to the base level. This allows a fast implementation, which can be used in a versatile way. The authors propose to use their algorithm also for ldr images - enhancing contrast in darker regions.

An example for frequency domain-based is the publication of 2002 [DD02] by Durand et al. They propose the usage of an edge preserving filtering technique, based on Gaussian filtering. Basically their algorithm compresses HDR images by blurring noisy textures without blurring strong contrasts (i.e. preserves edges). They use piecewise bilateral filtering in Fourier Domain, which greatly speeds up computation.

3.1.2 Photographic Tone Reproduction for Digital Images

The photographic tone reproduction operator by Erik Reinhard et al. is based on the zone system by Ansel Adams [Adaed]. It is spatial operator. The input RGB is converted to $log(LUM)$. Then the local adaption value is computed, based upon subsequent differences of a Gaussian pyramid. This procedure simulates the process of automatic “dodging and burning“. Every

pixel examined is part of some region, including itself and its neighborhood. On the one hand, we want to enlarge the region as much as possible, while on the other hand we don't want to loose details through smoothing. By enlarging the filter kernel (size s) and subsequent filter kernel (size s+1) and comparing these two, it is possible to find the right scale (adaption value) for every pixel, without loosing details or producing artifacts. Because of simplicity, robustness and speed, it is one of the most popular tone mapping algorithm today.

3.2 Ashikhmin's Tone Mapping Algorithm

Our implementation is based on M. Ashikhmin's paper "Tone Mapping Algorithm for High Contrast Images" [Ash02]. It operates on the HDR luminance. First it computes the luminance extrema (L_{min} and L_{max}). Then it creates a Gaussian pyramid of the input LUM, from level $s = [1..20]$.

$$lc = \frac{|L_s - L_{2s}|}{L_s} \tag{3.1}$$

Afterward, the so called "neighborhood growing procedure" tries to find the right size of the adaption region around every pixel. This is comparable to Reinhard's automatic "dodging and burning" technique. Instead of comparing two successive levels against each other, Ashikhmin's algorithm compares level s with level $2s$, iterating from $s = 1$ up to $s = 10$. In every iteration, the local contrast (lc) is computed for each pixel by 3.1. L_s and L_2s represent the luminance of the pixel at the respective level s.

As long as lc stays under a certain threshold (between 0.0 and 1.0, default at 0.5), we iterate further, setting the adaption value of the current pixel to $la = ls$. Once L_s gets over the threshold or we reach the maximum of iterations, we fix L_s of the last iteration to be the luminance adaption value (la) for that pixel and go on to the next pixel.

$$C(L)=\begin{cases} L/0.0014 & if L<0.0034 \\ 2.4483+\log(L/0.0034)/0.04027 & if 0.0034\leq L<1 \\ 16.5630+(L-1)/0.4027 & if 1\leq L<7.2444 \\ 32.0693+\log(L/7.2444)/0.0556 & otherwise \end{cases} \tag{3.2}$$

$$TM(L) = \frac{C(L) - C(L_{min})}{C(L_{max}) - C(L_{min})} \tag{3.3}$$

The resulting map, which holds the luminance adaption values for the whole image is called LUM_a, which is then tonemapped with eq.3.3 which uses the TVI function. The TVI (threshold vs. intensity) function, which is taken from [FPSG] and depicted in 3.1, simulates the response of the human visual system to luminance values against a certain background. Since the function is too complex to be integrated analytically, it is approximated by four line segments, see eq. 3.2.

The tone-mapped LUM_a is called display luminance map (LUM_d). This is then recombined with the HDR RGB and LUM_a by eq.3.4 to retrieve the final tone-mapped rgb image.

$$rgb = RGB * \frac{LUM_d}{LUM_a} \tag{3.4}$$

3.3 Real-Time Tone Mapping

3.3.1 Goodnight's interactive Tone Mapper

Goodnight et al. [GWWH03] demonstrated a GPU accelerated version of Reinhard's operator. They use a modified pixel layout - one pixel (RGBA-format) holds four adjacent luminance values (i.e. packed structure), which loads four luminance values with one texture read. An additional variable is then loaded with the filter values and a dot product computes the filtered value. For a 1D filter with size=11 (radius=5), this means that it needs $\lceil 11/4 \rceil = 3$ passes and accumulation of the values to retrieve the final filtered value. For separable 2D filter with size $n \times n$ it needs $n/2 + 2$ render passes.

Additionally to the two frame-buffers used for the Gaussian blur, their implementation uses 2 additional buffers for adaption zone computation (one

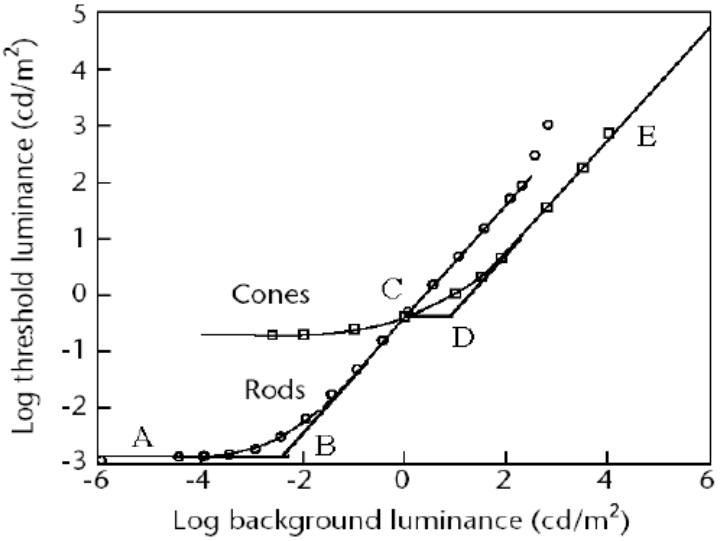

Figure 3.1: *The TVI function from [Ash02]*

for read and one for write) - the gauss buffers and the zone buffers are used in an alternating fashion (ping-pong)3.2, see section 4 for an explanation on ping-pong rendering. The process works as described in alg.3.3.1.

Algorithm 2 Implementation of Reinhard's algorithm by Goodnight et al.

for s=1..N **do**

Compute level s_{i+2} of the Gaussian pyramid

Set zone buffer as render target

Bind levels s_{i+1} and s_{i+2} of Gaussian pyramid as well as the other zone buffer

Render with the zone fragment shader to the zone buffer

end for

They implemented a simple temporal luminance adaption to avoid large discontinuities when panning over an image or during an animation.

$$\frac{dm}{dt} = \frac{m^* - m}{\tau} \tag{3.5}$$

Due the high number of texture reads, the implementation of Goodnight et al. is only suitable for small sizes of input data.

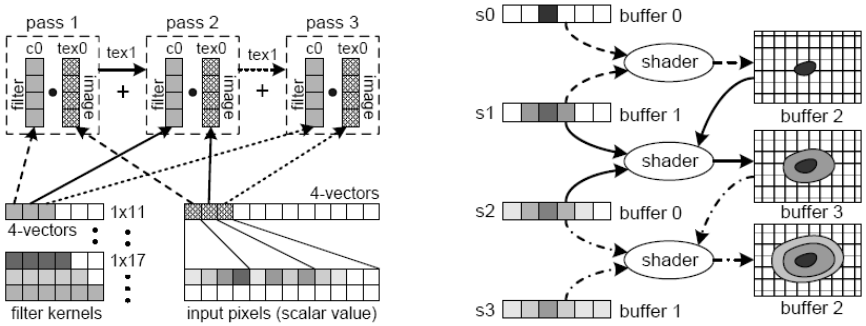

Figure 3.2: *Gaussian convolution (left) and accumulation of adaption zones (right) to the zone buffer in the implementation by Goodnight et al. [GWWH03]*

3.3.2 Real Time Effects by Krawczyk et al.

Krawczyk et al. [KMS05] also took Reinhard's algorithm as basis. They also presented a real-time implementation. Instead of using the packed luminance structure of Goodnight, they did not modify the pixel layout. They also separated the Gaussian filter ($2 \times 1D$). They use a real Gaussian pyramid (therefore each level is a down-sampled version of the previous one) and therefore perform faster than the previous one. They also include more visual effects simulation:

- Glare
- Photopic and scotopic vision
- Loss of acuity
- Blue shift

They propose to use the Gaussian image pyramid based on the input luminance to simulate these effects. The most prominent effect, glare, (bloom around a highlighted region) is caused by strong light, which is scattered in the eye. This effect appears only in highlighted regions, i.e. it is not reproduced by a tone-mapped image. Because the lack of such blooming produces an unnatural look, it has to be simulated. They also implemented simulation of loss of acuity and scotopic vision.

