UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCOS PAULO BERTELI SLOMP

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

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Manuel Menezes de Oliveira Neto
Advisor

Porto Alegre, March 2008

CIP – CATALOGING-IN-PUBLICATION

Berteli Slomp, Marcos Paulo

Real-time Photographic Local Tone Reproduction Using Summed-Area Tables / Marcos Paulo Berteli Slomp. – Porto Alegre: PPGC da UFRGS, 2008.

108 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Advisor: Manuel Menezes de Oliveira Neto.

1. Realistic image synthesis. 2. Real-time rendering. 3. High dynamic range imaging. 4. Tone-mapping. 5. Brightness perception model. 6. Prefix sum. 7. Summed-area tables. I. de Oliveira Neto, Manuel Menezes. II. Título.

“Usai moderação depois do triunfo.
O menor insulto às pessoas (...) será uma mancha em vossa glória.”
— General Bento Gonçalves

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

CUDA	NVIDIA Compute Unified Device Architecture
GPU	Graphics Processor Unit
LDR	Low Dynamic Range
HDR	High Dynamic Range
HVS	Human Visual System
IBL	Image-Based Lighting
IDE	Integrated Development Environment
NPR	Non-Photorealistic Rendering
SAT	Summed-Area Table
SDK	Software Development Kit
TMO	Tone-Mapping Operator
cd/m^2	Candela per Square Metre, photometric measurement of luminance

LIST OF FIGURES

ABSTRACT

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

Keywords: Realistic image synthesis, Real-time rendering, High dynamic range imaging, Tone-mapping, Brightness perception model, Prefix sum, Summed-area tables.

Reprodução Fotográfica Local de Tons em Tempo Real Usando Tabelas de Áreas Acumuladas

RESUMO

A síntese de imagens com alta faixa dinâmica é uma prática cada vez mais comum em computação gráfica. O desafio consiste em relacionar o grande conjunto de intensidades da imagem sintetizada com um sub-conjunto muito inferior suportado por um dispositivo de exibição, evitando a perda de detalhes contrastivos. Os operadores locais de reprodução de tons (*local tone-mapping operators*) são capazes de realizar tal compressão, adaptando o nível de luminância de cada pixel com respeito à sua vizinhança. Embora produzam resultados significativamente superiores aos operadores globais, o custo computacional é consideravelmente maior, o que vem impedindo sua utilização em aplicações em tempo real. Este trabalho apresenta uma técnica para aproximar o operador fotográfico local de reprodução de tons. Todas as etapas da técnica são implementadas em GPU, adequando-se ao cenário de aplicações em tempo real, sendo significativamente mais rápida que implementações existentes e produzindo resultados semelhantes. A abordagem é baseada no uso de tabelas de áreas acumuladas (*summed-area tables*) para acelerar a convolução das vizinhanças, usando filtros da média (*box-filter*), proporcionando uma solução elegante para aplicações que utilizam imagens em alta faixa dinâmica e que necessitam de performance sem comprometer a qualidade da imagem sintetizada. Uma investigação sobre algoritmos para a geração de somatórios pré-fixados (*prefix sum*) e uma possível melhoria para um deles também são apresentada

Palavras-chave: Síntese de imagens realísticas, Renderização em tempo-real, Imagens em ampla faixa dinâmica, Reprodução de tonalidades, Modelo de percepção de brilho, Somatórios pré-fixados, Tabelas de áreas acumuladas.

1 INTRODUCTION

One of the most relevant subjects in computer graphics is realistic image synthesis, leading an observer to a similar visual experience when faced with a real world scene and a computer-generated image. Such class of images are widely used in the entertainment industry (movies and games), industrial design, architecture, engineering, publicity, geology, to cite a few. One of the critical parts of realistic image synthesis is the proper simulation of the interaction of light with the surfaces and materials that constitute a synthetic scene. For this purpose, an accurate representation of the light intensities becomes necessary.

Such light intensities are commonly measured as *luminances*, which can be understood as the amount of light that arrives (or leaves) a particular area in a given direction. Thus, the luminance of the sunset is much higher than the one of the moonlight. Such a distance between the largest and lowest luminance is referred as *dynamic range*. Additionally, some details that are visible during the sunset might not be perceptible at moonlight. The human visual system (HVS) is unable to *instantly* distinguish all the luminances (dynamic range) of the real world. Instead, it tends to increasingly discern additional details, adapting from one lighting condition to another.

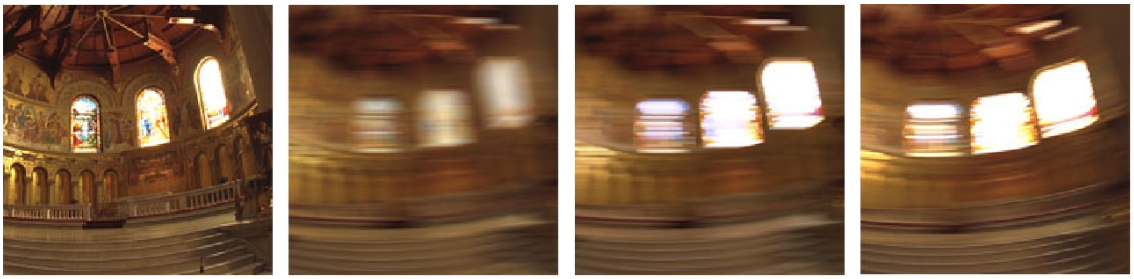

Figure 1.1: Motion blur in low and high dynamic range images. From left to right: a scanned photograph; the resulting simulation of motion blur using a low dynamic range image; the same motion blur applied to a high dynamic range image; motion blur produced using a real camera. Source: (DEBEVEC; MALIK, 1997).

While many devices allow to capture the large dynamic range of a real world scene, most current display devices have a fixed range of intensities for which they can effectively emit light. For this reason, a common practice is to represent images using only a small range of intensities (*e.g.* 8-bit integers), so that they respect the limitations of the target device. Such a process may imply in a huge loss of information. For realistic rendering, these encodings need to be replaced by others that are more suitable to accommodate the results of proportional physically-accurate intensities (*e.g.* 32-bit floats). Such

class of images is then referred as *high dynamic range* (HDR) images. This recalls to the initial problem: how to display HDR content in a device with limited dynamic range? The focus of this thesis is on how to provide a satisfactory answer to this question in real-time. Figure 1.1 shows the importance of HDR in realistic rendering.

The proper presentation of HDR content requires that the display devices support the dynamic range of the images. Otherwise, the resulting pictures will look either underexposed or overexposed, as can be seen in Figure 1.2. Current HDR displays are very expensive and still have a narrower dynamic range when compared to the real world (SEETZEN; WHITEHEAD; WARD, 2003; SEETZEN et al., 2004; Bit-Tech, 2008; Toshiba Corporation; Canon Inc, 2008; Dolby Laboratories, 2008). Compressing such large luminance ranges so that they fit into the much narrower ones supported by display devices, in a way that the resulting images are perceptually equivalent to the original scenes is known as *digital tone reproduction*, also known as *tone-mapping* (TUMBLIN; RUSHMEIER, 1993; REINHARD et al., 2006). For a illustrative comparison of some existing operators, refer to Figure 1.3.

Figure 1.2: A HDR image displayed in three ways: without any processing (left) is not suitable for display, as almost all details are lost; the use of gamma correction (middle) enhances the image quality, but a tone-mapping operator (right) can do much better. The memorial church image is courtesy of Paul Debevec.

Digital tone-mapping operators can be classified as *global* (spatially uniform), or *local* (spatially varying) (REINHARD et al., 2006). Global operators use the same parameters to process all the pixels of the image (TUMBLIN; RUSHMEIER, 1993; SCHLICK, 1994; WARD, 1994a; FERWERDA et al., 1996; WARD LARSON; RUSHMEIER; PIATKO, 1997; REINHARD et al., 2002; DRAGO et al., 2003; REINHARD; DEVLIN, 2005). In contrast, local operators try to find an optimal set of parameters for each pixel individually, considering a variable-size neighborhood around it (CHIU et al., 1993; PATTANAIK et al., 1998; ASHIKHMIN, 2002; FATTAL; LISCHINSKI; WERMAN, 2002; REINHARD et al., 2002; PATTANAIK; YEE, 2002; DURAND; DORSEY, 2002; YEE; PATTANAIK, 2003). For images with large dynamic ranges, local operators tend to produce significantly better results in terms of contrast preservation. This comes from the fact that the amount of compression is locally adapted to the different regions of the input

images. For this reason, local tone-mapping operators are computationally much more expensive than global ones. An illustrative comparisons between a global and a local operator is provided in Figure 1.4.

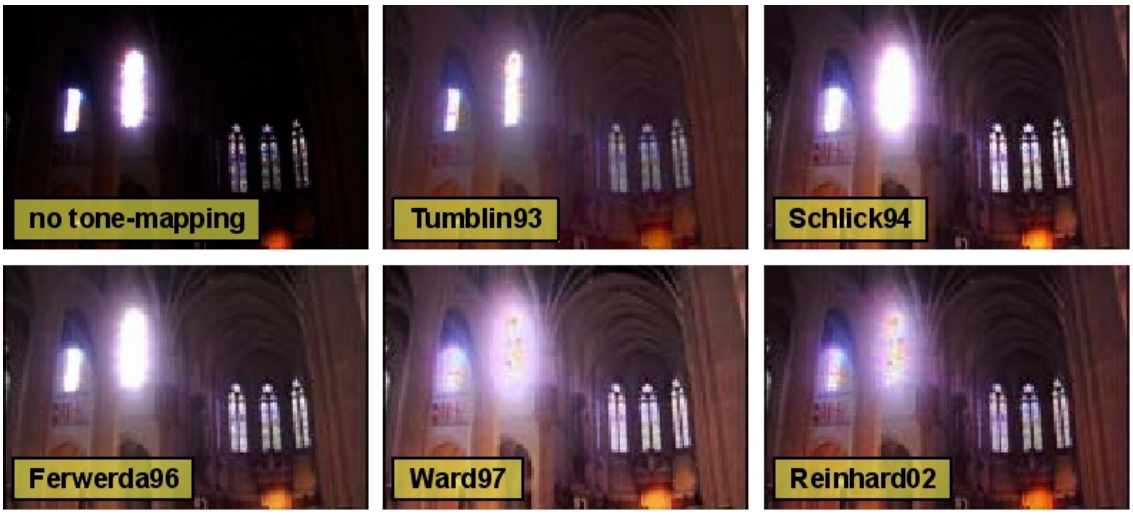

Figure 1.3: Example of a high dynamic range image (12 orders of magnitude) processed with various tone-mapping operators. Source: (REINHARD et al., 2002).

The photographic tone reproduction operator proposed by Reinhard *et al.* produces attractive results and it is probably the most balanced operator available, due to its simplicity, automacity, robustness, and perceptually-driven approach (REINHARD et al., 2002). Their method comprises both a global and a local operator (Figure 1.4). The former is fast, but potentially looses relevant contrastive detail. The later compresses an image's dynamic range, by using the average luminance around each pixel at different scales. To estimate these averages, Reinhard *et al.* used a brightness perception model presented by Blommaert and Martens, which is essentially a set of differences of Gaussian-filtered luminance images (BLOMMAERT; MARTENS, 1990). Unfortunately, the convolution with Gaussian kernels of various scales is an expensive operation.

In practice, interactive applications currently limit themselves to global operators (an observation that is valid to all tone-mapping operators and not limited to Reinhard *et al.*'s). An exception is the work of Goodnight *et al.* (GOODNIGHT et al., 2003), which efficiently implemented the 2D Gaussian convolution using separable 1D kernels on the GPU. However, despite all their optimization efforts, their technique can handle only a few scales of the original operator to run at interactive rates. A more efficient approach to approximate the convolutions was presented by Krawczyk *et al.*, but it introduces noticeable halo artifacts in the resulting image (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005). Figure 1.5 compares the result produced by Krawczyk *et al.*'s technique and the one obtained with the approach proposed in this thesis.

This document presents a novel GPU-based technique for performing Reinhard *et al.*'s photographic local tone reproduction in real time. The approach uses a summed-area table (CROW, 1984) to efficiently evaluate an approximation of Blommaert's and Martens's brightness perception model (BLOMMAERT; MARTENS, 1990), which is the most expensive part of the operator. The proposed technique produces results that are comparable to the original local operator (see Figure 1.5), but is significantly faster than previous GPU-based attempts. The approach is based on the following key observations:

- Local contrast can be estimated from the largest isoluminant region around each pixel. Identifying such regions is usually performed using differences of Gaussian-filtered luminance images which are then compared with a specified threshold. Similar results can be obtained using box-filtered differences and a lower threshold.

- Convolution with a box-filter is an average over a rectangular region, and it can be efficiently performed using summed-area tables for a fraction of the cost of a Gaussian convolution.

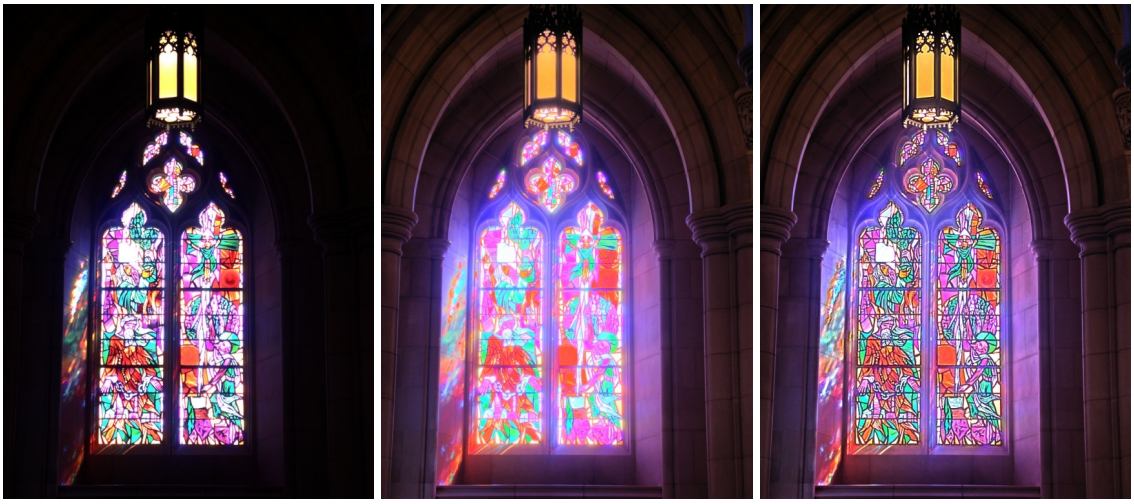

Figure 1.4: A comparison between the global and the local photographic operators introduced by Reinhard *et al.* (REINHARD et al., 2002). The original image (left) is submitted to the global operator (middle). Since many perceptive details are lost, its local variant (right) preserves contrastive details. Cathedral glass image courtesy of Greg Ward.

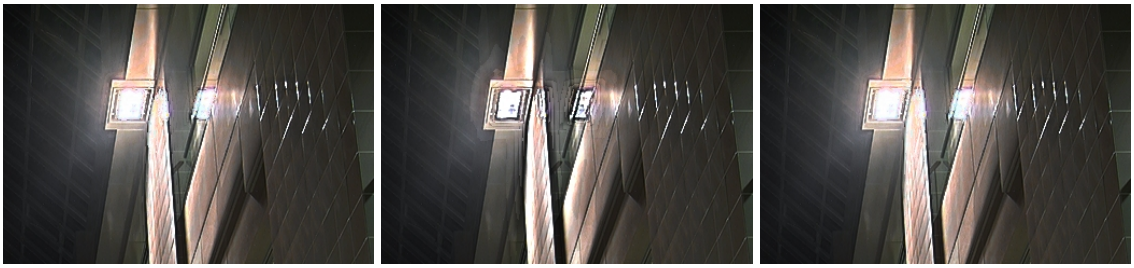

Figure 1.5: A comparison between existing implementations. Left: resulting image using the original photographic operator. Middle: resulting image using Krawczyk *et al.* implementation of the operator, introducing noticeable halos. Right: Resulting image from the proposed technique, which is nearly identical to the original operator. Atrium image courtesy of Greg Ward.

These two observations were used to create a real-time approximation of Reinhard *et al.*'s local tone reproduction operator. Compared to the approach described by Goodnight *et al.*, the proposed technique achieves an average speedup of about five to ten times, while producing very similar results. Compared with Krawczyk *et al.*'s approximation, the speedup is about two to three times, also lowering the memory requirements and minimizing the occurrence of halo artifacts that their method potentially introduces (see Fig-

ure 1.5). The proposed technique then provides an attractive solution for HDR rendering applications that require high performance without compromising image quality.

Computer graphics researches tend to rely on recursive doubling approaches (DUBOIS; RODRIGUE, 1977) to perform the prefix sum operations required to generate a summed-area table (HENSLEY et al., 2005; LAURITZEN, 2007). However, a more work-efficient method is available in the literature since 1990, relying on balanced trees (BLELLOCH, 1990), which is further improved in thesis, thus accelerating the computation of summed-area tables.

1.1 Thesis Contributions

The main contributions of this work can be stated as:

- An approximation for the Reinhard *et al.*'s photographic local tone reproduction operator that produces very similar results to the original technique. The proposed approach runs entirely on the GPU and achieves real-time performance for current display resolutions;
- An efficient approximation of the Blommaert and Martens brightness perception model that might be suitable to other classes of applications.

Besides the above contributions, this thesis also introduces the following contributions, not entirely related with tone-mapping:

- A survey on prefix sum algorithms on GPU and a comparison among them;
- An improved balanced tree algorithm for prefix sum that produces a scan without the need of being derived from a prescan;
- An alternative for generating prefix sums of large arrays using partially-generated summed-area tables and for which its performance on a GPU implementation provides a slight speedup when compared to an existing CUDA variation.

1.2 Structure of the Thesis

The remaining of the text is organized as follows:

- Chapter 2, Dynamic Range, introduces the terms related with dynamic range and associates them with the human visual system, imaging display devices and tone-mapping;
- Chapter 3, Exiting Tone-Mapping Operators, presents the existing tone-mapping operators;
- Chapter 4, The Photographic Operator, reviews the Reihard *et al.*'s operator and shows its qualities compared with the existing ones from Chapter 3;
- Chapter 5, Summed-Area Tables, offers a survey on summed-area tables and GPU algorithms for their generation;
- Chapter 6, Improvements on Prefix Sum Algorithms, presents the proposed improvements;

- Chapter 7, Photographic Operator with Summed-Area Tables, describes the proposed technique for approximating the photographic tone-mapping operator using summed-area tables;
- Chapter 8, Conclusion, summarizes the thesis and its contributions;
- Chapter 9, Future Work, points future directions for the results obtained in this thesis.

The thesis also includes five appendices:

- Appendix A, Complexity Analysis on Prefix Sum Algorithms, derives the complexity for the prefix sums and summed-area tables algorithms cited in the thesis;
- Appendix B, HDR Image Formats, briefly discusses the most widely used HDR image formats;
- Appendix C, HDR Acquisition and Rendering, provides some background about natural light acquisition and its use for realistic image synthesis.
- Appendix D, An Optimal Prefix-Sums Algorithm, includes a photocopy of an optimal prefix-sum algorithm similar to the one proposed in this thesis (see Section 6.1).
- Appendix E, Resumo Estendido em Português, offers an extended summary of this thesis written in Portuguese language.

2 DYNAMIC RANGE

Dynamic range is a term used to describe the distance between the lowest and highest intensity values of a varying quantity, such as sound or light. In the field of imaging, dynamic range is measured as the ratio of the highest and the lowest luminance values presented in the image. Real world scenes usually have a wide range of luminances covering about 14 orders of magnitude, ranging from sunlight (from $10^5 cd/m^2$ up to $10^8 cd/m^2$) to starlight (10^{-3} down to $10^{-6} cd/m^2$) (LEDDA et al., 2005). Figure 2.1 shows a HDR image (left) and a false color representation of its luminances (right). The dynamic range of a HDR image or display device is often called contrast ratio. The measure of contrast ratio is expressed with the notation $C : 1$, meaning that the highest luminance is C times greater than the lowest one. For the real world case, the contrast ratio is expressed as $100,000,000,000,000 : 1$ or, compactly, $10^{14} : 1$.

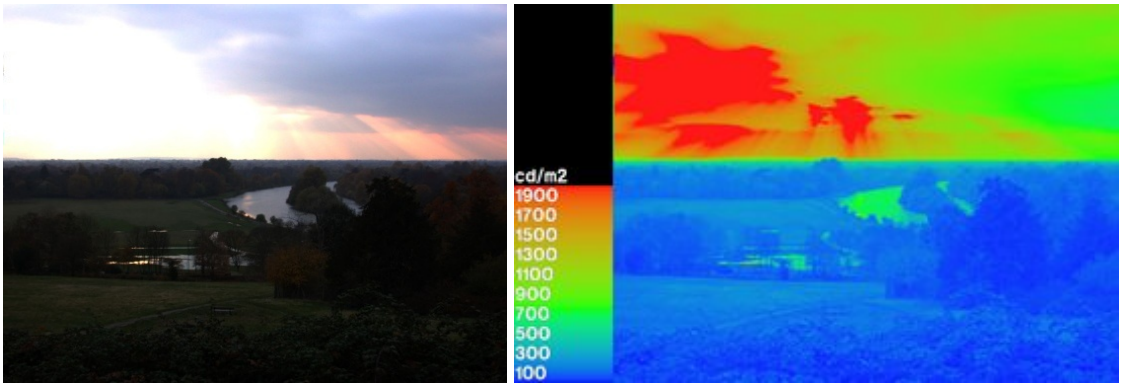

Figure 2.1: A high dynamic range image, without tone-mapping (left), and a corresponding false color representation for its luminance values (right). This image is used as a reference in the remaining of the chapter. Image courtesy of WebHDR Team.

2.1 Human Visual System Overview

The human visual system (HVS) is not able to instantly cover the wide range of luminosity present in the world. Actually, the HVS is limited to a much narrower range of only 4 orders of magnitude at any given time. An adaptation process *shifts* the perceptible range up or down in order to cover the luminosity range of a scene. This adaptation is not an instantaneous process: leaving a bright environment and entering a dark one may take several minutes to properly adapt to the new lighting condition, while the inverse situation is faster, requiring just a few seconds for a proper adaptation (LEDDA; SANTOS;

CHALMERS, 2004).

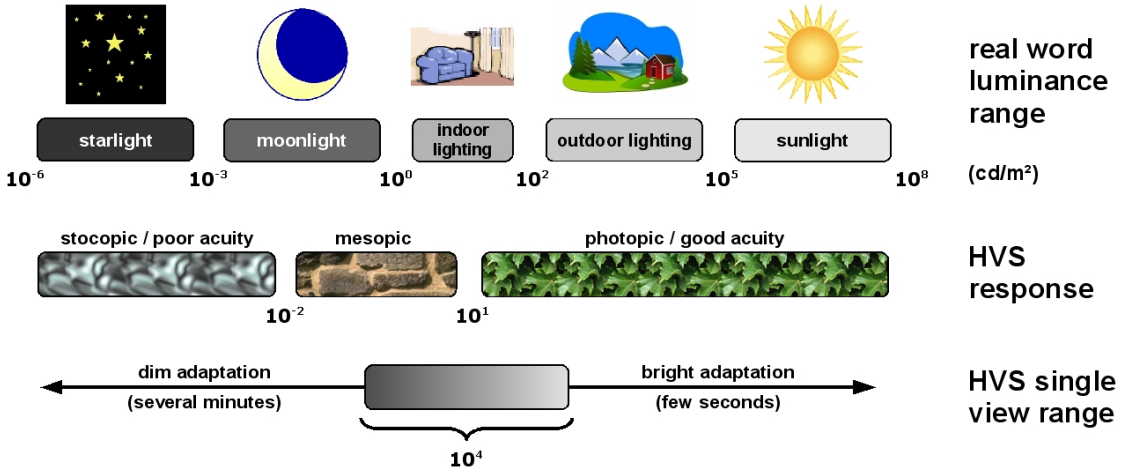

Figure 2.2: The dynamic range of the real world and the human visual system responses to it. In a given instant, the HVS is able to cover a small range of only 4 orders of magnitude, adapting to dark or bright areas accordingly. The overall luminance of the scene is a determinant factor for color and contrast perception. Illustration adapted from (LEDDA et al., 2005).

However, even with this adaptation process, the HVS is unable to equally perceive all the magnitude present in the world (see Figure 2.2). The HVS possesses many physical constraints regarding its response to intensity, color and shape, which lead to an imprecise perception of the details of a scene (SHLAER, 1937; CRAWFORD, 1949). In low-lighting conditions (scotopic vision), the HVS cone photoreceptors tend to be inhibited and the rods photoreceptors take the place since they can effectively deal with small luminance variations, but this leads to poor color dysfunction and poor visual acuity. On the other hand, at well-lit environments (photopic vision), rods become inhibited and cones take place, being responsible for sharp vision and color perception. The transition zone between scotopic and photopic visions is called mesopic vision. Figure 2.3 illustrates the effects of scotopic and photopic vision.

Figure 2.3: Examples of loss of visual acuity and color perception. In low-light environments (left) the HVS is unable to properly distinguish colors and high contrast edges. As the amount of light increases, more details and colors are perceived more accurately (middle and right). Source: (SCHLICK, 1994).

The HVS has a logarithmic curve of response to light intensities. Psychophysical studies have shown that, over a wide range of intensities, the minimum distinguishable difference between two intensities lies in about 1%, usually referred as *visible threshold*

or *noticeable difference* (SHLAER, 1937; STEVENS; STEVENS, 1960; REINHARD et al., 2006). These observations are critical in the design of image encoding formats and display devices technologies (See Appendix B and the next section). Figure 2.4 shows a simulation of how the HVS of an individual with normal vision perceives the HDR image from Figure 2.1.

Figure 2.4: RADIANCE software simulation of human visual system responses for the image of Figure 2.1 (WARD, 1994b).

An additional characteristic of the HVS is that colors and luminances are not faithfully perceived, *i.e.*, the same color or luminance tends to look dark or bright depending on its surrounding (CRAWFORD, 1949; GILCHRIST et al., 1999). Such interesting behavior of the HSV is called *simultaneous contrast* (or hue induction, specifically for colors), and is caused by a visual phenomena called *lateral inhibition*. Actually, the lateral inhibition phenomena lies at the heart of many optical illusions (see Figure 2.5, middle and right). Although it seems to be a flaw of the HVS, practical situations can take advantage of it, which is the case of the widely used display technique called dithering (Figure 2.5, left) (WIEGAND; WALOSZEK, 2003). Blommaert and Martens's investigated how to reproduce the lateral inhibition phenomena in their brightness perception model (BLOMMAERT; MARTENS, 1990). This model was lately used by Reinhard *et al.* to guide the local luminance adaptation process in their photographic tone reproduction operator (REINHARD et al., 2002).

2.2 Dynamic Range Compression

Similarly to the human visual system, display devices are unable to cover the wide range of luminances of real world scenes. Moreover, these devices have a fixed dynamic range and can not adapt to particular lighting conditions. A traditional CRT has a dynamic range of about 2 orders of magnitude (WARD, 2005; FATTAL; LISCHINSKI; WERMAN, 2002). Current LCD and plasma display technology can hold 3 or 4 orders of magnitude (YEE; PATTANAIK, 2003; SEETZEN; WHITEHEAD; WARD, 2003; Reuters, 2008). Some specific HDR displays can reproduce 5 orders of magnitude, but they are very expensive and even such high displayable range is several times smaller than the real world range (Bit-Tech, 2008; Toshiba Corporation; Canon Inc, 2008; Dolby Laboratories, 2008). This constraint is not limited to electronic display devices. Photographic paper has a much narrower dynamic range than photographic film, which also has a limited dynamic range when compared to a real world scene. In fact, the dynamic range compression problem was addressed almost two decades ago by photographers (ADAMS,

1983; WHITE; ZAKIA; LORENZ, 1984; WOODS, 1993; REINHARD et al., 2002).

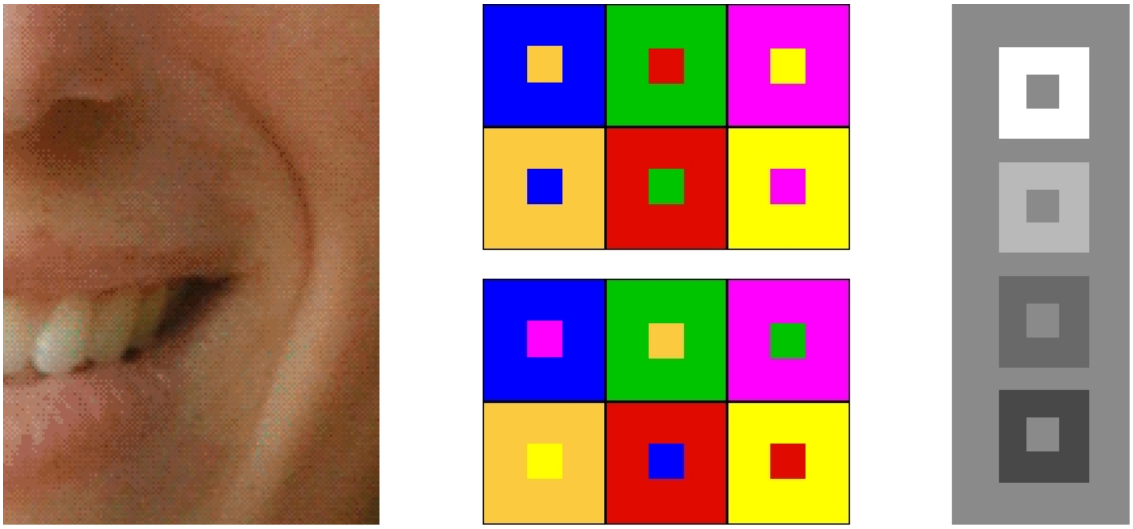

Figure 2.5: The lateral inhibition phenomena in action. Left: dithering is a process of placing different-colored pixels next to each other to create an illusion of additional colors; the eye sees the two adjacent colors, and the mind blends them into a third one. Middle: color contrast intensifies complementary colors; the same central colors look less intensive with other surrounding colors. Right: simultaneous contrast lets the central gray square look differently, depending on the surrounding gray level, although all the central squares are composed of the same shade of gray. Source: (WIEGAND; WALOSZEK, 2003).

The proper presentation of imaging content requires that the display devices support the dynamic range of the image. When this is not possible, the dynamic range of the image must be compressed to fit into the displayable range capabilities. Without such compression, the resulting image might look excessively dark or bright (as can be seen in Figure 2.1). The most straight-forward approach to high dynamic range compression is a linear map, normalizing each pixel luminance using the largest luminance of the image, but that leads to poor image quality, except in some isolated situations. For such a reason, the use of non-linear mapping is more suitable to fit the HVS logarithmic curve of responses, and image capture and display devices usually employ non-linear quantization techniques to avoid the loss of perceptive details (*e.g.* gamma encoding) (ADAMS, 1983; CGSD, 1990; POYNTON, 2003). Refer to Figure 2.6 for an example.