3.4 Visual Attention

Itti et al present a concept [IKN98], which concentrates solely on the bottom-up process, i.e. non-volitional part, of the visual attention model. They create a so called saliency map, which consists of several feature maps extracted from the scene. These feature maps can be divided into three groups: intensity, colors and orientations. Gaussian pyramids are built from the intensity and the colors, while Gabor pyramid are built to form the orientations map. This maps are then, after the center-surround differences stage and normalization combined to the saliency map. The saliency map then represents the important parts of the scene.

As explained in 2, visual attention is used by primates to identify important objects. We now explain the process for computing the saliency map.3.3 in a short form.

First the intensity map is computed with $I = (r + g + b)/3$, with which then a Gaussian pyramid with 9 levels is created3.3. The red, green and blue channels of the input image are then normalized by I, but only at locations, where $I > 1/10$ of its maximum. Locations, having $I <= 1/10$ set r,g, and b to zero, because color perception is not given at such low luminance levels.

Then four broadly tuned color channels are generated: $R = r - (g + b)/2$, $G = g - (r + b)/2$, $B = b - (r + g)/2$ and $Y = (r + g)/2 - |r - g|/2 - b$ (for yellow), negative values are set to zero. A set of six intensity maps is created, I(c,s), with $c \in \{2, 3, 4\}$ and $s = c + \delta$, $\delta \in \{3, 4\}$, as $I(c, s) = |I(c) \ominus I(s)|$.

A pair of opponent-color pyramids is computed by $RG(c, s) = |(R(c) - G(c)) \ominus (G(s) - R(s))|$ and $BY(c, s) = |(B(c) - Y(c)) \ominus (Y(s) - B(s))|$. Local orientation maps are computed using the Gabor filter, as $O(\sigma, \theta)$, where $\sigma \in [0..8]$ represents the scale and θ the degree, $\theta \in \{0, 45, 90, 135\}$. These maps are then used for computing the orientation feature maps: $O(c, s, \theta) = |O(c, \theta) \ominus O(s, \theta)|$. In total 42 maps are computed - 6 intensity maps, 12 color maps and 24 for orientation.

In the next step, these maps are then combined into three "conspicuity maps" $\overline{I}$, $\overline{O}$ and $\overline{C}$. The conspicuity maps are then normalized and combined into the saliency map.

The real-time performance of the above mentioned model is limited due

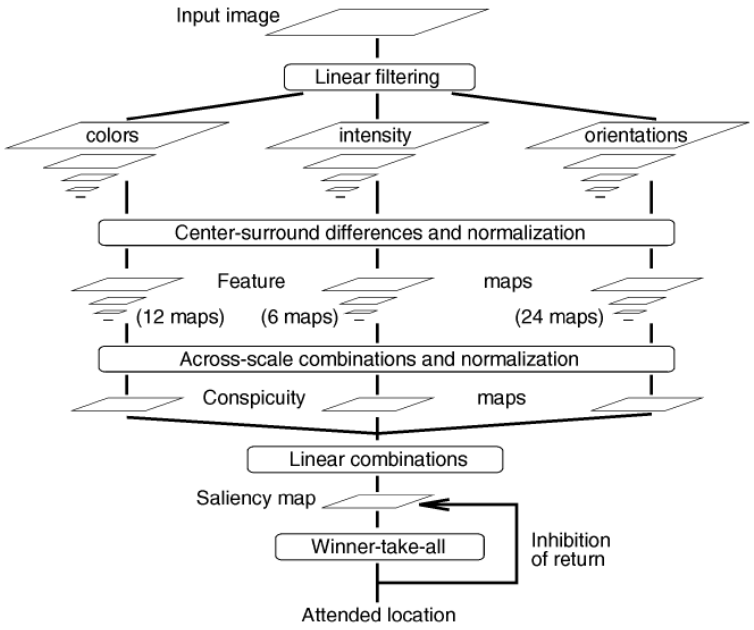

Figure 3.3: *Process of saliency map creation. Taken from [IKN98]*

the mass of needed computations. Several publications using saliency mapping already exist [LDC05], [SDL+05], [LDC06], proving the concept. The computational cost of generating the saliency map can be reduced by downsampling the input image. As an example for the performance gain achieved, we mention here the implementation using perception-aware rendering of [LDC05]. This implementation is able to compute a global illumination scene without a perceptable loss of details in OpenGL in just $2ms$, compared to $382sec$ without using the saliency map process.

Chapter 4

Implementation

If you optimize everything
you will always be unhappy

Donald E. Knuth

Our approach [ARCC] is implemented using C++ and OpenGL 2.0 [SWND05] [HB03], which allows for easy use of vendor extensions and supports indirect rendering.

The OpenGL pipeline consists merely of two big stages: the vertex and the fragment stage. The application provides vertex input in form of vertex arrays or display lists. This vertex data is then handled by the vertex processor which applies transformations. Then the vertex data is converted to raster graphics and forwarded to the fragment stage, which applies color and texture to the pixels.

The OpenGL shading language (GLSL) [Ros04] allows to modify the GL's fixed function pipeline (fig. 4.1) by replacing the vertex or/and the fragment stage with programmable shader objects. Our implementation solely operates on the fragment stage. We just have one vertex shader, loading an

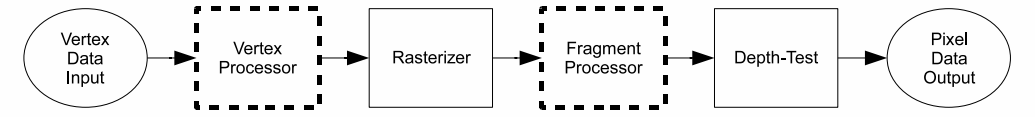

Figure 4.1: *Simplified view of the standard OpenGL pipeline*

orthographic projection, which creates a $(0,0)\times(1,1)$ projection matrix. This equivalence eases manipulation of pixel data. The GL_TEXTURE_WRAP mode is set to GL_CLAMP_TO_EDGE for all textures in this implementation.

Since version 2.0 it is no problem to use arbitrary sized textures by default, which makes it possible to use any rectangular region of pixels without further resizing (as long as it stays under the maximum size). OpenGL uses the normalized address space, i.e. that every texture is addressed by $[0,1]\times[0,1]$, independently from its size. Every pixel is referenced at its center, the origin of textures is at the lower left corner, see fig.4.2. To get the offset for a $width \times height$ sized image, we have to divide 1.0 by the size, having then $(1.0/width, 1.0/height)$. This offset is then used to address neighborhood pixels. We will explain that concept further down below.

Normally, rendered content targets the framebuffer. Indirect rendering allows to use so called render targets. One such a render target is the extension GL_FRAMEBUFFER_OBJECT (FBO). It allows a dedicated memory area residing on the GPU to be used as alternative "framebuffer", which can then be used (bound) as a texture for further processing. FBOs have more numerical formats than the default framebuffer. The default way in which OpenGL handles numerical data is fixed point, which is clamped to the interval [0.0,1.0]. This means, that it is not usable for HDR data, which lies in [0.0,10000.0]. Normalizing data is not an option, since we loose precision, see 2, therefore other techniques are needed. The solution is to use a floating point precision pixel format, like GL_RGBA16F_ARB and GL_RGBA32F_ARB, which represent 16 and 32 bit floating point precision.

There can be used more than one textures per FBO, this allows for iterative rendering, called ping-pong in case of two textures. The FBO can then use its own output as input for the next iteration, see alg.4.

4.1 Framework

Our system is easily attachable to existing solutions. The only input is a floating point color texture. We have chosen the GL_RGBA16F_ARB as the default format, since it has enough precision, while still being fast to pro-

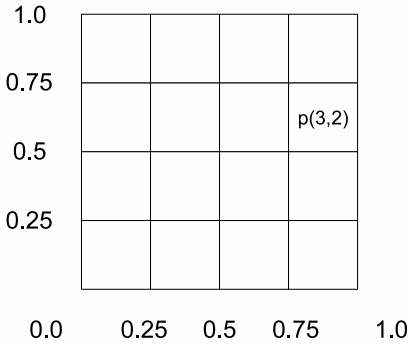

Figure 4.2: *Normalized texture coordinate system in OpenGL, for a* 4×4 *texture. The pixel p(3,2) is addressed by (0.875,0.625)*

Algorithm 3 "Ping-pong" rendering

Bind texture A of FBO as render target
Render scene
Unbind FBO
Swap FBO
Bind texture A of FBO as texture
Bind texture B of FBO as render target
Render scene
Unbind FBO
Swap FBO

cess. The basis of the implementation form the Shader and the Framebuffer Object class - as far as possible, we tried to follow OpenGL's naming conventions. Therefore having en-/disable and bind (for the Framebuffer Object) methods. The basic public methods of the Shader class are:

- enable
- disable
- setUniform

The enable and disable methods bind/unbind the object as the current shader to the OpenGL system. The method setUniform is polymorph and is able to set the uniform variables of the shader. The code is loaded by the Shader class' constructor. The methods of the Framebuffer Object are as follows:

- enable
- disable
- swap
- bindAsTexture

The en-/disable methods bind the framebuffer as current render target. As mentioned before, FBOs can also be double-buffered, which means that we need a mechanism to switch the texture we want to render to. This is realized with swap, while bindAsTexture, as the name implies, binds the current texture (the front texture for double buffered FBOs) to the OpenGL system.

The actual algorithm is implemented in one class - AshikhminHW. The class has the following main methods:

- enable

- disable
- processImage
- bindResultTex

The en-/disable methods have a slightly different interpretation than before. They (re-) set states, which control the stage, and selects which effects are applied. They also control enabling and disabling of early-Z optimization. The processImage is the "main" method, which calls the other needed (private) methods. The result, again a texture, is then bound with bindResultTex.

4.2 Overview

An overview of our implementation is shown in fig. 4.3. The diagram is divided in three parts (from left to right): The actual flow diagram, the used structures, and the pixel layout in the last column[1]. Every position of the flow diagram represents basically one stage of the Ashikhmin algorithm. The only memory intense data structures, our implementation uses, are one double-buffered FBO and an image pyramid, consisting of single-buffered FBOs.

First of all, the algorithm converts the input HDR RGB data to LUM, filling the red, green and blue channels with the HDR luminance and setting the alpha channel to 0. Then minimum and maximum of LUM as well as a Gauss filtered version and a approximated LUM average are computed with the pyramid method described in 4.4. If early-Z optimization is enabled, we compute the edgemap of the LUM to the z-buffer. The following step is computing the luminance adaption map L_a. This is done in an iterative fashion, using the ping-pong principle described earlier. The first pass computes the horizontal blur, we call $\overline{G_s}$ and $\overline{G_{2s}}$, followed by the second pass, which computes the vertical blur (resulting in fully blurred G_s and G_{2s}) including also the lc value. The alpha channel of each pixel is used as a switch. If $lc > threshold$, the alpha value is set from 0 to 1, fixing the pixel for the

[1]The 4 boxes of the pixel layout represent the red,green,blue and alpha channel

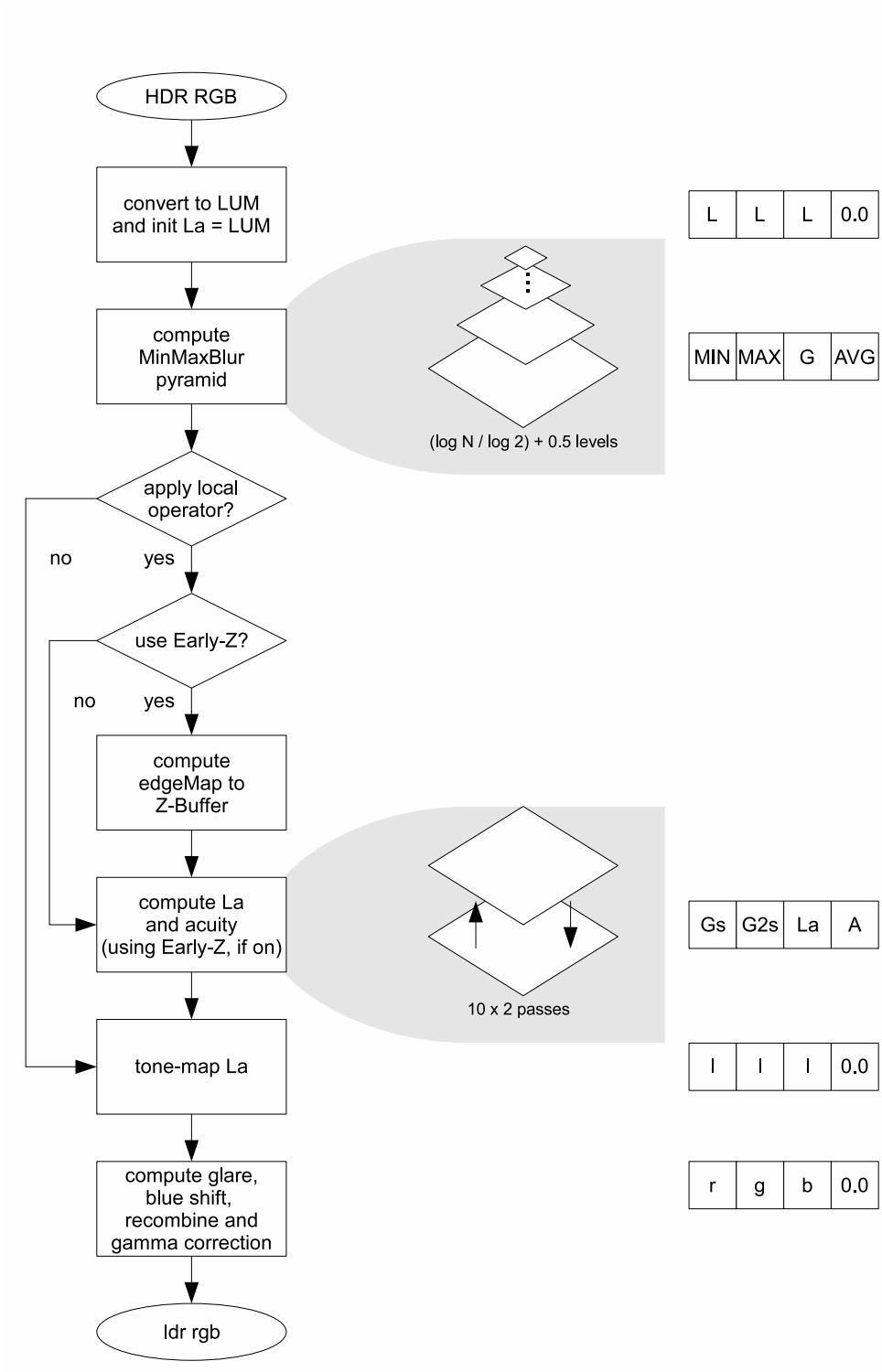

Figure 4.3: *Flow diagram of our implementation*

rest of iterations. After $NUMITERATIONS$ (default is set to 10) the L_a is tone-mapped and used to scale the HDR RGB image.

4.3 Accelerating Local Tone Mapping

For optimizing and therefore accelerating graphics processing it is needed to put as much workload from the CPU to the GPU. The communication between these two parts must be reduced to a minimum. The biggest performance drop during execution of the fragment stage is caused by texture reads.

The fact, that a lot of pixels are processed by the graphics card leads to the need of reducing this number. There are several ways to do this. One way is down-sampling the input image and process only this downsized version. Another solution would be to lower the number of iterations. Both lead to loss of quality. With the following optimizations we reduce the number of texture reads and processed fragments without noticeable degradation in terms of quality.

4.4 MinMax computation

The first thing we need for tone mapping is the minimum and the maximum of the input LUM. Reading back the HDR luminance texture to CPU memory and analyzing the data would be far too slow for real-time applications. The imaging subset of OpenGL, which implements basic image modification tasks like convolution, is not widely available. The most practical solution is to build up a so called image pyramid. An image pyramid is built upon the input image, which forms level 0. Every subsequent level is then a fourth size compared to the previous one (half width and half height). Image pyramids are also widely used for filtering implementing Gaussian or Laplacian convolution kernels. Examples can be found in [AAB+84] and [SKE06].

We compute the pyramid, by simply taking the next lower $pow2$ size of the width and height of the input image, e.g. $size = 400x300$ (level $i = 0$) we use 256x256 as our $1st(i = 1)$ level. We iterate over i, bisecting level i

until we get a 1×1 texture, as can be seen in fig.4.4.