Since brightness perception by the HVS behaves according to a logarithmic function (*i.e.* brightness is perceived non-linear), taking fixed steps to quantize luminances (linear encoding) produces abrupt changes in darker regions, which tend to become smooth in bright ones, as shown in Figure 2.7. A gamma encoding uses varying step sizes: as the luminance grows the step size grows as well, providing more uniform smooth transitions (CGSD, 1990). An effective encoding keeps the difference between two close quantized values below the visible threshold of the HVS (1%) (WARD, 2005).

Gamma encoding seems to be an attractive solution to encode and display HDR content. However, since luminance ranges from small fractions to millions, it is impossible to assume that an observer is adapted to a particular level, and the simple addition of bits to a gamma encoding will not produce a good distribution of steps (WARD, 2005). To properly display HDR content, more sophisticated compressions approaches must be em-

Figure 2.6: Automatic luminance compression using a linear scaling (left) and a gamma compression (right). The former loses almost all of the details of the image. A gamma compression is more attractive, but it still is far away from the results expected from Figure 2.4.

Figure 2.7: A linear encoding (left) takes fixed steps to quantize luminances. Since luminances are perceived non-linearly, in a logarithm fashion, abrupt changes are more noticeable in darker regions. A gamma encoding (right) uses a variable step size for quantization, providing a better fitting for the logarithm nature of the HVS responses to luminances. A good encoding keeps the relative difference of two close quantized intensities below the visible threshold of 1%. Source: (WARD, 2005).

ployed, and it is important to drive this compression in a way that it handles perceptual characteristics of the human visual system. Tone mapping operators provide such kind of compression scheme (TUMBLIN; RUSHMEIER, 1993).

2.3 Tone-Mapping Operators

Tone reproduction is a practice being addressed by photographers for almost two centuries (LONDON; UPTON, 1998). Its goal consists on reproducing relevant details of a photographed scene on a limited photographic paper. Similarly, digital tone-mapping is a technique to scale the luminances of a digital HDR image to the displayable range of the target devices. In the context of this work, unless explicitly stated otherwise, all references to *tone-mapping* or *tone-reproduction* have the word *digital* implicitly placed before them.

A theoretical basis for tone reproduction operators' design was introduced in computer graphics by Tumbling and Rushmeier (TUMBLIN; RUSHMEIER, 1993), although Miller *et al.* and Upstill early stated the problem of brightness matching between real scenes and device-displayed ones (MILLER; NGAI; D., 1984; UPSTILL, 1985). An overview of their theoretical basis is illustrated in Figure 2.8.

Basically, Figure 2.8 states that a good tone reproduction operator must give the same perceptual response to a viewer faced with a tone-mapped image and its respective real scene. Although this is a simple and intuitive model, the development of such a robust

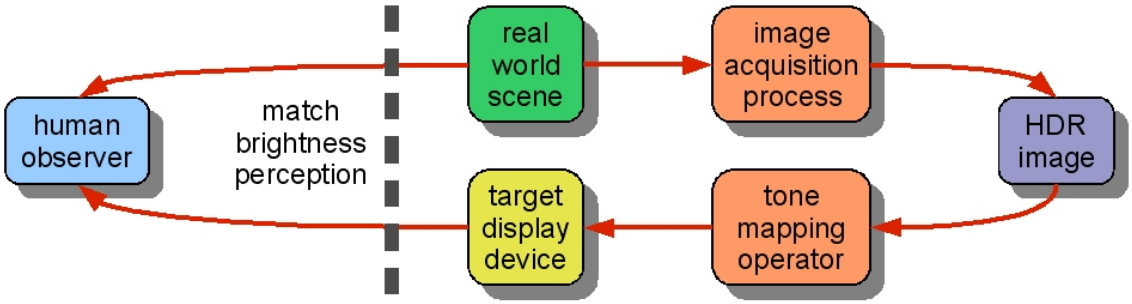

Figure 2.8: A theoretical model for tone-mapping operators' design. A good operator must be able to produce the same visual response to a viewer faced with the original scene and with the tone-mapped one.

operator is a challenge. Such challenge is closely related to the subjective nature of the HVS as well as with the limited displayable range of the target devices. Note that the model also requires a proper solution to acquire and store a HDR image, and an overview on this topic is available in Appendices B and C.

Figure 2.9: The resulting image after applying Reinhard *et al.*'s photographic tone-mapping operator (REINHARD et al., 2002). Note the similarities with the simulated results on RADIANCE in Figure 2.4.

Tone-mapping operators can be either global (spatially uniform) or local (spatially varying). Due to the perceptual evocation, operators tend to rely on psychophysical experiments, and many were proposed to effectively fulfill Tumblin and Rushmeier's framework requirements. Refer to Figure 2.9 for an example of tone-mapped image using a perceptually-driven operator.

There is another class of tone reproduction operators that tries to maximize the detail preservation, exploring all the capabilities of the device. However, such class of operators often ignores any psychophysical characteristic of the HVS, relying only on mathematical models. Although they can allegedly produce artistically appealing results, they tend to look unnatural (see Figure 2.10). In the field of realistic image synthesis, perceptual data is crucial and should not be ignored. For such reason, this class of operators is not covered in this work.

An important note about any tone-mapping operators is that they are not a physically-sound approach. They comprise just a practical solution to properly display HDR content in limited devices, preventing the loss of perceptible contrastive details as much as possible. An overview on exiting tone-mapping operators is available in Chapter 3.

Figure 2.10: All images above were taken from real scenes and submitted to a special tone-mapping operator. The process maximized details so strongly that the resulting images appear to be unreal. Source: HDR Japan.

2.4 A Few Words on Exposure Control

It is common to confuse exposure control with tone-mapping. A simple exposure control of a HDR image does not imply that the dynamic range of the resulting image will fit in the displayable range. In contrast, tone-mapping operators effectively fulfill this requirement.

A similar approach to a tone-mapping operator can be achieved just by changing exposure and gamma correction parameters (NVIDIA Corporation, 2004). Although fast, it has serious limitations. The first limitation is that it is not a true tone-mapping operator, since it just jumps from an exposure to another, which does not guarantee that the resulting image range of intensities will fit in the display range (see Figure 2.11). The second limitation is that gamma compression can not properly deal with high dynamic range intensities. Another limitation is that it is not robust enough to be applied together with environment mapping, since a proper exposure and gamma parameter must be determined for each viewing directions and for each environment map. This requires a lot of manual intervention by an artist to optimally setup the parameters. The last limitation is that it does not reproduce HVS perceptual effects. For these reasons, more robust operators are desirable.

Figure 2.11: A simple exposure control on a HDR image does not ensure that the luminances will fit in the displayable range. At each figure, from left to right, the exposure was doubled.

3 EXISTING TONE-MAPPING OPERATORS

This chapter presents the related work concerning existing tone-mapping operators, and Chapter 4 reviews in detail the photographic tone reproduction operator (REINHARD et al., 2002). Devlin's survey on tone-mapping operators (DEVLIN, 2002) and Reinhard *et al.*'s book on HDR rendering (REINHARD et al., 2006) provide further information about the operators cited in this chapter.

3.1 Global Tone-Mapping Operators

Global (spatially uniform) tone-mapping operators use the same set of parameters to compress the intensity of any pixel of the image. Since both linear map and gamma compression are not suitable for HDR images, Tumbling and Rushmeier used a non-linear mapping, based on the brightness perception measurements of Stevens and Stevens (STEVENS; STEVENS, 1960), to mimic the human visual system responses (TUMBLIN; RUSHMEIER, 1993). A slightly different approach was presented by Ward, ensuring that the smallest perceptual difference in the real scene matches the smallest perceptible difference in the displayed image (WARD, 1994a).

Ferwerda *et al.* presented an operator that deals with color perception at different lighting conditions (FERWERDA et al., 1996). By monitoring the color sensitivity acuity, it can gradually move from scotopic (lowlight) vision to photopic (well-lit) vision. Ward *et al.* proposed a histogram-based operator that iterates until finding an acceptable adjustment that matches with the display capabilities (WARD LARSON; RUSHMEIER; PIATKO, 1997). Just like Ferwerda *et al.*'s operator, it deals with scotopic, mesopic, and photopic vision. Tumblin *et al.* then presented two new operators in the same work (TUMBLIN; HODGINS; GUENTER, 1999): the former is based on a layered HVS model and is suitable only for synthetic scenes, since layered data must be stored along with the image; the latter requires manual intervention to select regions where fine details must be preserved, compressing the remainder. Following the perceptually-driven approach of Tumblin and Rushmeier's operator, Pattanaik *et al.* developed a time-dependent operator, making it suitable for dynamic scenes and video (PATTANAIK et al., 2000).ed a time-dependent operator, making it suitable for dynamic scenes and video (PATTANAIK et al., 2000).

More recent global tone-mapping algorithms include Drago *et al.* biased logarithmic luminance compression (DRAGO et al., 2003), and Reinhard *et al.* photographic global operator (REINHARD et al., 2002). Both are simple and fast, but the former tends to reduce the overall contrast of the image. The latter is based on a photographic technique called Zone System (WHITE; ZAKIA; LORENZ, 1984), which effectively maps the image's average luminances to the medium intensity of a target display. Examples of the

various global tone-mapping operators cited so far are given in Figure 3.1.

Figure 3.1: A plate of resulting images from various global tone-mapping operators. In yellow, the source of each image.

Although most of them are fast enough to pursuit real-time performance, global operators fail to preserve relevant contrastive details of the image. To surpass this limitation, more sophisticated operators can be employed, comprising the class of local tone-mapping operators.

3.2 Local Tone-Mapping Operators

Local (spatially-varying) tone-mapping operators essentially work by adjusting each pixel's luminance based on its neighborhood. By potentially applying a separate function to each pixel of the image, they are able to successfully compress much larger dynamic ranges and tend to produce images with better contrast of high-frequency details when compared to global ones. This comes at the expense of higher computational efforts and, in some cases, the need of many user-specified parameters or manual intervention, which has prevented their use in real-time applications. The support for high-precision floating-point buffers in modern GPUs and the availability of a large number of HDR environment maps is making HDR rendering increasingly popular, but essentially restricted to the use of global operators, due to the performance drawbacks and tunning constraints introduced by local operators.

Chiu *et al.* presented the first work regarding spatially-varying luminance compression (CHIU et al., 1993). They used an average of luminances at each pixel neighborhood to perform dynamic range compression. On images with low frequency transitions, their approach proved to be satisfactory. As for high frequency variations, halos artifacts are clearly noticeable. Lately, Schlick *et al.* used polynomial functions to perform luminance compression and most of their results also suffered from strong halos artifacts (SCHLICK, 1994). Pattanaik *et al.* presented a multi-resolution HVS-based local operator that deals with scotopic and photopic visions (PATTANAIK et al., 1998). Their technique comprises two stages: the visual model, encoding both the perceived chromatic and achromatic contrasts; and the display model, which takes the encoded data to output a reconstructed image. They also suffered from serious halo artifacts. As can be seen, a feature common

to all local tone-mapping operators is the occurrence of such visible halos around high-contrast edges. The nature of such halos is often not well understood, but it essentially results from the mixing of low and high luminance values across high-contrast edges in certain pixel neighborhoods. This affects the average luminance used to compress the pixel's own luminance, resulting in such undesirable artifacts (REINHARD; DEVLIN, 2005; REINHARD et al., 2006). Although all local tone-mapping operators are prone to this kind of artifacts, good operators try to minimize their occurrence.

Reinhard *et al.* presented an elegant photographic-based technique to perform tone reproduction (REINHARD et al., 2002). Their automatic spatially-varying filter enhances contrastive details, driven by a HVS-based brightness perception model (BLOMMAERT; MARTENS, 1990), and prevents the occurrence of halos. They confronted their results with a large number of existing operators and HDR images, proving its effectiveness. The photographic operator is probably the most balanced and robust operator available. A complete review of the photographic operator is available in Chapter 4.

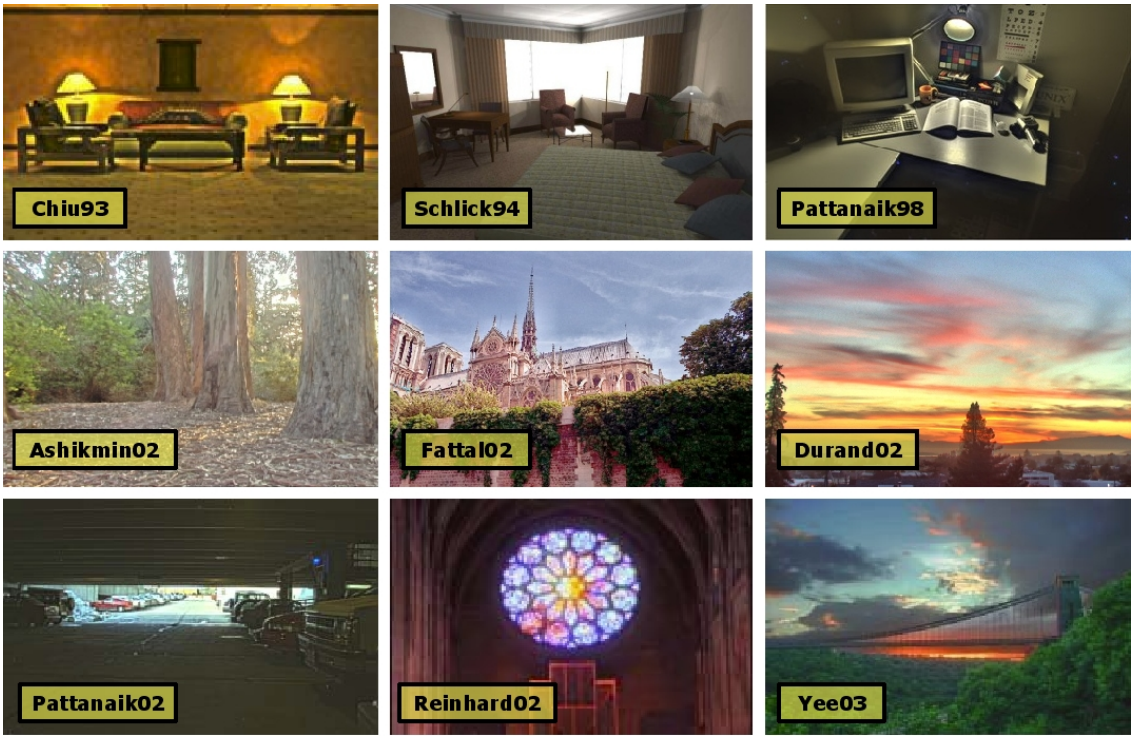

Figure 3.2: A plate of resulting images from various local tone-mapping operators. In yellow, the source of each image.

Recent works on local tone-reproduction include the work of Fattal *et al.*, which uses luminance gradient attenuation to compress large gradients and insert fine details (FATTAL; LISCHINSKI; WERMAN, 2002). However, the operator is not perceptually-driven and, while fast, its performance is not acceptable for real-time purposes. Goodnight *et al.* also stated that a reproduction of Fattal *et al.*'s operator in GPU is challenging (GOODNIGHT et al., 2003). Lately, Durand and Dorsey used an edge-preserving filter, the bilateral filter (DURAND; DORSEY, 2002), to split the image in two layers: a base layer, consisting of large intensity variations; and a detail layer, consisting of fine details. Their approach minimizes halo artifacts but is expensive and does not address well time-dependent situations. In a similar way from the two previously cited operators, Pattanaik *et al.* proposed a method to evaluate the influence of the surrounding pixel inten-

sities based on absolute magnitude changes (PATTANAIK; YEE, 2002). They used some photographic standards to guide the operator in order to find the proper acceptable iso-luminant neighborhood (HUNT, 1996). This approach reduces halo artifacts, but is not suitable for time-dependent rendering and does not address well the perceptual evocation.