By bisecting the texture, we compute the min and max of 4 pixels to 1 pixel of the following level. We use the red and green channel of the texture to store these two values, the blue channel holds a Gaussian-blurred version and the alpha channel stores an approximated average, needed later for the visual effects. We use a 3×3 filter kernel, fig.4.4, which would need 9 texture reads per pixel, if done in 1 pass. Because of the separability of Gaussian filters, it is normally implemented in a 2-pass fashion with an 1D-horizontal and vertical filter kernel, which then needs 6 texture reads (but 2 passes).

Figure 4.6 shows the process of computing both, for a 4×4 pixel region. On the left side you can see the 8 circles, which represent the neighborhood samples (linear sampled). By taking the min and max of the 4 inner sampling points for each level of the pyramid we end up with a 1×1 sized layer, holding the min and max in the red and green channel. The gauss value, which is kept in the blue channel, is computed by dividing the sum of these 8 sample points by 7, which equals $\frac{A+B+C+D+E+I+H+L+M+N+O+P+5*(F+G+J+K)}{28}$. The average value is just the linear sampled value of the central 4 pixels.

The quality of the Gaussian blur is enough for the visual effects, but not for the luminance adaption computation. Therefore it will be computed in a separate stage, see section 4.5.

4.5 Gaussian Filter

Since Ashikhmin's tone mapping algorithm [Ash02] and [RAM+07] needs 20 iterations to build up the luminance adaption map L_a, this is the most crucial part for optimizations. Basically, there are 2 possibilities of computing the Gaussian blurred layers: a cube or a pyramid. The cube means the same as the aforementioned pyramid, with the difference, that it just consists of equal sized layers. If we use the cube and 2D Gaussian filtering, we need $width * height * iterations * 9$ texture reads in 20 passes, which is a lot. If we separate the filter, we end up with $width * height * iterations * 6$ texture reads in 40 passes, which is still too much. A pyramid is a method of enormously reducing these numbers. Thereby needing only $O(2 * w * h) * iterations * 6$ texture reads and 40 passes.

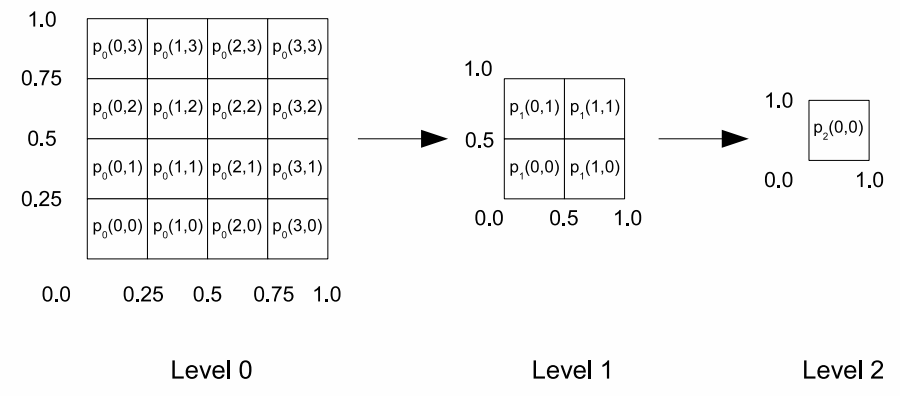

Figure 4.4: *Image pyramid in normalized texture space of OpenGL. Base level size* $= 4 \times 4$ *with NUMLEVELS=3*

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Figure 4.5: *Filter kernel for 3x3 Gaussian filtering (left), followed by the seperated horizontal (center) and vertical (right) base elements*

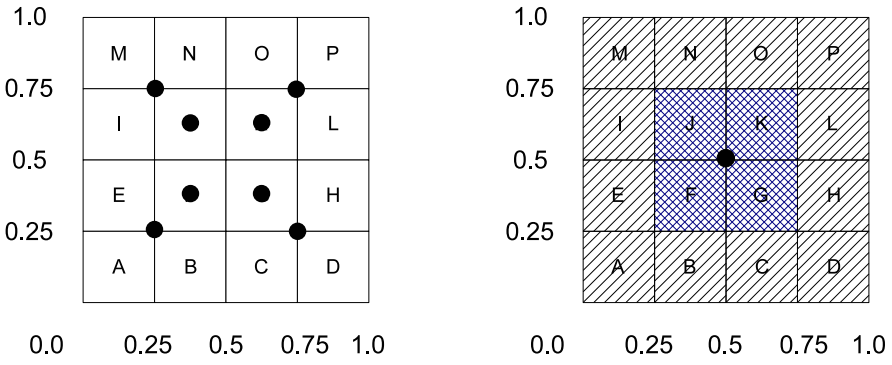

Figure 4.6: *For computing minmaxGaussAvg pyramid, we use GL_LINEAR filtering of OpenGL. On the left the sampling points are shown, on the right you can see the influences for the sampled pixel*

Since we want to keep quality, we take the cube approach. The next thing to change is the number of passes. It is possible to lower this, by using two Gaussian filters in parallel - the first one with radius s, the second one radius $2s$. In the following, we introduce an approach to iteratively compute Gaussian blur, with s and $2s$ in parallel. We also prove that this is equal to compute the filtering with a non-iterative approach.

We will now demonstrate the equality of the recursive Gaussian filter method and the Gaussian filter with growing filter size using Pascal's triangle as basis for the kernels. Gaussian filters itself are separable, i.e. that it is enough to just prove this for the one dimensional form of the filter.

i	Coefficients
0	1
1	1 1
2	1 2 1
3	1 3 3 1
4	1 4 6 4 1
5	1 5 10 10 5 1
6	1 6 15 20 15 6 1
...	...

Every line with the index $i = 2 * s$ can be taken as a Gaussian 1D filter, with radius s ((1), (1,2,1), ...).

s	$i = 2 * s$	Coefficients
0	0	1
1	2	1 2 1
2	4	1 4 6 4 1
3	6	1 6 15 20 15 6 1
...	...	...

We now want to prove that s-times execution of the recursive method with the base element (1,2,1) on a line of pixels is equal to a filter having index $2 * s$. First we compute the result by processing the pixel line twice with the base element (1,2,1)

$$\begin{aligned} A' &= ? + 2A + B \\ B' &= A + 2B + C \\ C' &= B + 2C + D \\ &... \\ H' &= G + 2H + I \\ I' &= H + 2I + ? \end{aligned}$$

which yields the following scanline: $A'B'C'D'E'F'G'H'I'$, while the second iteration looks like this:

$$\begin{aligned} A'' &= ? + 2A' + B' \\ B'' &= A' + 2B' + C' \\ C'' &= B' + 2C' + D' \\ &... \\ H'' &= G' + 2H' + I' \\ I'' &= H' + 2I' + ? \end{aligned}$$

which results in: $A''B''C''D''E''F''G''H''I''$. The ? symbolizes the question of which value has to be taken, if the filter runs over the border of the input image. This is solved by the GL_CLAMP_TO_EDGE mode, which automatically clamps the texture coordiantes to $[\frac{1}{2N}, 1 - \frac{1}{2N}]$. We compute pixel values for a sample line of pixels: $ABCDEFGHI$ with filter of $s = 2$ (i.e. $(1, 4, 6, 4, 1)$), resulting in

$$\begin{aligned} \tilde{A} &= ? + 6A + 4B + C \\ \tilde{B} &= ? + 4A + 6B + 4C + D \\ \tilde{C} &= A + 4B + 6C + 4D + E \\ \tilde{D} &= B + 4C + 6D + 4E + F \\ &... \\ \tilde{I} &= G + 4H + 6I + ? \end{aligned}$$

Which results in the pixel line $\tilde{A}\tilde{B}\tilde{C}\tilde{D}\tilde{E}\tilde{F}\tilde{G}\tilde{H}\tilde{I}$. We know that, starting with C" we have: $C'' = B' + 2C' + D' = (A + 2B + C) + 2 * (B + 2C + D) + (C + 2D + E) = A + 4B + 6C + 4D + E = \tilde{C}$.

Because $s = 1$ provides the same result for both (we use (1,2,1) in both cases), and is also valid for $s = 2$, we need to prove the way from $s \rightarrow s + 1$ holds for all s. Because of the binomial coefficient, where we can construct the coefficients for every $(x + y)^n$ by looking at Pascal's triangle at index n, we know that every $2 * nth$ element is constructible by taking the n-th power of the base element $(x + y)^2$:

n	$(x + y)^{2n}$	Coefficients
0	1	1
1	$(x + y)^2$	1 2 1
2	$(x + y)^4$	1 4 6 4 1
...	...	...

The iterated application of the base element (1,2,1) means nothing else than the above, which proves the concept.
Mathematical viewed, both methods are equal, but numerical errors can lead to very small differences. These differences are no problem for us.