Lately, Ashikhmin used an interesting multi-pass approach to reinsert relevant details that were lost due to luminance compression (ASHIKHMIN, 2002). First, local adaptation luminance of each pixel is estimated, followed by a simple compression function to map them into the displayable range. Lost details are identified and reintroduced in a final pass. His approach is simple, fast, and produces good results, but the author himself claimed that Reinhard *et al.*'s photographic operator produces better results and handles a wider range of images. The latest local tone-mapping operators comprises the work of Yee *et al.* (YEE; PATTANAIK, 2003) and Chen *et al.* (CHEN; PARIS; DURAND, 2007). The former uses image segmentation, grouping and graph operations to locally adapt luminance. The technique suffers from many parameters' tunning and from flood-fill operations, which are not GPU friendly (LEFOHN et al., 2003; EISEMANN; DÉCORET, 2008). The latter presents an attractive data structure, the bilateral grid, allowing fast edge-aware image processing, which includes Durand and Dorsey's bilateral-filtering tone-mapping operator (DURAND; DORSEY, 2002). Although it effectively accelerates the operator, interactive rates are possible only on small regions, requiring manual intervention to select small areas where the image must be locally adapted. Examples of the various local tone-mapping operators cited here are given in Figure 3.2.

4 THE PHOTOGRAPHIC OPERATOR

Reinhard *et al.* digital photographic tone reproduction operator (REINHARD et al., 2002) is based on a photographic technique called Zone System (WHITE; ZAKIA; LORENZ, 1984). The term *photographic* on its name is solely related to the nature of the photography background used to develop the operator. Their method does not intend to exactly mimic the photographic process, but instead uses its basic conceptual framework to manage tone reproduction choices (REINHARD et al., 2002). It is important to emphasize that, unless explicitly stated otherwise, all claims to the terms *tone-mapping* or *tone-reproduction* have the word *digital* implicitly placed before them.

The photographic operator has many advantages over existing operators, which can be summarized as follows:

- Simplicity: requires few parameters. Some of them can be automatically adjusted, while default settings for the remaining are effective in most cases;
- Automaticity: requires no user intervention. The operator enhances details in the whole image and no manual user intervention is required to mark interest regions;
- Robustness: suitable for a wide range of images, from overexposed to underexposed, indoor to outdoor scenes, and enhances contrast even in low dynamic range ones;
- Perceptually driven: enhances relevant contrastive details relying on a model of brightness perception of the human visual system;
- Popularity: its global operator is fast and GPU implementations are available in traditional graphics SDKs (Microsoft Corporation, 2004; NVIDIA Corporation, 2005). In a system that already uses it, the local variant is easy to adapt.

Although the original operator was unable to deal with some important perceptual effects of the HVS, further research successfully extended it to surpass these limitations in a way that introduces a minimal performance overhead. Goodnight *et al.* shown how to add temporal luminance adaptation to the operator, making it suitable to be used in real-time and video applications (GOODNIGHT et al., 2003). Krawczyk *et al.* presented a method to derive perceptual effects of the HVS directly from the set of Gaussian-filtered luminance images computed by the local operator, simulating loss of visual acuity, scotopic and photopic vision, and veiling luminance effects (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005). Renihard and Krawczyk *et al.* also proposed approaches for automatically estimating some parameters of the operator, deriving them from the contents of the input image (REINHARD et al., 2002; KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005).

Such perceptual effects and additional automaticity are certainly desirable features, but they are not critical requirements. Thus, the present work focus mainly on the performance speedup of the most costly part of the local operator, the convolution at different scales. The photographic tone reproduction operator can be summarized as:

- Obtain a luminance representation of the input HDR image;
- Produce a relative luminance image for which the middle-tone of the original image matches the middle intensity of the display device;
- Apply a spatially-uniform filter or a spatially-varying filter to refine the compression, ensuring that the resulting image intensities will fit in the displayable range;
- Scale the original chromatic components accordingly to the compressed luminance.

Before going any further in the photographic operator algorithm, it is useful to understand the nature and the goals of the photographic technique that inspired it: the Zone System.

4.1 The Zone System

The Zone System was introduced in the photography field by Adams and Archer in early 1940 (ADAMS, 1983; WHITE; ZAKIA; LORENZ, 1984). Basically, it can be defined as a set of zones that maps real scene luminances (scene zones) to the reflectance capabilities of a photographic print (print zones). Such concept is similar to dynamic range compression, but photographers have a subjective concept of dynamic range, since they are more interested in the ratio between the highest and lowest luminance regions where detail is visible (REINHARD et al., 2002).

The zone system is discretized in eleven print zones, ranging from the lowest to the highest reflectance permitted by the print. The extreme zones are called pure black (zone 0) and pure white (zone X), respectively, and the middle zone V corresponds to 18% of the print reflectance (the middle-gray of the print). Each zone differs from the preceding or following zone by a factor of two, so that zone I exposure is twice the one of zone 0, and so forth. The Zone System is illustrated in Figure4.1. Actually, the useful range comprises zone I to zone IX; Adams stated that zone 0 and zone X correspond to extreme situations where detail will never be perceived (ADAMS, 1983). In the Zone System, the dynamic range is then stated as the distinguishable difference between zones (REINHARD et al., 2002).

To perform tone reproduction, relying on a photometer or on his own photographic skills, a photographer must select what he believes to be the *middle-gray tone* of the entire scene. This is indeed a subjective choice, but there are guidelines to assist the photographer in this task, which is the case of the *key value* (see Figure 4.2).

Bright scenes are said to be high-key, while dark ones are low-key. In typical scenes (medium-key), it is reasonable to think to map the middle-gray tone of the scene to the medium zone of the print (zone V). When the results are not satisfactory, the photographer can favor bright or dark areas, just by altering the key value (see Figure 4.2). This prevents the photographer having to take taking several subjective measurements of the scene's middle-gray, being able to work only with the print zones. Note that one can also think in keeping the key value unchanged and varying the middle-zone of the print, but since the reflectance of the print is not a subjective value, altering the key value is more appropriate.

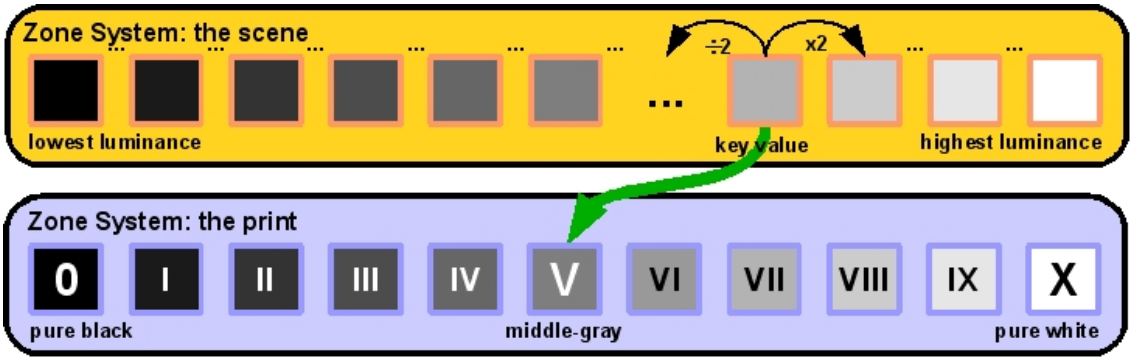

Figure 4.1: The Zone System maps scene zones into print zones. The proper middle-gray tone of the scene is mapped to the middle-zone of the print, which typically is 18% of the reflectance of the print. Each print zone comprises twice the scene luminance of the previous one.

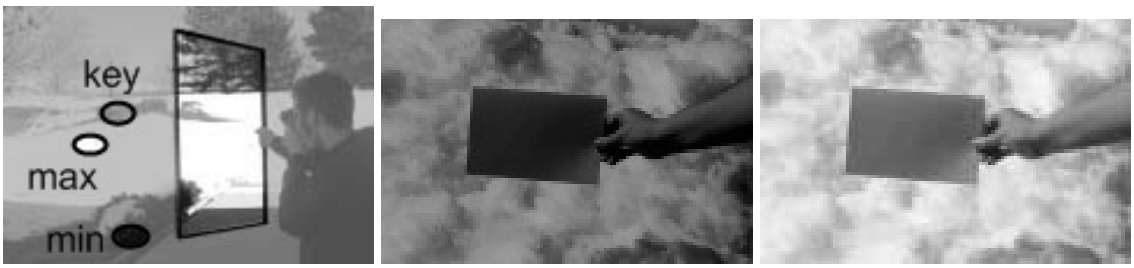

Figure 4.2: A photographer estimates what he believes to be the proper key value of the scene (left). That value is used to map the luminances to the print reflectance using the Zone System. If the resulting image looks dark (middle), a low-key was estimated and can adjusted it to perform a high-key map (right), enhancing the overall image quality. As can be seen in the leftmost image, it is useful to also keep notes about the highest and lowest luminances of the scene. When the luminance range of the scene exceeds the nine intermediate zones, the photographer can use this information to locally adapt the contrast of the image. Source: (REINHARD et al., 2002).

An experienced photographer might also predict printing problems, taking notes about the darkest and brightest areas of the scene. If the dynamic range of the scene is compatible with what the print is able to reproduce, a proper choice of a middle-gray tone guarantees that all contrastive details will be properly noticed. Otherwise, perceptible details will certainly be lost. Note that such loss of contrastive details might be acceptable in some cases to suppress excessively dark or bright regions.

The photographer can then simply ignore the exceedings, which will be automatically mapped to pure black and pure white. However, this rarely leads to good results. To prevent such a loss, the photographer can compresses the lowest and highest luminances using a transfer function. Although this improves the overall quality, the resulting image appears to be blurred due to potential contrast reduction. As a final resource, the photographer can employ a *dodging-and-burning* technique and locally adapt the contrast for each region, independently, thus preserving substantial contrastive details. However, this process is exhaustive and very difficult to generalize and automate.

4.2 The Digital Photographic Operator

Reinhard *et al.*'s digital photographic tone reproduction operator performs the stages of the Zone System and also provides an automatic process for *dodging-and-burning* to enhance relevant contrastive details. Since the process requires to work with luminances, a mapping between color and luminance (and *vice-versa*) is required. For a given input HDR image I, the RGB components of a pixel with coordinates (x, y) are expressed as:

$$I(x,y) = (I(x,y)_r, I(x,y)_g, I(x,y)_b) \tag{4.1}$$

and the luminance L of the given pixel is defined as a weighted sum over each of its color components, as follows:

$$L(x,y) = 0.299 \cdot I(x,y)_r + 0.587 \cdot I(x,y)_g + 0.114 \cdot I(x,y)_b \tag{4.2}$$

Figure 4.3 shows an example of such RGB to luminance conversion.

Figure 4.3: The unprocessed HDR RGB image (left) and its corresponding HDR luminance image (right), obtained by the evaluation of Equation 4.2 for each of the color pixels.

After dynamic range compression (this will be described later in this chapter), the *resulting tone-mapped image* is obtained by scaling the original RGB color components of each pixel $I(x, y)$ according to its *compressed luminance* L' as follows:

$$I'(x,y) = L'(x,y) \cdot \left(\left(\frac{I(x,y)_r}{L(x,y)} \right)^{\gamma}, \left(\frac{I(x,y)_g}{L(x,y)} \right)^{\gamma}, \left(\frac{I(x,y)_b}{L(x,y)} \right)^{\gamma} \right) \tag{4.3}$$

where γ is used to perform gamma correction, which controls the saturation of the recovery, typically ranging from 0.4 to 0.8 (GOODNIGHT et al., 2003).

From the statements above, a more formal definition of the photographic operator is as follows:

$$L \longmapsto L \longmapsto L' \longmapsto I' \tag{4.4}$$

where the first and the last mappings are performed according to Equations 4.2 and 4.3, respectively.

The challenge now is to find an appropriate mapping for $L \longmapsto L'$. For such a goal, the photographic operator uses the background provided by the Zone System, and the first requirement is to find an adequate key value for the whole scene. The logarithmic average

of luminance of the image proved to be a good approximation, as stated by previous researches (TUMBLIN; RUSHMEIER, 1993; WARD, 1994a), and is evaluated as:

$$\widetilde{L} = exp\left(\frac{1}{N}\sum_{x,y} log(L(x,y)+\delta)\right) \tag{4.5}$$

where N is the number of pixels in the image and δ is a small value to prevent $log(0)$.

Next, each pixel luminance must be linearly scaled, according to the Zone System print zones. Supposing that the input image has normal key, it is desirable to map the image's key value, $\widetilde{L}$, to the middle zone of the target print, 18%, suggesting the following equation:

$$L_r(x,y) = \alpha\frac{L(x,y)}{\widetilde{L}} \tag{4.6}$$

with α being 0.18, in a scale ranging from 0 to 1.

Note that the parameter α must be adjusted for low or high-key images because it relates to the key of the image after applying the scaling. However, in practical terms, α can also be used to favor dark or bright areas, no matter if the input image has low or high-key. Typically, a value of $\alpha = 0.18$ is the same used in automatic exposure control systems in cameras (GOODNIGHT et al., 2003).

The equation above gives a measurement of *relative luminance* of the image. However, it does not ensure that all values will fit in the displayable range of the target display device. Recall that, in the original Zone System, when such happens, the photographer can do three things: (i) accept the potential loss of detail, mapping the bounding luminances to pure black and pure white; (ii) apply a transfer function to compress the exceeding luminances; (iii) or employ a *dodging-and-burning* process to locally adapt the contrast.

Figure 4.4: The relative luminance image (left) obtained by evaluating Equation 4.6 on the luminance image of Figure 4.3. At the right, the resulting tone-mapped image obtained by plugging the relative luminances on Equation 4.3 (*i.e.*, making $L'(x,y) = L_r(x,y)$). As can be seen, the resulting image barely improved the quality of the unprocessed image, since it has a dynamic range higher than the one that is support to display it.

4.3 The Global Operator

The global variant of the photographic operator is basically a transfer function to compress any luminance that exceeds the displayable range. Reinhard *et al.* employed

a function that mimics the same characteristics of the ones commonly used by photographers (MITCHELL, 1984), performing a non-linear mapping, as defined below:

$$L_d(x,y) = \frac{L_r(x,y)}{1 + L_r(x,y)} \tag{4.7}$$

where $L_d(x,y)$ is agreeably inside the output displayable range of $[0,1)$. A plot of this function over a high dynamic range is shown in Figure 4.5.

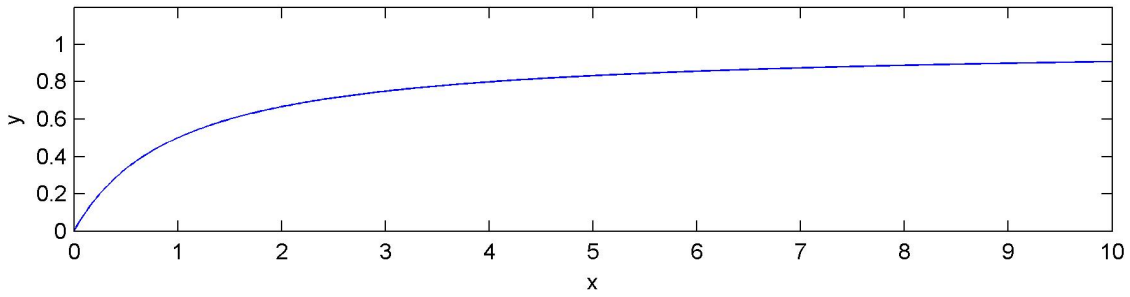

Figure 4.5: A plot of the spatially uniform filtering function (Equation 4.7) of the photographic operator. This functions ensures that the compressed luminances will fit in the displayable range of the target display. The $x\text{-}axis$ represents the logarithm of the input relative luminance ($L_r(x,y)$), while the $y\text{-}axis$ represents the compressed value within the displayable range of $[0,1)$.

Such a simple spatially-uniform filter is able to gracefully map the relative luminance $L_r(x,y) = 1$ to the pixel intensity $L_d(x,y) = 0.5$, thus effectively mapping the middle luminance of the scene to the the middle intensity of the device. The display device can then present the resulting image contents using its own encoding procedure. However, due to the extensive contrast compression, it only preserves details in low-contrast regions. Perceptible details tend to be lost in very bright regions as well as in high-frequency textures. See Figure 4.6 for an example of the global operator.

4.4 The Local Operator

The local photographic operator replaces the spatially-uniform function of its global variant with a spatially-varying one. The goal is to automatically simulate *dodging-and-burning* in photography, which is applied over regions bounded by large contrasts (ADAMS, 1983; REINHARD et al., 2002). Using the background given in previous sections, one can think about it as selecting a proper value for for each individual pixel.

To find such large contrasts, one approach concerns the determination of the size of each local region, thus giving a measure of the local contrast (PELI, 1990). This can be estimated convolving the image using different center-surround functions at multiple spatial scales. Typical solutions employ the difference between Gaussian-blurred images, which is the case of Blommaert and Martens' model for brightness perception (BLOMMAERT; MARTENS, 1990), which proved to give the best results for the photographic operator (REINHARD et al., 2002).

The spatially-varying function is obtained directly from Equation 4.7 by replacing the relative luminance $L_r(x,y)$ in the denominator with the weighted average $V(x,y,s_{max})$

Figure 4.6: The result of evaluating Equation 4.5 with the relative luminance image from Figure 4.4. Color was recovered by evaluating Equation 4.3 with $L'(x, y) = L_d(x, y)$. The spatially-uniform function ensures that the resulting compressed luminances will fit the displayable range. Although the resulting image quality is superior to the one of Figure 4.4, a spatially-varying function is able to produce better results.

of the approximately isoluminant neighborhood of the given pixel:

$$L_d(x, y) = \frac{L_r(x, y)}{1 + V(x, y, s_{max})} \tag{4.8}$$

The choice of an appropriate scale s_{max} is a determinant factor to adapt local contrast. If a small scale is assumed, details can be lost. Such statement is intuitive: if one uses a Gaussian hierarchy of just one level, no differences are computed and no threshold is applied, reverting Equation 4.8 back to the spatially-uniform filter of Equation 4.7. On the other hand, a large scale potentially introduces halos in the resulting image, due to the mixing of low and high luminance values across high-contrast edges (Figure 4.7).

To determine the appropriate scale s_{max}, a set of Gaussian-filtered images is generated from the relative luminance image L_r of Equation 4.6, at successive scales s:

$$V(x, y, s) = L_r(x, y) \otimes G(x, y, s) \tag{4.9}$$

where $\otimes$ is the convolution operator and $G(x, y, s)$ is a rotationally-symmetric Gaussian kernel, as defined below:

$$G(x, y, s) = \frac{1}{\pi\sigma^2} e^{-\frac{x^2+y^2}{\sigma^2}} \tag{4.10}$$

where $\sigma = \frac{1}{2\sqrt{2}}s$ and s is the scale. The choices for the sizes of each scale will be explained later. Figure 4.8 shows some Gaussian-filtered images produced from the relative luminance one from Figure 4.4.

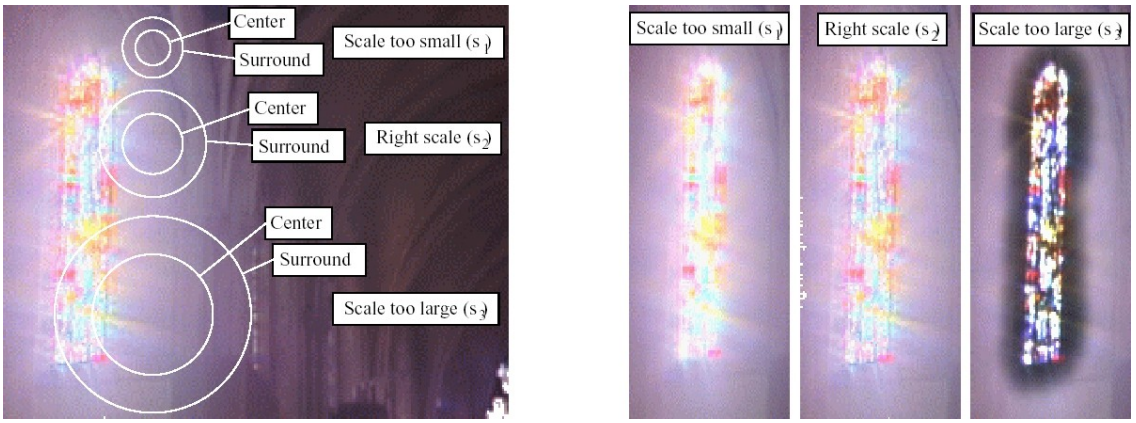

Figure 4.7: Example of scale selection. The left image shows kernels at different sizes, and the images at the right shows the results of particular choices to scale selection. Detail is lost when a small scale is chosen, but halos start to appear when the scale is too large. Source: (REINHARD et al., 2002).

Each resulting image in the hierarchy of filtered images, defined by Equation 4.9, is referred in this context as an *adaptation zone* (the term zone in this context *bears no connection* to the zones of the Zone System). These adaptation zones are then used to compute normalized differences, according to the brightness perception model proposed by Blommaert and Martens (BLOMMAERT; MARTENS, 1990):

$$W(x, y, s_i) = \frac{V(x, y, s_i) - V(x, y, s_{i+1})}{2^{\phi}/s^2 + V(x, y, s_i)} \tag{4.11}$$

where α is the key value from Equation 4.6, and the entire denominator is a normalization factor guided by a sharpening factor, which typically is $\phi = 8$ (REINHARD et al., 2002). Although ϕ is able to control edge enhancement, it has a small influence over the resulting compressed image (REINHARD et al., 2006). Krawczyk *et al.* made $2^{\phi}/s^2 = 1$ in their work (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005). Figure 4.9 shows the normalized difference images applied to the Gaussian-filtered images from Figure 4.8.

Finally, to determine the appropriate scale s_{max} for a given pixel, the technique looks for the largest scaled difference between two consecutive adaptation zones that do not exceed a given threshold ϵ, suggesting the following equation:

$$|W(x, y, s_{max})| < \epsilon \tag{4.12}$$

Such a scale s_{max} represents the largest Gaussian-modulated neighborhood around a pixel for which no substantial luminance variation occurs. Reinhard *et al.* optimized the threshold empirically, setting it to $\epsilon = 0.05$ (REINHARD et al., 2002, 2006). The resulting largest scale is then used to measure the local contrast of the given pixel, $V(x, y, s_{max})$, performing its local compression as defined in Equation 4.8. Figure 4.10 shows a false color representation of the estimated maximum scale s_{max} for each pixel of the relative luminance image from Figure 4.4, as well as their corresponding local contrast measurement $V(x, y, s_{max})$. The resulting tone-mapped image is shown in Figure 4.11.

Although an arbitrary number of adaptation zones is possible, a total of 8 is sufficient, due to the lateral inhibition phenomena of the human visual system, where the competition between the photoreceptors is more intense over small scales (BLOMMAERT; MARTENS, 1990; WIEGAND; WALOSZEK, 2003). The first scale is 1 pixel wide, and each subsequent scale is 1.6 times larger than the previous. Considering only the center surround kernels (where their sizes are odd numbers) one has $s \in \{1, 3, 5, 7, 11, 17, 27, 43\}$ (REINHARD et al., 2002).

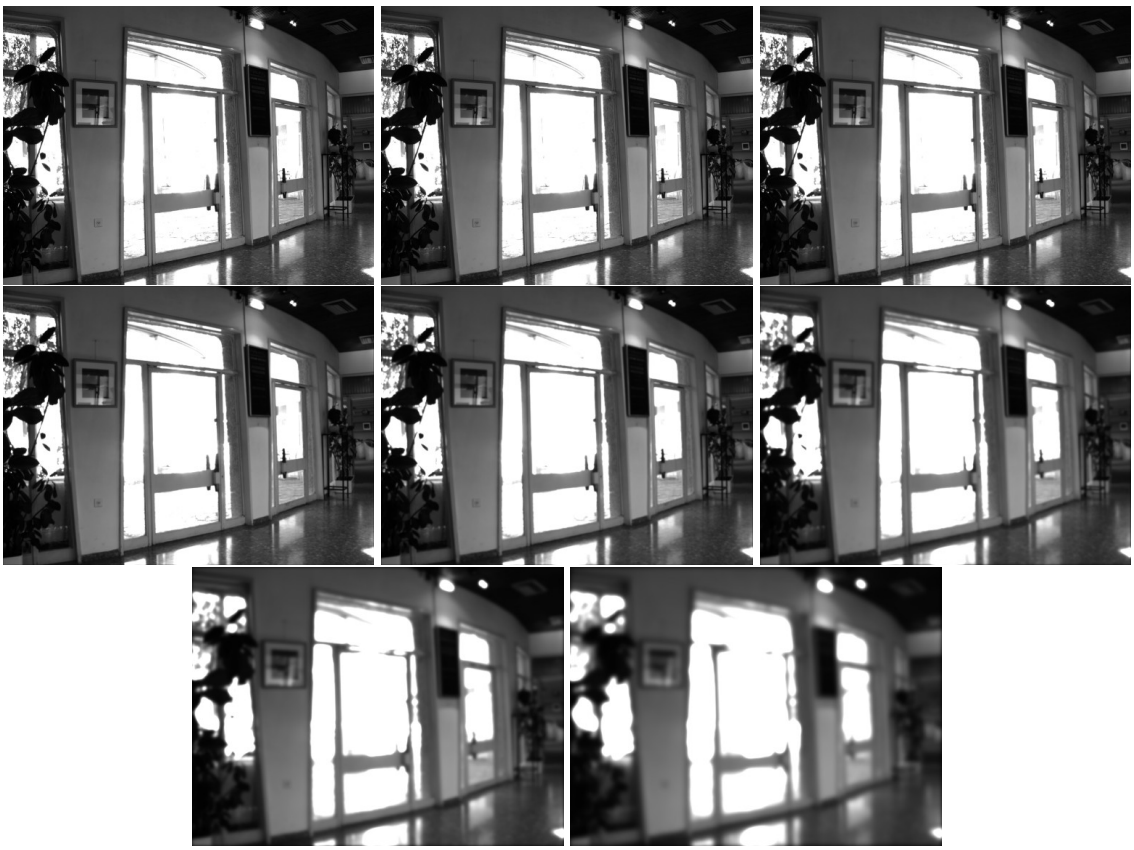

Figure 4.8: A set of Gaussian-filtered images from the relative luminance from Figure 4.4, according to Equation 4.9. The kernel sizes are, from left to right, top to bottom, 1, 3, 5, 7, 11, 17, 27 and 43, respectively.

4.5 Summary of the Operator

By identifying the image's key value, the operator is able to compress its dynamic range into a displayable one. A simple spatially-uniform function can be used to non-linearly compress the luminance range, but this mapping only preserves details in low-contrast areas. A spatially-varying filter that simulates *dodging-and-burning* in photography effectively enhances perceptive detail, using center-surround Gaussian kernels derived from Blommaert and Martens' brightness perception model (BLOMMAERT; MARTENS, 1990). The choice of an appropriate scale for the kernel allows contrast to be locally adapted in regions bounded by large contrasts, preventing the occurrence of halo artifacts thus enhancing perceptual contrastive details in the resulting images. However, this process requires the computation of a set of differences of Gaussian-filtered luminance images, which is a computationally expensive operation. Figure 4.12 shows some examples of images that were produced using the photographic operator.

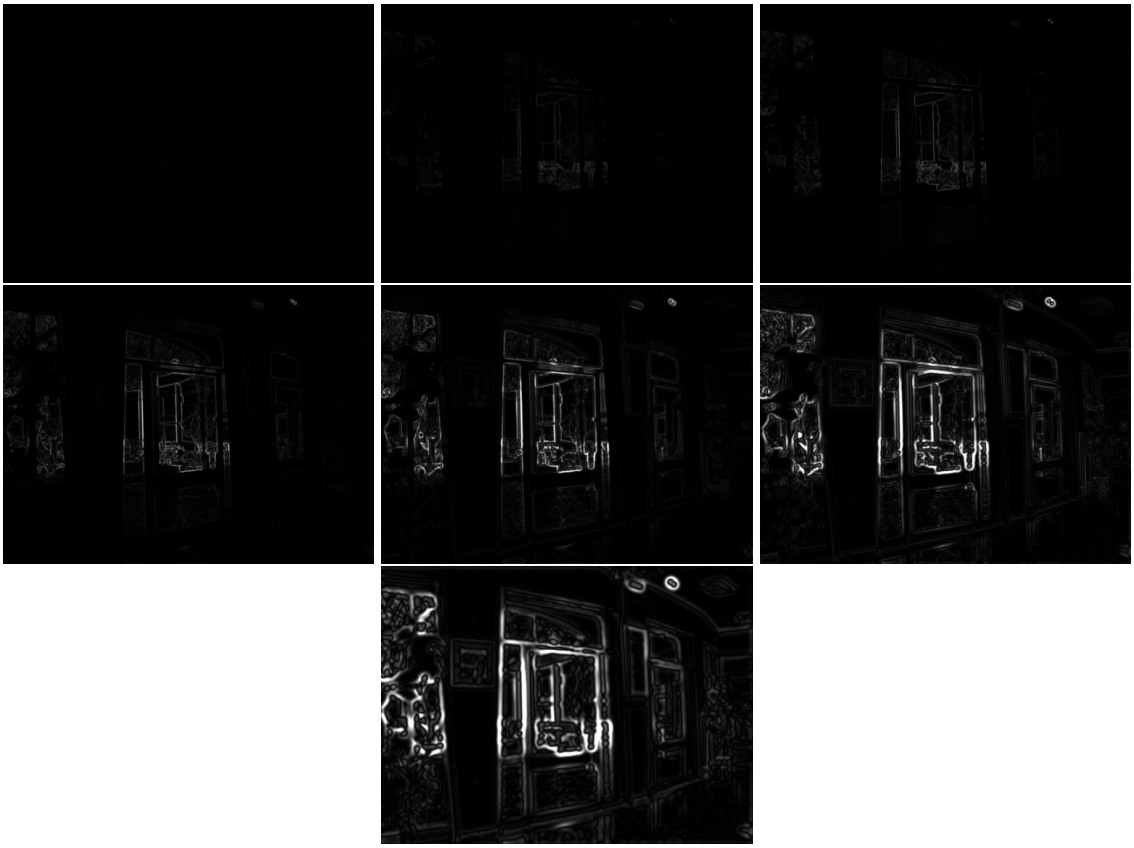

Figure 4.9: The set of normalized differences according to Equation 4.11. The first image is the normalized difference of the first and second Gaussian-filtered images from Figure 4.8, while the second one represents the difference of the second and third images from Figure 4.8, and so forth.

4.6 Photographic Operator on GPU

The global version of the operator requires only the relative luminance image (Equation 4.7). The key value of the image (Equation 4.5) can be computed on the GPU using mip-mapping (GOODNIGHT et al., 2003). The remaining of the global operator can be implemented as a single additional rendering pass. Implementations of the global operator in GPU are available in common graphics SDK (Microsoft Corporation, 2004; NVIDIA Corporation, 2005). However, a fast and plausible algorithm to evaluate its spatially-varying (*i.e.* local) operator remains as a challenge.