4.6 Visual Attention

Because we concentrate only on one kind of feature, namely high contrast regions, with a high amount of detail we decided to not fully use the saliency map process. Instead we selected the edge computation process to be our only feature extractor for selecting important regions.

Since the luminance adaption method of Ashikhmin's algorithm is basically a multi-scale feature detection method, we just have to provide a first estimation to identify interesting regions in a fast and reliable way.

4.6.1 Edge Map Computation

First we know, that this regions have a high amount of luminance, compared to the rest of the image. Second, regions of interest are strong details which we also want to preserve. Therefore we conduct a simple thresholded (default is set to 0.1) edge detection with the Sobel operator (fig.4.6.1). The threshold avoids single pixels, which otherwise would prevent hierarchical-Z from being efficient4.7.3.

4.7 Selective Rendering

Selective rendering is a method to reduce the amount of computations per pixel. It is used in algorithms, which operate in the spatial domain, often in

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Figure 4.7: *Horizontal (left) and vertical (right) Sobel edge detection operator*

an iterative fashion. We call such pixels “marked“. The marking step can be done by aforementioned edge detection, saliency map, etc. There are several possibilites in OpenGL to implement selective rendering:

- Tiling
- Conditionals in GLSL
- Early-Z

4.7.1 Tiling

Tiling is a method of dividing the input image into several sub-images of equal size. Based upon a tile-based marking, the computational expensive procedure is applied on a per-tile basis. In its most basic form, the OpenGL viewport is tiled in small equal-sized rectangles and each of this rectangles is rendered based on its marking. Tile based rendering has two shortcomings. Since tiles consist of many pixels, it is not very coarse method of pre-selection. The other problem is, that as long as tile-based rendering is not supported by the hardware, it is far too slow for real-time applications.

4.7.2 Conditionals

Conditionals in the OpenGL shading language are supported only in the newer GPU generations. With conditionals it is able to implement a per-pixel based selcetive rendering. The criteria is either given as an input to the fragment input shader or it is computed on the fly. The implementation and performance of conditionals varies over a wide range. Since we wanted our implementation to be widely usable, we avoided the usage of conditionals completely.

4.7.3 Early-Z Culling and Hierarchical-Z

We use the edge detection method as the only feature recognitation, which is enough for our needs. We conducted several experiments related to the size of the edge-map. They showed, that the edge-detection pass itself is not

as costly that it would justify additional down- and up-sampling of the map. Due this, we directly compute the edge-map to the depth buffer, without any indirect rendering in between.

Early-Z and Hierarchical-Z are both technologies, developed for optimizing fragment throughput in connection with depth testing. Early-Z means relocation of the Z-Test before the fragment stage 4.8. This means, that the depth test can happen, *before* fragments are written to the depth buffer, avoiding expensive computations on pixels, which would later be discarded by their depth value. Therefore early-Z is not always enabled, only at the following conditions:

- Alpha and stencil tests are disabled
- Z-Buffer is not modified in fragment stage

Only if every condition holds, early-Z is enabled. Preferably it is used as a mask and replacement for conditionals [SI05]. Using early-Z consists of a setup pass (alg.4) and one or more following rendering passes. Basically, the setup process is allowed to do everything what's needed to compute the needed mask values into the depth buffer. Once the depth buffer is ready, one or more render passes follow. It is not allowed to write to the depth buffer, do alpha test nor execute stencil test functionality during these render passes, therefore we set glDepthMask(GL_FALSE).

We apply the early-Z principle to mask out fragments which are not perceived as important details. In our case the renderDataToDepthBuffer method is a shader which computes an edge-map (Sobel operator) of the input LUM. The resulting values are thresholded ($defaultedgeThreshold = 1.0$), which also eliminates isolated pixels. Singular pixels would make hierarchical-Z ineffective. Hierarchical-Z (fig.4.9) means that the depth-test is not conducted on a per-pixel basis, but on bigger regions. The Z-values are therefore

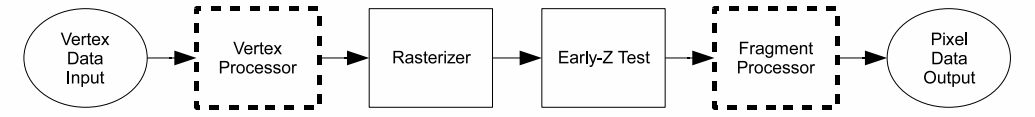

Figure 4.8: *Simplified view of the OpenGL pipeline with early-Z enabled*

Algorithm 4 Setup pass of Early-Z in OpenGL

```
glClearDepth( 1.0 );
glDepthMask( GL_TRUE );
glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );
glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT );
glLoadIdentity();
renderDataToDepthBuffer();
glColorMask( GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE );
glDepthMask( GL_FALSE );
```

propagated and subsumed in a pixel group. This example has 4×4 sized regions (which is rather small). For the lower-left rectangle (region) we know, that there are no blocking pixels, we don't have to examine no more pixels contained in that region, and can go on to the next one.

4.8 Visual Effects

As mentioned in section 2.4, realtime tone mapping is not complete without implementing basic visual effects. Our implementation is based upon [KMS05], which describes the biophysical basics of these effects in detail. Since they applied these effects to a similar local tone mapping operator [RSSF02] it is easy adaptable to our needs. The rest of this section describes, which effects were implemented at which stage of our TMO implementation.

4.8.1 Temporal Luminance Adaption

Simulating luminance adaption is a primary thing and can eased by interpolating the actual luminance extrema. Equation 4.1 shows the version from [KMS05]. As stated in [GWWH03] luminance adaption is more important to look good, than to be physically correct.

We apply the adaption to L_{min} and L_{max}, therfore influencing the whole scene (at the recombination). The L terms are 2d vectors, holding the minimum and the maximum, $delta_t$ represents the time the last frame took to render. The τ is depending on the simulated sensors' adaption time, see

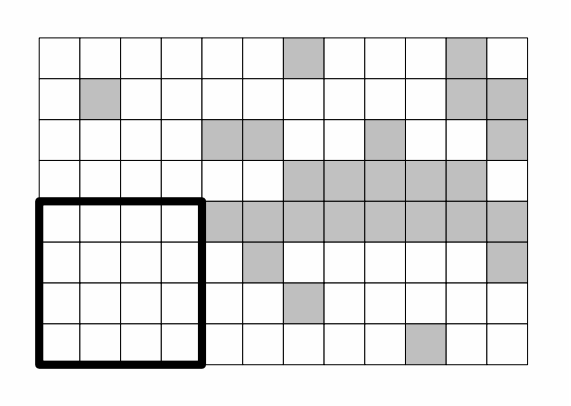

Figure 4.9: *Depiction of hierarchical-Z*

section 2.

$$L_{newMinMax} = L_{oldMinMax} + (L_{MinMax} - L_{oldMinMax}) * (1.0 - e^{\frac{-delta_t}{\tau}})\tilde{L} \quad (4.1)$$

4.8.2 Loss of Acuity

The computation of Loss of Acuity has to be done on a per-pixel basis. The right level of convolution is selected based on the input L value of the pixel, see fig. 4.10. This diagram is based on measurements of the HVS and taken from [KMS05]. The selection is thereby easily implementable. We recompute the numerical values by the e function and use them to index an array of values.

We implement this in the luminance adaption computation stage, where per-pixel information is available. We already have one stop criteria, lc, and add the value of the diagram as an additional one. In each iteration we pass the current range to be examined to the fragment shader, which computes the adaption value. If the original L value of the pixel lies in the range, we fix it, otherwise it will be further iterated.

The resulting luminance adaption value, which is basically the Gaussian filter value with radius s from (s is the current iteration) is fixed then. This allows for an easy integration of acuity in the adaption computation, without any further modifications (except one additional texture read and one

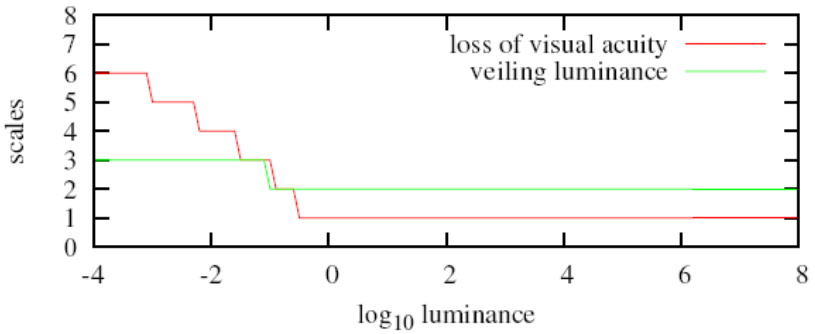

Figure 4.10: *Graph showing which level of the Gaussian pyramid has to be selected for simulating glare and loss of acuity. Taken from [KMS05]*

computation).

4.8.3 Glare

Glare describes the effect of veiling luminance, caused by strong streaks of light, which is scattered in the human eye. The selection of the glare-map, which is a blurred version of the input LUM, is based on a global value. Krawczyk proposes the relative luminance value of Reinhard's algorithm, which can be approximated by the log avg LUM. We approximate the avg LUM by using the avg value in the alpha channel of the minmax pyramid, see fig. 4.3. When the right glare-map is selected, the actual glare is computed by

$$glare = L_{glareMap} * (1.0 - \frac{0.9}{0.9 + L_{glareMap}}) \tag{4.2}$$

4.9 Recombination

Basically the recombination is done as shown in [Ash02]. While the acuity modifies the L_a, the glare map modifes the hdr luminance. Therefore we have to compute the glare-map and the resulting glare after the L_a computation stage in the recombination.

$$\begin{pmatrix} R_L \\ G_L \\ B_L \end{pmatrix} = \begin{pmatrix} R \\ G \\ B \end{pmatrix} * \frac{L * (1 - \sigma(Y))}{Y} + \begin{pmatrix} 1.05 \\ 0.97 \\ 1.27 \end{pmatrix} * L * \sigma(Y) \tag{4.3}$$

Scotopic vision describes the perception under bad lighting conditions (e.g. dark rooms). Low light means, that the HVS has to concentrate on recognizing contours instead of details. Since rods are specialized in "collecting and using all light possible", they are not able to extract detailed features or color - therefore a "blue shift" occurs, which compresses the lower end of the visible spectrum (red) towards the upper end (blue). The model of [KMS05] does not handle the two types of sensors separately. Instead they are combined using eq.4.3, which also takes care of "greying-out" chromaticity.

Chapter 5

Results

Computers are useless
They can only give you answers

Pablo Picasso

We present the results in two sections. A quality and a performance related section. For the local tone mapping algorithm in both cases the same parameters are used. We compute the results with 10 iterations with threshold set to 0.5 and gamma of 2.4. Exceptions are the global part, which is computed with 0 iterations. The default value for the edge threshold used by the selective tone mapper is set to 0.1.

5.1 Quality

We compare the output of several configurations of our implementation:

- Software Local
- GPU-based Global-only
- GPU-based Local
- GPU-based Selective with saliency map input
- GPU-based Selective with edge map input

Our reference software implementation also uses Pascal's triangle as base elements, but instead of iteratively applying the two elements $(1, 2, 1)$ and $(1, 4, 6, 4, 1)$ the kernels are taken by their appropriate radius. This is done to prove the equality of the concept.

Our GPU-based implementation is able, beside computing the edge-map as depth mask, to also take other sources for the depth buffer input. We comptued the saliency map with the software by [LDC06] and used it as an input to our implementation to show the equality of the two concepts.

The numbers presented here are taken with the visible difference predictor (vdp) [Man]. This software shows, by comparing two images, how many pixels would be perceived as different by a human observer.

Visual effects are not applied in this part of the comparison, since they are a real-time related topic. Comparison of quality with effects applied can be seen in section 5.1.1.

In table 5.1, we compare the different configurations. The most interesting result here is the Sel. TM vs Loc. TM which shows, that, though the operator is fast, it still produces results of high quality. We know, that using just the edge map computation part of the saliency process is not enough to retreive a full saliency map, we think that the results show, that especially in spatial based tone mapping operators it still provides good results.

HDR Image	Glo vs Loc	Sel vs Loc	Sal vs Loc
Belgium	1.63	0.21	0.34
FogMap	0.26	0.08	0.21
Memorial	5.4	2.4	3.6
Nave	3.7	1.47	2.81

Table 5.1: *Percentage of difference given for Global vs Local, Selective vs Local, Saliency vs Local. In this case the edge threshold is set to 0.1*

Table 5.2 even more justifies the observation we stated before. The edge map approach performs better in this type of operator in terms of quality than the saliency-based one.

HDR Image	Sel vs CPU Loc	Sal vs CPU Loc
Belgium	0.22	0.35
FogMap	0.08	0.20
Memorial	2.4	3.7
Nave	1.86	3.2

Table 5.2: *Percentage of difference given for Selective vs CPU Local and Saliency vs CPULocal. In this case the threshold is 0.1*

5.1.1 Visual Effects

The visual effects of loss of acuity, blue shift and glare can be applied separately. We present the result of applying all effects in fig. 5.1. The figure shows two details of the rosette image, with visual effects applied. The first row of both details show the result using selective tone-mapping, while the second row shows the result using local tone-mapping.

5.2 Real-Time Performance

All real-time experiments were conducted on an AMD Athlon XP 2500+ equipped with 1.83GHz and 1GB of main memory under MS Windows XP. The graphics card used was a nVidia GeForce 6800 with 128MB of on-board memory. This represents a common setup having mid-range characteristics.

The real-time performance figures in terms of frames per seconds (fps) we achieved with our implementation are listed in table 5.3. Table 5.4 and table 5.4 show the measurements of our implementation used as a backend for a simple cube-mapped teapot scene.

The stills reflect the possible use case of an HDR image editing application, with a low number of modifications on the whole image per time. Since the figures can be optimized by the GPU through caching, the teapot scene proves that the concept is usable.

As you can see for the stills, we gain a performance boost of around 3, when comparing local-tonemapping with and without selective rendering. You can also see, that equi-sized images like nave and rosette both perform equal, as long as it comes to selective tone-mapping. This shows that the