Further research provided approaches to accelerate the local operator also with the aid of modern programmable graphics hardware. Goodnight *et al.* implemented 2D Gaussian convolutions using a two-pass 1D convolution method (GOODNIGHT et al., 2003). Their approach accelerated the operator, but interactive rates were possible only when using a limited number of adaptation zones (two zones, against the eight zones used by Reinhard *et al.*). The use of such a limited range of adaptation zones is very similar to the use of the global operator. The authors have also included a time-dependent luminance adaptation in a way that introduces a minimal performance overhead. Temporal luminance adaptation is required when dealing with real-time applications or video.

Krawczyk *et al.* reduced the cost of the Gaussian convolutions implementing an approximation of Goodnight *et al.* technique (KRAWCZYK; MYSZKOWSKI; SEIDEL,

Figure 4.10: The choice of the proper scale for each pixel. At the left, a false color representation of the size of the local neighborhood (s_{max}) for each pixel. They are obtained by the evaluation of Equation 4.12 over the set of normalized difference images from Figure 4.9. Dark pixels means smaller isoluminant neighborhoods. At the right, the corresponding estimation of local contrast for each pixel ($V(xy, s_{max})$). This image will be used to perform the tone-mapping, accordingly to Equations 4.8 and 4.3.

2005). Their method consists of down-sampling the relative luminance images to $1/4$, $1/16$, and $1/64$ of the original size and convolving them using smaller approximate Gaussian kernels. The filtered images are up-sampled back and used to compute the normalized differences $W(x, y, s_i)$. This approach significantly speeds up the algorithm, but has some inherent constraints: the down-sampling blurs high-contrast edges and the up-sampling that follows implies additional blurring. The occurrence of excessive blurring across the high contrast edges tends to introduce clearly noticeable halo artifacts. Their approach is also memory intensive, requiring, in an optimal implementation, five times the required memory to store the input image.

In the remaining of this work, a novel approach to accelerate the local photographic operator will be presented. The method uses summed-area tables (CROW, 1984) to approximate the costly Gaussian-filtering process, providing results that are very similar to the ones produced by the original operator. The performance is up to two to ten times faster than existing implementations. The proposed technique is not tied to a limited number of local adaptation zones, is less prone to halo artifacts and has a lower memory requirements than existing methods. Before entering in details on the new approach, Chapter 5 offers a survey on summed-area tables generation, presenting the technique that will be used to perform box-filtering.

Figure 4.11: The resulting tone-mapped image using the local variant of the photographic operator. The luminance of each pixel is evaluated with Equation 4.8, using its corresponding estimation of local contrast $V(x, y, s_{max})$ from Figure 4.10. Note that the local operator enhances more contrastive details when compared with its global variant from Figure 4.6.

Figure 4.12: A plate of resulting images obtained from the use of the photographic tone reproduction operator. Values inside the yellow boxes represent the dynamic range of the original images. Source: (REINHARD et al., 2002).

5 SUMMED-AREA TABLES

Summed-area tables (SATs) were introduced in computer graphics by Crow as a texture filtering technique (CROW, 1984). His method proved to be useful to efficiently simulate glossy reflections, depth of field, transparency effects and soft-shadows (HENSLEY et al., 2005; LAURITZEN, 2007). The use of a summed-area table for filtering is quite simple and, for invariant data, a SAT can be computed offline. However, for variable data, typically in real-time and dynamic environments, new SATs must be computed on-the-fly and the key for its effectiveness lies in the time required to generate them.

A SAT is a *cumulative table*, where each of its cell corresponds to the sum of all elements above and to the left of its location in the original table. For a given input table T with dimensions $w \times h$, its corresponding summed-area table S has the same dimensions and is computed as follows:

$$S(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} T(i, j) \tag{5.1}$$

for all $1 \leq x \leq w$ and $1 \leq y \leq h$, as illustrated in Figure 5.1.

input table						summed-area table					
1	3	0	2	1	2	1	4	4	6	7	9
3	2	4	3	6	0	4	9	13	18	25	27
0	5	1	1	5	3	4	14	19	25	37	42
2	2	3	3	7	2	6	18	26	35	54	61
4	2	8	4	2	5	10	24	40	53	74	86

Figure 5.1: A table of dimensions 6×5 (left) and its corresponding summed-area table (right). Each cell of a SAT is the sum of all elements to the left and above from it in the original table, as can be seen in the highlighted blue cell on the SAT, where is the sum of all the highlighted ones in the original table.

Once a SAT is generated, it can be used to sum or filter areas on the original table. The most remarkable feature is that, for *rectangular regions*, this can be done by performing only four lookups in the SAT. A rectangular region R on T can be filtered using the following equation (see Figure 5.2):

$$Filter(R) = \frac{Sum(R)}{R_{width} \cdot R_{height}} \tag{5.2}$$

where $Sum(R)$ is defined as:

$$Sum(R) = A(R) - B(R) - C(R) + D(R) \tag{5.3}$$

and the key cells $A(R)$, $B(R)$, $C(R)$ and $D(R)$ are obtained as follows:

$$A(R) = S(R_{xmax}, R_{ymax}) \tag{5.4}$$

$$B(R) = S(R_{xmin-1}, R_{ymax}) \tag{5.5}$$

$$C(R) = S(R_{xmax}, R_{ymin-1}) \tag{5.6}$$

$$D(R) = S(R_{xmin-1}, R_{ymin-1}) \tag{5.7}$$

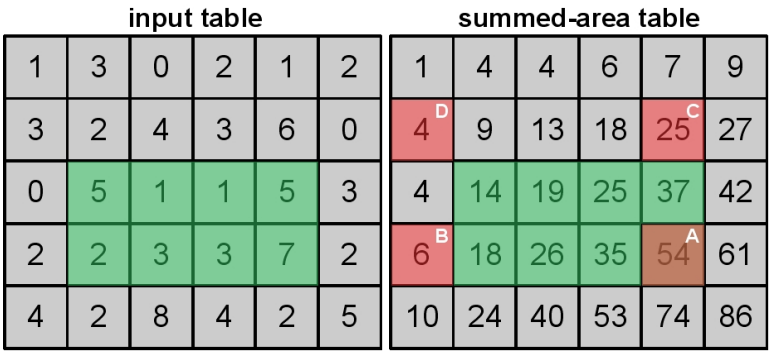

Figure 5.2: Example of filtering with a SAT. A rectangular region (in green) of a table (left) can be filtered using only four cells (red cells A, B C and D) of its corresponding SAT (right). In this example, the filtering result is $\frac{54-6-25+4}{4 \cdot 2} = \frac{27}{8} = 3.375$.

When a region has boundary cells that lie outside the limits of the table, it must be adjusted. Lower and right bounds of R are clamped accordingly to the lower-right bounds of T, as well as for the upper-left bounds, producing an adapted region R', as defined below. In practice, a region R is always clamped before being used to perform filtering (in most of cases, $R' = R$).

$$R'_{xmin} = max(R_{xmin}, 1) \tag{5.8}$$

$$R'_{ymin} = max(R_{ymin}, 1) \tag{5.9}$$

$$R'_{xmax} = min(R_{xmax}, w) \tag{5.10}$$

$$R'_{ymax} = min(R_{ymax}, h) \tag{5.11}$$

Note that even after being adjusted, the boundary cells $B(R')$, $C(R')$ and $D(R')$ may still lie outside the upper-left bounds of S, in some cases. When such happens, it is ensured that they exceeds the bounds in a maximum distance of 1. This behavior is desirable and, for such cells it is assumed that they have zero value. Refer to Figure 5.3 for an example of a filtering that exceeds the boundaries of a table.

Another interesting feature of a summed-area table is that it can effectively replace the entire input table. To recover the value of a specific cell of the original table from its corresponding SAT, one just needs to filter that cell's unit region on the SAT.

Summed-area tables offer an elegant solution to filter rectangular regions in constant time ($O(1)$), and the challenge is to find a fast way to compute the SATs. Fortunately, there is a class of problems related to parallel processing that uses a similar build: a SAT can be seen as a bi-dimensional generalization of an one-dimensional operation called *inclusive prefix sum*.

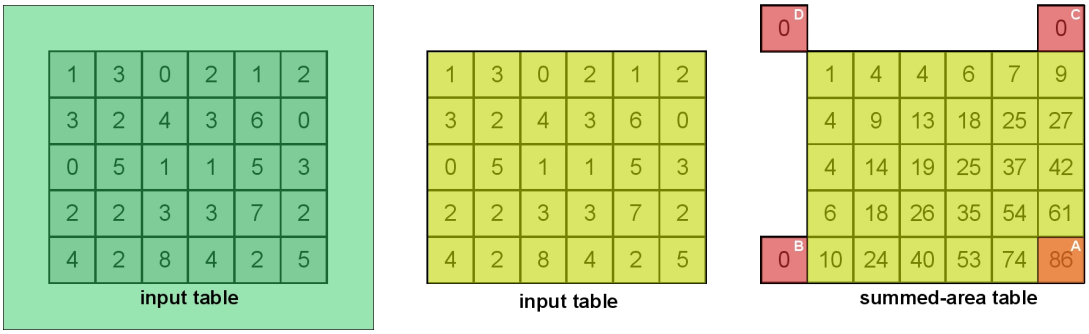

Figure 5.3: An example of filtering an area that exceeds the dimensions of the table. The green area (left) is clamped to the yellow one (middle), respecting the dimensions of the table. On the SAT (right), the key cells B, C and D are permitted to lie outside the bounds and, when this happens, they are assumed to have zero value.

5.1 Prefix Sum

Prefix sum comprises a versatile building block for parallel algorithms, as well as a good example of a computation that seems inherently sequential, but for which there is an efficient parallel algorithm (BLELLOCH, 1990). A prefix sum can be either inclusive (*all-prefix-sum* or simply *scan*) or exclusive (*prescan*), and there are several applications that benefits from it, such as lexical analysis, regular expressions, polynomial evaluation, radix sort and quicksort, to cite a few (BLELLOCH, 1990; HARRIS, 2007).

For a given input array A with n elements and an associative operator $\oplus$:

$$A = [a_1, a_2, ..., a_n] \tag{5.12}$$

a scan returns an array such as:

$$I = [a_1, (a_1 \oplus a_2), ..., (a_1 \oplus a_2 \oplus ... \oplus a_n)] \tag{5.13}$$

and assuming that $\oplus$ has a neutral element ω, a prescan computes:

$$E = [\omega, a_1, (a_1 \oplus a_2), ..., (a_1 \oplus a_2 \oplus ... \oplus a_{n-1})] \tag{5.14}$$

Note that a scan can be derived from a prescan in two ways: shifting all the elements to the left by one and placing at the end the operation between the last element of the input array and the last element of the prescan; or simply by operating each element of the prescan with its corresponding element in the input. Analogously, a prescan can be derived from a scan by shifting all its elements to the right and inserting the neutral element at the beginning.

Although a prefix sum can be performed with any associative operator, it is more intuitive to explain and illustrate the algorithms assuming the addition operator on $\mathbb{R}$. Moreover, since a summed-area table is a grid of accumulated elements, any other operator would be out of the scope of this work. Thus, compact expressions can be written:

$$I(x) = \sum_{i=1}^{x} A(i) \tag{5.15}$$

$$E(x) = \begin{cases} 0 & \text{if } x = 1 \\ \sum_{i=1}^{x-1} A(i) & \text{if } 1 < x \leq n \end{cases} \tag{5.16}$$

For a real example, recall the input table from Figure 5.1. Let its first row to be the input array. Its corresponding inclusive and exclusive prefix sums, with respect to Equations 5.15 and 5.16, respectively, are illustrated in Figure 5.4.

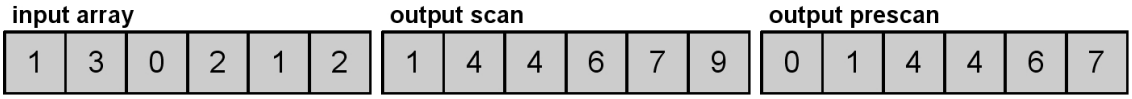

Figure 5.4: Examples of prefix sums on an array. The leftmost is the input array. At middle, its corresponding inclusive prefix sum (scan), followed by its corresponding exclusive prefix sum (prescan) at right.

The most straightforward way to perform a prefix sum is using a *brute force* algorithm, solving Equation 5.15 or 5.16 for each element of the array. This is a rather inefficient choice, since it is easy to figure out that this leads to a $O(n^2)$ algorithm. Efficient parallel algorithms exist and they can be mapped to a GPU implementation. Basically, there are two algorithmic patterns to assist the operation: *recursive doubling* and *balanced trees*. The former is $O(n\ log(n))$, while the later is $O(n)$ but requires twice the number of steps. Additionally, the later is said to be *naturally exclusive*, or in other words, produces only a prescan. Refer to Appendix A for a complexity analysis of the algorithms.

5.1.1 Recursive Doubling Overview

Relying on the recursive doubling pattern, a parallel gather operation amongst an array with n elements is performed in $log(n)$ steps (DUBOIS; RODRIGUE, 1977). Each step consists on updating a number of elements that will then be used in the next step. The best way to introduce the algorithm is with an illustrative example, such as the one of Figure 5.5.

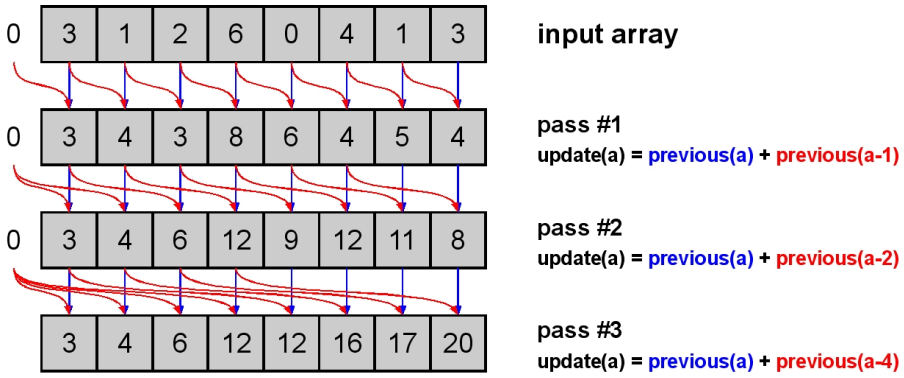

Figure 5.5: An example of prefix sum using the recursive doubling pattern. At each step, two elements are retrieved from the previous one and accumulated. The first parcel (blue arrow) has the same index of the element currently being computed. The second parcel (red arrow) is distanced to the former by an offset that grows as the passes are executed. In a typical recursive doubling, the offset starts at 1 and is doubled at each step. Any reading outside the bounds of the array is assumed to has zero value.

Figure 5.5 shows that, at each pass, all the elements of the array reads two values from the previous pass and performs one addition operation. For a given element, the first retrieved element has its same index (blue arrow), while the second one is distanced to the left by an offset (red arrow). The offset increases with the passes, and when an

offset falls out of the array, it is assumed to be zero. From the example, one can conclude that, for some pass i, the offset is 2^{i-1}. Another look at Figure 5.5 also suggests some improvements. As can be seen, at each pass i, the first 2^{i-1} elements are already computed and do not require to be updated anymore. Processing such elements again will only imply in the addition of them with zero. One can then just focus on updating the remaining $n - 2^{i-1}$ elements. Figure 5.6 illustrates this.

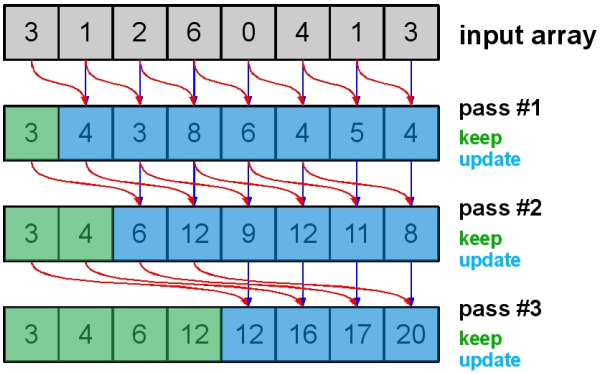

Figure 5.6: An improved execution of the example in Figure 5.5. Only the latest elements (blue) require to be updated, while the remaining (green) are already computed.

Typically, the recursive doubling accumulates only two elements per update (DUBOIS; RODRIGUE, 1977; HENSLEY et al., 2005). However, the method is not tied to this assumption and can be generalized to perform more accumulations per update. Accumulating $k \geq 2$ elements, leads to the following:

- a total of $\lceil log_k(n) \rceil$ passes
- for each pass i:
  - the first k^{i-1} elements are already computed and are kept unchanged
  - and the latest $n - k^{i-1}$ elements must be updated
- for each updated element:
  - a total of k elements are retrieved from the previous step
  - the offset between the samples is given by k^{i-1}
  - and $k - 1$ addition operations are performed

From the statements above, one can conclude that a higher number for k might improve the performance, since it lowers the number of steps as well as the overall number of updated elements. However, it raises the number of reads per update and the number of addition operations. An optimal value for k depends on the array size and on features of the intended platform, such as the number of processors and concurrent memory access latency.

The recursive doubling approach requires only one caution: for a given pass, all the elements of the array must be read before they can be effectively updated; otherwise, the output will be unstable. In parallel computing terms, this implies in a *synchronization* process amongst the processors; in GPU terms, things get more complicated, as described in Section 5.1.2. Assuming that such caution was taken, at the end of the execution of

the algorithm the input array is replaced by its corresponding scan. Recursive doubling can be adapted to perform a prescan instead of a scan (avoiding further conversions). For such a goal, the first element of the input array is replaced by the neutral element and the algorithm is kept unchanged. The complexity of the recursive doubling prefix sum algorithm is $O(n\ log(n))$ (see Appendix A).

5.1.2 Recursive Doubling on GPU

Just as before, a GPU can accumulate results so that $log(n)$ passes are need for a prefix sum using the recursive doubling pattern (HENSLEY et al., 2005). However, a GPU-based implementation requires extra caution: the input array is stored in texture memory and it is not possible to read and write simultaneously on such memory.

The solution comes with *additional* memory. Using two arrays, the input and an auxiliary one, each of them updates the required elements by retrieving data from the other one. When a pass finishes, the two arrays are swapped and the algorithm continues. Due to this behavior, the data stability between both arrays must be ensured before starting a new pass. In the original approach in Section 5.1.1, the algorithm works on a single array, and the first 2^{i-1} elements of a step i are simply ignored since they are already computed. However, on a GPU implementation, *both* arrays must stay synchronized, otherwise the algorithm leads to unstable results, as shown in Figure 5.7.

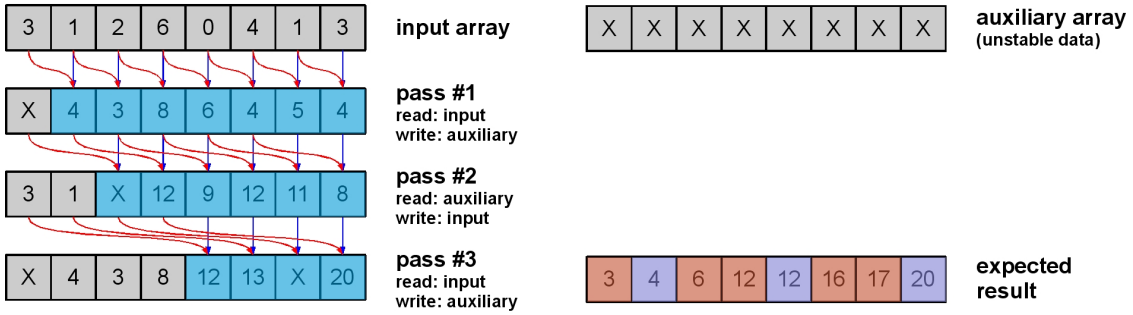

Figure 5.7: An example of an unstable execution of the recursive doubling on GPU. In the beginning, just the input array has stable data (unstable data are X-valued cells). As the passes proceed and the arrays are swapped, unstable data of the auxiliary array are mixed with stable data, thus compromising the result. At the bottom right, the expected prefix sum output. Values highlighted in red represent the misleading results.

To prevent data instability, some values must be copied from one array to another before the pass switching. One way to do this is to *always* update *all* the elements of the array (recalling to a similar case to the one of Figure 5.5), which is not an attractive solution. A more efficient alternative is to identify the elements that require synchronization and update them together with the remaining. As shown in Figure 5.8, in a given pass i, the first $\lfloor 2^{i-2} \rfloor$ elements are already synchronized in both arrays and the latest $n - 2^{i-1}$ require update (highlighted in blue). The remaining elements (in green) then require to be synchronized along with the updates.

Note that, a the end of the execution of the algorithm, the input array may not contain the prefix sum. Since the arrays are swapped at each step, the last one might write to the auxiliary one. When such happens, one can copy the latest updates to the input array. To avoid these copies, another choice is to warn the caller that the required prefix sum lies in another array, returning an identifier.

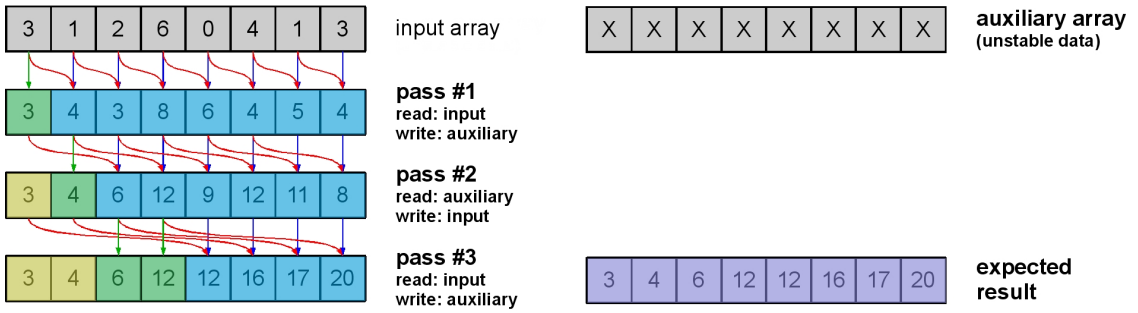

Figure 5.8: A work-efficient and stable execution of recursive doubling in GPU. Elements marked in yellow are already synchronized in both arrays and are kept unchanged. Green elements must be updated along with the remaining (blue cells). As can be seen, the resulting array matches with the expected at bottom-right.

The GPU implementation can also be generalized to accumulate more elements per update. Assuming a value $k \geq 2$, the summary of the algorithm is similar to the one of Section 5.1.2, but with the following requirements:

- An additional array of size n
- Each pass i preforms a swap between the arrays, where:
  - the first $\lfloor k^{i-2} \rfloor$ elements are kept unchanged
  - the latest $n - \lfloor k^{i-2} \rfloor$ elements require update

Hensley *et al.* stated that the optimal value for k (in a GPU-based implementation) is related to the context switching overhead between passes, texture fetching latency and texture cache accuracy (since offset sizes increase exponentially) of the intended hardware (HENSLEY et al., 2005).

As programmable graphics hardware continues to evolve from rendering-oriented to a more parallel computing-oriented tasks, constraints regarding texture memory access tend to disappear (NVIDIA Corporation, 2007). However, this is not the case yet and the vast majority of commodity GPUs available today does not support such advanced architectures. And even those that support require specific drivers that still behaves unstably. For a GPU, CUDA-based implementation of recursive doubling, refer to Harris's report (HARRIS, 2007).

5.1.3 Balanced Tree Overview

Blelloch presented a novel parallel algorithm to perform prescan (BLELLOCH, 1990). A scan operation can be derived from a prescan using one of the methods described in Section 5.1. Blelloch replaced the recursive doubling building block by a more work-efficient one: a balanced binary tree. Basically, the input is submitted to a reduction process and its intermediate results are used later to generate a new tree, in a *bottom-up-and-top-down* fashion. The resulting leaves of the later tree correspond to a prescan array. A binary tree with n leaves has $\lceil log_2(n) \rceil + 1$ levels, and each of its nodes can have up to 2 children. This leads to $O(n)$ cost for both reduction and expansion stages (refer to Appendix A for details).

The process starts with a reduction stage, which consists on accumulating two nodes of the previous level. The leaves of the reduction tree are nothing but the input array.

Each level of the tree then corresponds to a set of partial sums on the input. The last level will be a single node (the root) comprising the sum of all elements of the input array. A complete execution of the reduction stage is illustrated in Figure 5.9.

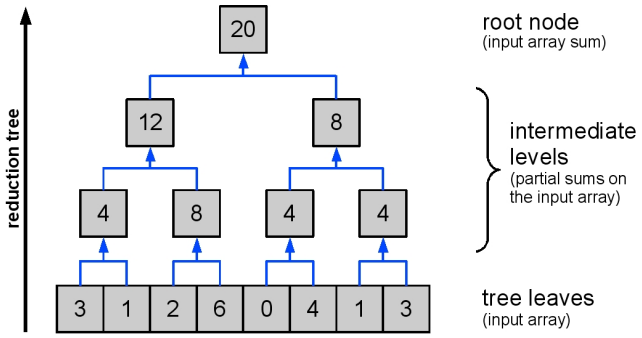

Figure 5.9: The reduction phase of the binary balanced tree pattern. Starting from the leaves (the input array), they are accumulated and stored in a parent node. The process repeats until the root is generated, containing the sum of all of the elements of the input array. The entire tree is called a reduction tree.

As for the expansion stage, the root node is replaced by the neutral element. For each remaining level of the expansion tree, the parent's value propagates to its left child, and the right child is evaluated as the accumulation of the parent's value with the value of its left sibling node on the reduction tree. The process can be a little confusing, so a complete execution is illustrated in Figure 5.10.

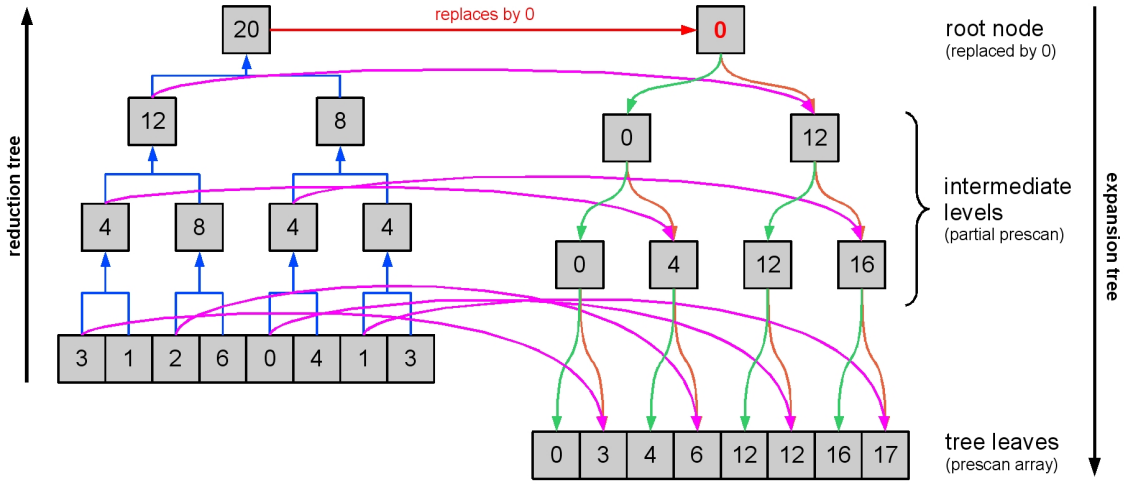

Figure 5.10: The expansion phase of the balanced binary tree pattern. The root node is replaced by the neutral element (zero). From the root node, two new nodes are generated. The root's value is propagated to its left child (green arrow), while the right child is the sum of the root's value (orange arrow) and its left sibling (pink arrow) on the reduction tree. The process is repeated for each intermediate node until the entire tree is produced. The leaves of the last level comprise a prescan on the input array.

Note that Blelloch's algorithm also requires caution. Intermediate results from the reduction stage are used by the expansion stage and the later tree can not simply reuses the former's tree nodes (the later requires reading the former's nodes, and the only exception is the root node which can be directly replaced by the neutral element). In parallel

computing terms, this requires a synchronization process to ensure data stability. Blelloch presented an efficient implementation of the algorithm that requires no additional memory, only the input array (BLELLOCH, 1990). Figure 5.11 illustrates the technique. In GPU terms, however, this implies in additional memory, as usual.

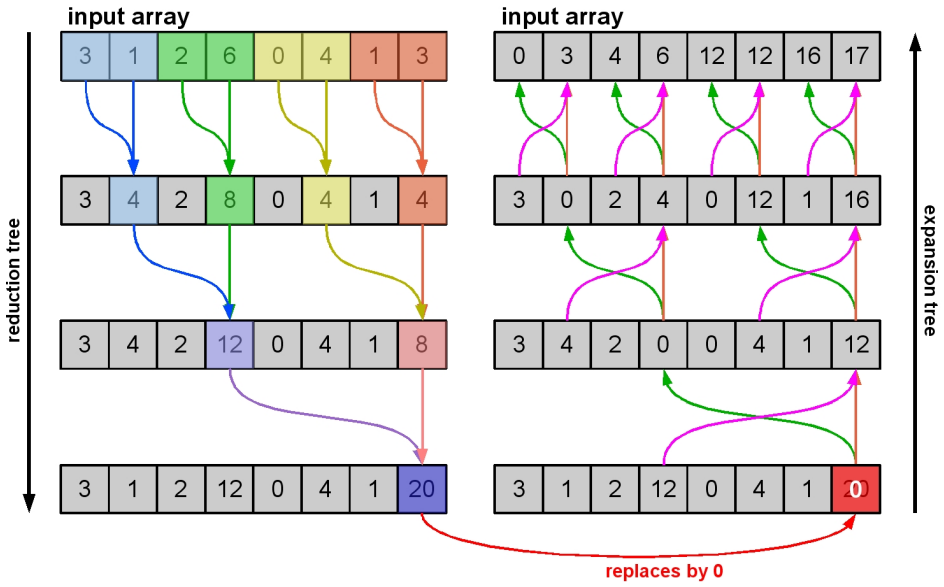

Figure 5.11: A complete execution of Blelloch's algorithm using only one array. At each step, updates are placed in special cells (marked in colors) and it is ensured that all the processors that are executing a step will not update their values until all of them allegedly retrieved the required data. In the expansion phase (right), green arrows show the propagated values, while orange and pink ones represents the parcels that are accumulated on that update.

Blelloch's algorithm is not restricted to binary trees: any tree with a maximum of k children per node can be used. The reduction pass is straight-forward, just by accumulating k nodes from the previous level. The expansion pass is more complicated: the parent node is propagated to the *leftmost child*, and *each remaining child* is the result of accumulating the parent node with *all* of its left siblings on the reduction tree. Note that the nodes of the expansion tree do not perform the same number of computation. Indexing the children nodes of a given parent with $1 \leq d \leq k$, from left to right, a total of d readings and $d-1$ operations are performed on that child. The upper bound is k readings and $k-1$ additions (the rightmost child). To summarize the method:

- Two trees with $\lceil log_k(n) \rceil + 1$ levels each are generated
  - giving a total of $2\lceil log_k(n) \rceil + 1$ passes (excluding the leaves when reducing)
- Each node of the reduction tree (except the leaves) performs
  - a total of k readings on the previous level
  - a total of $k-1$ addition operations
- Each node of the expansion tree performs
  - a maximum of k readings
  - a maximum of $k-1$ addition operations

Just like the recursive doubling case, a higher value for k might improve the performance, but for the same reasons, an optimal value for k is hard to define analytically.