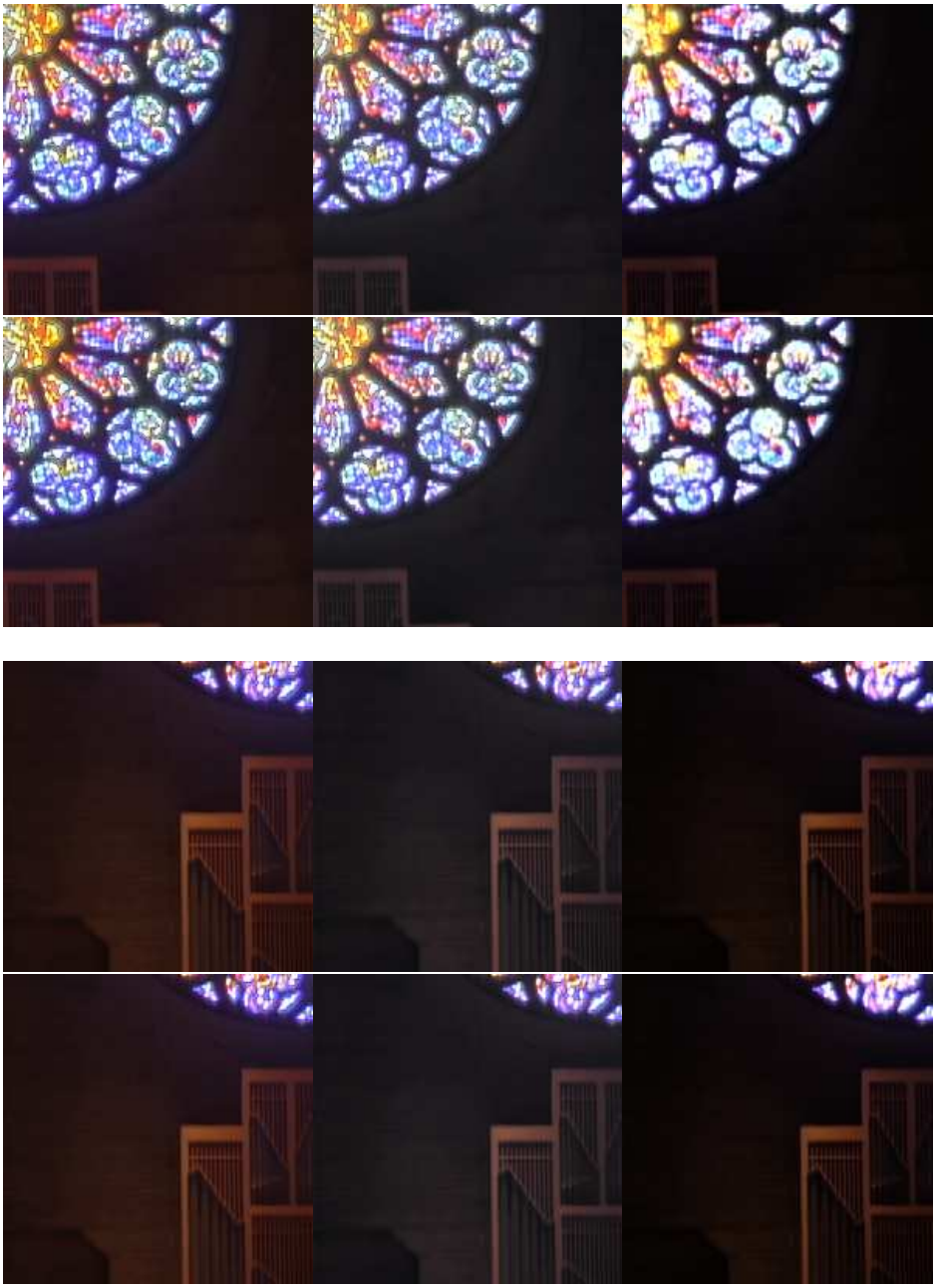

Figure 5.1: *Rosette with selective (first row) and local (second row) tone mapping, with visual effects applied seperately. The two rows show details, one taken from a high-lighted and one from a dark region. They are (from left to right): tone mapped only, loss of acuity including blue shift applied, the effect of glare*

performance rises noticeable when reducing the number of pixels being processed.

HDR Image	Size	Global	Local	Selective	$\frac{Selective}{Local}$
16RPP	900x900	63	12	44	3.76
Aeroporto	1024x705	71	13	40	3.08
Belgium	1025x769	66	13	30	2.31
Desk	644x874	95	18	31	1.7
Fogmap	751x1130	63	12	37	3.08
Lamp	400x300	395	78	248	3.18
Memorial	512x768	136	25	82	3.28
Nave	720x480	147	29	94	3.24
Rosette	720x480	147	29	78	2.69
Stillife	1240x846	49	10	26	2.6

Table 5.3: *Frames per second, applying our GPU-based implementation to stills*

The viewport sizes we used for the teapot scene are standard screen resolutions. As you can see in table 5.4, the ratio in the last column shows an increase, which means nothing else, then the computation with selective tone-mapping is more resident to higher resolutions.

Size	TM disabled	Global	Local	Selective	$\frac{Selective}{Local}$
320x240	369	226	83	141	1.70
640x480	269	111	28	78	2.79
800x600	239	80	19	55	2.89
1024x768	206	55	12	37	3.08

Table 5.4: *Realtime performance of applying our operator using selective rendering to a simple cube-mapped teapot scene*

Table 5.5 subsumes results of different edge detection thresholds. The lower the edge detection threshold, the less pixels are discarded during the iterative process. The results show, that it is not needed to raise the edge threshold for better performance. We found out, that 0.1 is a good value to go.

Size	0.1	0.2	0.5	1.0	5.0	10.0	20.0
320x240	141	145	145	145	146	146	146
640x480	76	79	82	82	84	85	85
800x600	54	58	59	59	60	60	60
1024x768	37	38	40	40	40	40	40

Table 5.5: *Realtime performance of applying our operator using selective rendering with varying edge threshold to a simple cube-mapped teapot scene with varying edge-threshold*

Showing results is nothing without showing interesting images. Following is a small gallery presenting results of our implementation.

Figure 5.2: *Images generated with selective tone mapper*

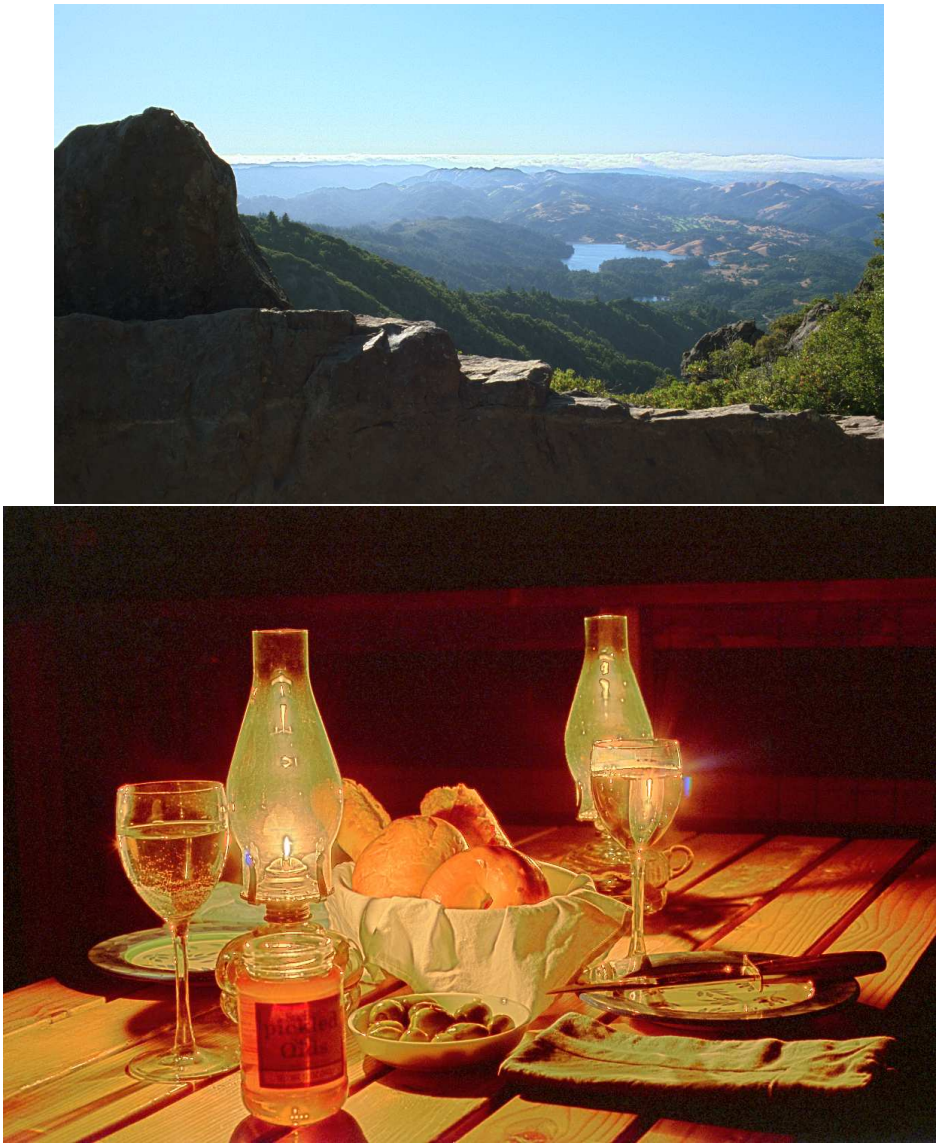

Figure 5.3: *Images generated with selective tone mapper*

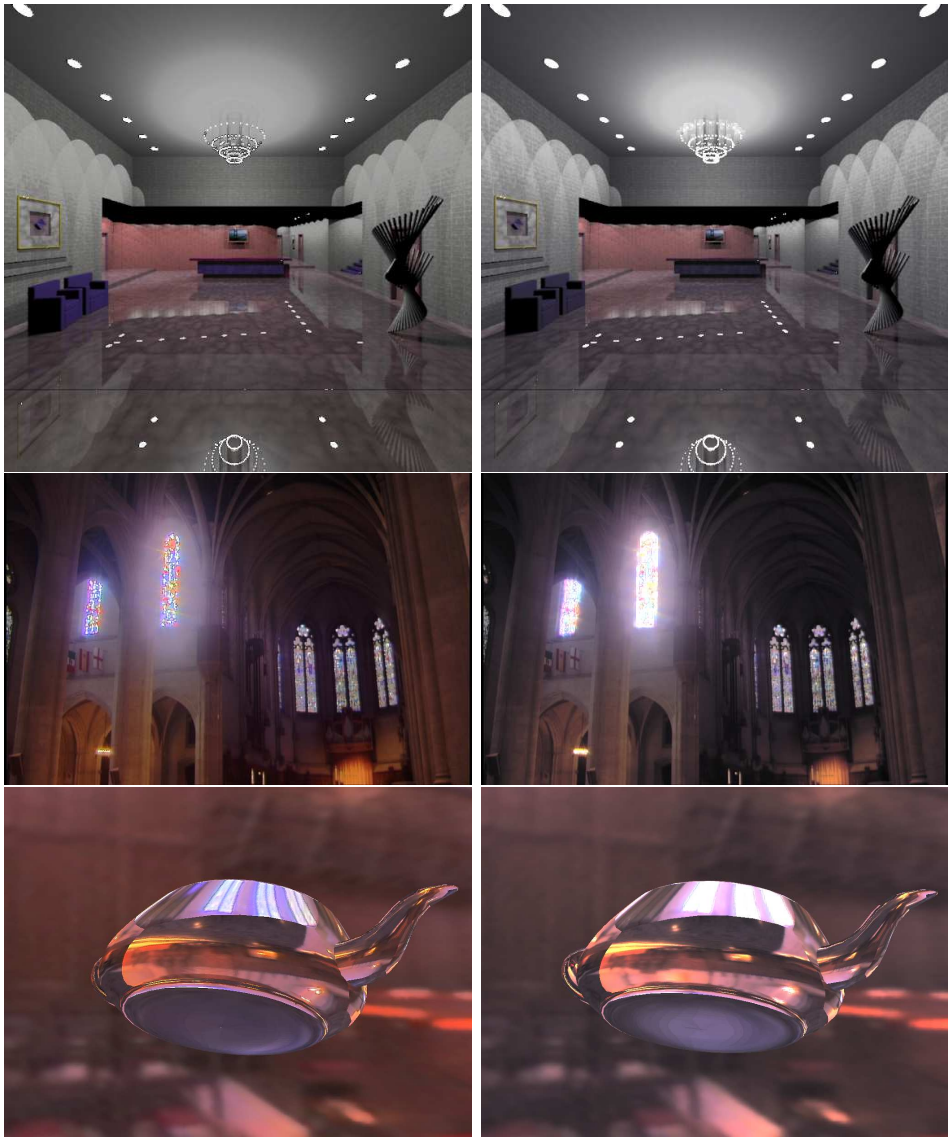

Figure 5.4: *Images generated with selective tone mapper (left) and visual effects applied additionally (right). The last row shows a shot from our real-time demo, used to do the measurements from above*

Chapter 6

Summary

In three words
I can sum up everything
I've learned about life
It goes on

Robert Frost

In this thesis we presented optimizations for an existing local tone mapping operator. We shortly described the fundamental concepts our work is based on: the human visual system, and its strategies of perceiving its surroundings. We explained the basic concepts of high dynamic range imaging, the acquisition of HDR data, as well as the tone mapping process. We then concentrated on the problem of tone-mapping, also in connection with real-time applications. The introduction was closed by a short discussion of visual attention in relation to optimizing throughput of the rendering process.
Based on related publications we tried to devise a model of the saliency-map process, which is also usable under real-time conditions. We chose to only use the edge detection stage as the process to build up a map, which represents the important regions.
We combined this alongside other optimizations into a C++ framework, based on OpenGL 2.0 with fragment shaders. This framework is easily attachable to existing rendering engines, which provide HDR output in the form of a floating point texture. This texture is then used as input to our framework. The implementation is fully controllable by enabling and disabling

states. It uses the combined Gauss convolution and the early-Z optimization to accelerate the pixel throughput of the luminance adaption computation. We also implemented the visual effects like temporal luminance adaption, glare, scotopic vision and loss of acuity.
The results we generated show that it is possible to gain real-time performance, while still preserving quality of the unmodified local operator.

6.1 Drawbacks

Our model roots only on saliency based mapping. We only use edge detection as a feature extraction process. Though the procedure produces good results in terms of quality and execution speed, it is not clear, if we miss regions which would be otherwise judged as salient by the observer.

6.2 Conclusion

By using perceptual based rendering, we proved that it is possible to accelerate local tone mapping operators, without any noticeable loss of detail and quality. Most of the optimizations we applied are also applicable to other spatial oriented tone mapping algorithms. We present a well-performing implementation of an existing broadly usable tone mapping operator, by combining passes and reducing the number of texture reads to a minimum. Comparing the ratios of selective vs. local the figures prove the good scalability to changing viewport sizes.

6.3 Future Work

Our model can be easily extended. Maybe finding a way of directly extracting the information for important regions from the local tone mapping algorithm would improve the performance even more, since on the one hand we reduce further by one pass and on the other hand have a more precise definition which pixels should be processed with the local operator and which can by discarded. Computing the edge-map not for every frame, but somehow

use information from previous frames, could be an additional modification, increasing the performance.

Chapter 7

Appendix A - Abbrevations

HVS	Human Visual System
TM	Tone Mapping
TMO	Tone Mapping Operator
HDR	High Dynamic Range
ldr	Low Dynamic Range
RGB	HDR rgb image
LUM	HDR luminance image
rgb	ldr rgb image
lum	ldr luminance image
L	HDR luminance value
l	ldr luminance value
FBO	FramebufferObject
$Z - Buffer$	Depth Buffer

List of Figures

List of Tables

Bibliography

[AAB+84] Edward H. Adelson, C. H. Anderson, J. R. Bergen, Peter J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA Engineer*, 29(6), 1984.

[Adaed] Ansel Adams. The negative, 1995 (republished).

[ARCC] Alessandro Artusi, Benjamin Roch, Alan Chalmers, and Yiorgos Chrysanthou. Selective tone mapper, application no. gb0709392.5. Technical report.

[Ash02] Michael Ashikhmin. A tone mapping algorithm for high contrast images. 2002.

[cvr] Cvrl color & vision database. `http://www.cvrl.org`.

[DD02] Fredo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. 2002.

[FPSG] James Ferwerda, Sumanta Pattanaik, Peter Shirley, and Donald P. Greenberg. A model of visual adaption for realistic image synthesis.

[GC84] G.S.Miller and C.R.Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. 1984.

[GWWH03] Nolan Goodnight, Rui Wang, Cliff Woolley, and Greg Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. 2003.

[HB03] Donald Hearn and M. Pauline Baker. *Computer Graphics with OpenGL*. Prentice Hall International, 3rd edition, 2003.

[hdr] Hdrshop. `http://gl.ict.usc.edu/HDRShop`.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. 1998.

[KMS05] Gregorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Perceptual effects in real-time tone mapping. 2005.

[LDC05] Peter Longhurst, Kurt Debattista, and Alan Chalmers. Snapshot: A rapid technique for driving selective global illumination renderer. 2005.

[LDC06] Peter Longhurst, Kurt Debattista, and Alan Chalmers. A gpu based saliency map for high-fidelity selective rendering. 2006.

[LW02] Raanan Fattal Dani Lischinski and Michael Werman. Gradient domain high dynamic range compression. 2002.

[Man] Rafal Mantiuk. Visible difference predictor. `http://www.mpi-inf.mpg.de/resources/hdr/vdp`.

[nas] Nasa moon thumbnails. `http://nssdc.gsfc.nasa.gov/imgcat/thumbnail_pages/moon_thumbnails.html`.

[Poya] Charles Poynton. Frequently asked questions about color. `http://www.poynton.com/PDFs/ColorFAQ.pdf`.

[Poyb] Charles Poynton. Frequently asked questions about gamma. `http://www.poynton.com/PDFs/GammaFAQ.pdf`.

[RAM+07] Benjamin Roch, Alessandro Artusi, Despina Michael, Yiorgos Chrysanthou, and Alan Chalmers. Interactive local tone mapping operator with the support of graphics hardware. 2007.

[Ros04] Randi J. Rost. *OpenGL Shading Language*. Addison Wesley, 2004.

[RSSF02] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. 2002.

[RWPD06] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging*. Morgan Kaufmann, 2006.

[SDL+05] Veronica Sundstedt, Kurt Debattista, Peter Longhurst, Alan Chalmers, and Tom Troscianko. Visual attention for efficient high-fidelity graphics. 2005.

[SI05] Pedro V. Sander and John Isidoro. Explicit early-z culling and dynamic flow control on graphics hardware, 2005.

[SKE06] Magnus Strengert, Martin Kraus, and Thomas Ertl. Pyramid methods in gpu-based image processing. 2006.

[SWND05] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison Wesley, 5th edition, 2005.

[TT99] Jack Tumblin and Greg Turk. LCIS: A boundary hierarchy for detail-preserving contrast reduction. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 83–90, Los Angeles, 1999. Addison Wesley Longman.

[War] Greg Ward. Radiance system. `http://radsite.lbl.gov/radiance/`.

[War92] Greg Ward. *Real Pixels*. 1992.

[wik] Wikipedia. `http://www.wikipedia.org`.