Note that Blelloch's algorithm produces only a prescan. When a scan is desired, it must be derived from the prescan using one of the two methods described in Section 5.1. However, both methods require that the input be available in order to perform such conversion. Since the algorithm *destroys* the input, more memory is required to keep the input array unchanged.

5.1.4 Balanced Tree on GPU

For the same reasons of Section 5.1.2, a GPU-based implementation of the balanced tree approach requires extra caution to preserve data stability, and this basically requires the use of some additional textures. This Section provides some guidelines to assist a real GPU-based implementation of Blelloch's algorithm.

The approach requires *two* additional arrays, with the same size of the input one. For the reduction stage, in the first pass, the input is reduced to one of the auxiliary arrays. In the second pass, the remaining auxiliary array reduces the data obtained from the previous pass, with an increasingly stride. For the remaining passes, only the auxiliary arrays are swapped (the input is kept unchanged, since it will be required in the expansion stage), the stride increases, and the data is reduced. The reduction phase is illustrated in Figure 5.12.

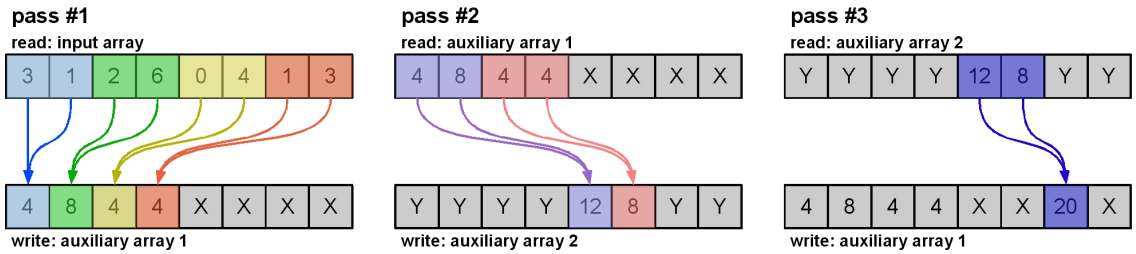

Figure 5.12: The reduction stage on a GPU implementation, using two auxiliary arrays. In the first step, the input array is read and, in the remaining ones, only the auxiliary arrays are used. Updates are performed as usual, but at each step, a stride is applied. Note that, at the end of the reduction phase, some values of the arrays still contain unstable data (X and Y-valued cells). These *gaps* will be filled in the expansion stage.

The expansion stage starts by replacing the root node by zero (note that due to this replacement, the last pass of the reduction tree can be skipped). The algorithm then continues the expansion operations, swapping the arrays at end of each step. As can be seen in Figure 5.13, the *gaps* produced by the reduction stage are filled by the expansion nodes.

The last step was omitted in Figure 5.13 and there is a reason for such. The last expansion pass requires reading from both input array and the last written array. The remaining auxiliary array reads their values and, finally, holds the prescan. This is illustrated in Figure 5.14.

From the examples given, one should note that the last element of both auxiliary arrays was not used. This is because a full balanced binary tree has $n - 1$ intermediate nodes. However, they can not be simply removed since the last pass must have space to store all the prefix sum elements. A possible optimization is to save memory in one of the arrays and ensure that the last pass will write on the larger one.

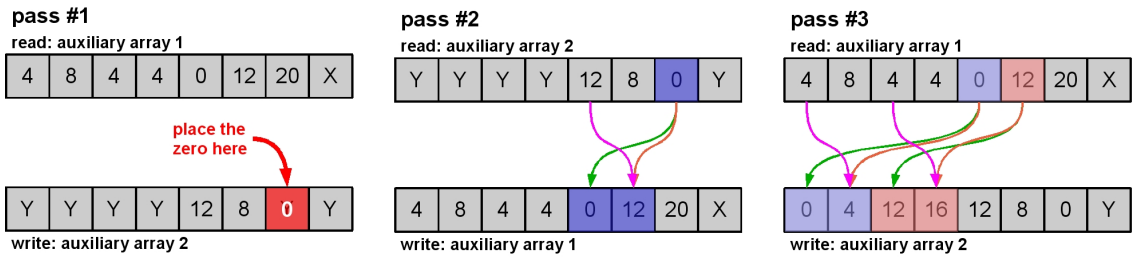

Figure 5.13: The expansion stage on GPU. First, the zero is placed in the swapped array. At each remaining step, elements with odd indices receive a copy of their corresponding parent node (green arrows). Even elements are the accumulation of their parent's value (orange arrows) with their left siblings in the reduction phase (pink arrows). Swapping is performed at the end of each step, as usual. The last one was omitted because it is a special case.

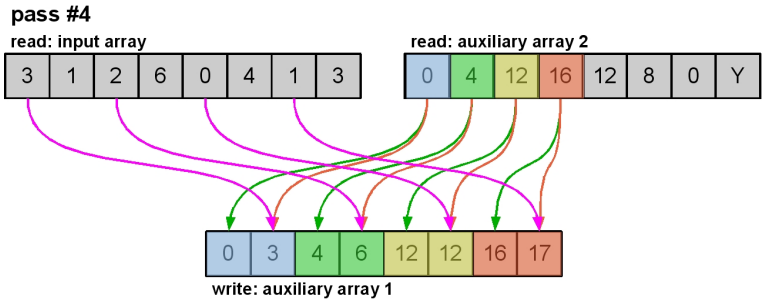

Figure 5.14: The last step of the expansion stage. The input array is read together with the auxiliary one. Odd elements are propagated from their parent's value (green arrows), while even elements are computed as the addition of their parent's value on the auxiliary array (orange arrows) with their left siblings on the input (pink arrows).

Analyzing the statement above, a bigger value for k can potentially reduce the memory usage, since less intermediate nodes are produced. Similarly to the case of the binary tree, one of the arrays just needs to have storage for the intermediate nodes, while the other one must have size n, thus ensuring that the last pass will write to the later. For instance, from a given input array with size $n = 25$ and $k = 5$, a total of 6 intermediate nodes are produced on each tree. One array of size 6 and another of size 25 are sufficient to a full execution of the algorithm.

The algorithm's summary is similar to the one of Section 5.1.3, with the following additional requirements:

- An additional array with size n
- An additional array with size $\sum_{i=1}^{\lceil log_k(n) \rceil} \left\lceil \frac{n}{k^i} \right\rceil$
- A total of $2\lceil log_k(n) \rceil$ array swappings
- For each reduction pass i:
  - a stride is applied
  - a total of $\lceil log_k(n) \rceil$ intermediate levels are produced
  - each intermediate level $1 \leq L \leq \lceil log_k(n) \rceil$ has $\left\lceil \frac{n}{k^L} \right\rceil$ nodes
- Expansion is similar to reduction, but removes strides and produces the leaves

As can be seen, Blelloch's algorithm is tricky to optimize for a GPU-based implementation. This section provided an overview on the workflow of the algorithm, but improvements regarding memory usage can be employed. The main drawback of Blelloch's algorithm is the number of swappings, the double of the required in a recursive doubling pattern. However, it is still faster since it reduces considerably ($O(log(n))$) the amount of processing. Another drawback is the need to derive the scan from the prescan.

5.2 Deriving SATs From Prefix Sum

As stated in the beginning of this chapter, a SAT can be seen as a bi-dimensional scan operation. Relying on a scan algorithm (*inclusive prefix sum*), a summed-area table can generated in two phases. First, a scan is performed on each row of the input table, as shown in Figure 5.15:

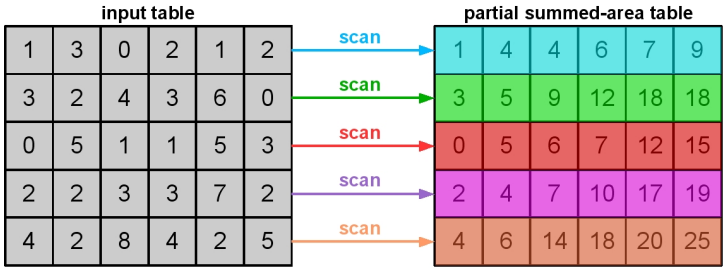

Figure 5.15: From the input table of Figure 5.1, the first step to generate its summed-area table is to perform a scan on each of its rows. The resulting table is said to be a partial summed-area table.

and then another scan is applied on *each* ***resulting*** *column*, as can be see in Figure 5.16:

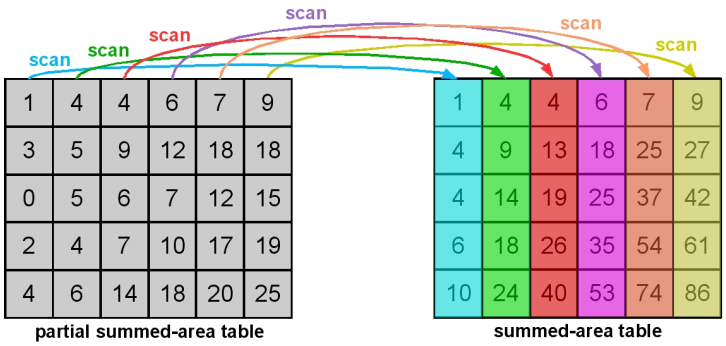

Figure 5.16: The full summed-area table of the input table in Figure 5.1 is generated by applying a scan operation on each column of its partially generated one from Figure 5.15.

Similarly, one could think in scan the columns before the rows. The results are the same, as well as the amount of computation, so the convention used is a matter of convenience. A more formal conclusion for this multi-step approach can be obtained by matching Equations 5.1 and 5.15. Rearranging the order of the sums in Equation 5.1, one has:

$$S(x,y) = \sum_{j=1}^{y} \left[\sum_{i=1}^{x} T(i,j) \right] \tag{5.17}$$

and since the sum inside the brackets is fixed to a specific row j, it reduces to an intermediate set of scan operations on (horizontal) one-dimensional arrays (Equation 5.17), conveniently expressed as:

$$I(x, j) = \sum_{i=1}^{x} T(i, j) \tag{5.18}$$

and plugging the above expression back to Equation 5.17 gives:

$$S(x, y) = \sum_{j=1}^{y} I(x, j) \tag{5.19}$$

and since x is now fixed and only j varies, this reduces again to another set of scan operations on (vertical) one-dimensional arrays.

Not only prefix sums on arrays are useful to generate summed-area tables, but the inverse process might be interesting as well, when there are restrictions concerning the maximum size of an array. As one of the contributions of the present work, a simple algorithm to produce the prefix sum of large input arrays is presented, using the graceful arrangement of partially generated SATs. Details are presented in Chapter 6.

5.3 Summed-Area Table Generation on GPU

In order to generate SATs on GPU, for each pass, a screen-aligned quad is rendered. This region must cover all pixels that require to be updated. To avoid the need of any conditional statements in the shaders to deal with fetches outside the bounds of the textures, their samplers can be set to use *clamp to border color* mode with the border color set to 0, or rendering a black border manually around them and activating *clamp to edge* mode.

Specifically for the case of the balanced tree approach, after finishing a phase, the produced results must be converted from a prescan to a scan, and this procedure has to be applied for both the horizontal and vertical phases. Some computational effort could be saved if a way to directly perform a scan was available. For such a goal, a modification on the algorithm is presented in Chapter 6.

Summed-area tables may suffer from precision constraints, since they are monotonic functions. As the indices grows, their values grows as well, and the floating-point representation becomes more prone to loss of precision. Artifacts start to appear when computing the difference between relatively large finite precision numbers with very close values on the SAT, as can be seen in the middle column of Figure 5.17.

To avoid such loss of precision, Hensley *et al.* proposed a simple and efficient solution. It basically consists on centering each pixel value based on the average value of the entire image, *i.e.*, subtracting the average of the image from each of its pixels. This modification benefits precision in two ways: there is a 1-bit gain in precision because the sign bit now becomes useful; and the summed-area function becomes non-monotonic, and therefore the maximum value reached has a relatively lower magnitude (HENSLEY et al., 2005). Results of their improvement are shown in the last column of Figure 5.17.

5.4 Discussion

Computer graphics techniques that benefits from summed-area tables tend to rely on recursive doubling algorithms (HENSLEY et al., 2005; LAURITZEN, 2007). The reasons behind the choice for such pattern by the authors remains unclear since Blelloch

Figure 5.17: Example of precision loss on summed-area tables. In the first column, the original table (image). The middle column is a reconstruction of the original image using its corresponding summed-area table. The last column is an improved reconstruction that minimizes the precision loss. Bottom images are zoomed versions of the top images. All SATs were built using 24-bit floats. Note that the artifacts appear in the top-right portion of the image, instead of the bottom-right. This is because, in OpenGL, textures are parametrized from left to right and bottom to up, so the resulting SAT is flipped vertically. Source: (HENSLEY et al., 2005).

presented a better work-efficient algorithm in early 1990 based on a balanced tree pattern (BLELLOCH, 1990).

One possibility is that recursive doubling was used since it can immediately produce a scan, contrasting with Blelloch's approach which only produces a prescan. Another possibility lies in the fact that Blelloch's prescan requires more memory and more steps in a GPU-based implementation. However, these facts do not justify their decisions, since a balanced tree prefix sum computes $O(log(n))$ less elements than recursive doubling. Actually, neither Hensley *et al.* nor Lauritzen mention the existence of Blelloch's algorithm or any related work (HENSLEY et al., 2005; LAURITZEN, 2007).

It seems that only recently Blelloch's prescan algorithm was used in computer graphics, being introduced by researchers working on GPGPU (SENGUPTA; LEFOHN; OWENS, 2006; SENGUPTA et al., 2007), and in CUDA (HARRIS, 2007). Harris also presented implementations for recursive doubling and enforced the superiority of the balanced-tree approach (HARRIS, 2007).

Harris provided an API on CUDAPP to perform multi-scan, so that it can be used to generate SATs (HARRIS, 2007; HARRIS et al., 2008), but it also requires to lately convert them from a prescan to a scan. Also, his multi-scan algorithm is designed to deal with horizontal arrays only. For the summed-area table case, after the first multi-scan on its rows, there is a need to rearrange the resulting columns in rows, performs the second

multi-scan, and then rearrange them back to the original placement. Unfortunately, Harris did not provided results for the summed-area table case on his work (HARRIS, 2007).

One contribution of this work is an alternate method to perform scan operations on the balanced tree pattern without the need of deriving them from a prescan ones. Although it does not reduces the overall complexity, it can save some computation and improve performance. Details of these novel approaches are available in Chapter 6.

6 IMPROVEMENTS ON PREFIX SUM ALGORITHMS

This chapter introduces an improvement to the balanced tree approach for prefix sum. It basically consists in an alternate way to produce a scan without the need of deriving it from a prescan. This improvement can benefit existing techniques that rely on summed-area tables or prefix sum, including the proposed approximation for the photographic tone reproduction operator, presented in Chapter 7. This chapter also introduces an alternate method to generate prefix sums for large arrays, derived from partially generated summed-area tables. This is useful when constraints are imposed on the maximum size of the input array.

6.1 Addendum

The improvement to the balanced tree algorithm for prefix sums described in Section 6.2, although initially thought to be a novel, efficient approach to the problem, was later found to be analogous to a procedure already available in the literature. Unfortunately, it was not possible to track the proper reference to this publication, but a photocopy of the document follows attached to this thesis in Appendix D. This fact was only noticed after the results of the thesis were already officially presented to and evaluated by the examination committee.

In short, the reduction stage is identical for both strategies, diverging only during the expansion stage. While the proposed approach (Section 6.2) performs subtractions over the partial sum sets originated from the reduction stage, the algorithm in Appendix D guides the expansion stage by performing additions over the same partial sum sets. The net result, algorithm complexity, number of read/write and arithmetic operations, however, remain the same.

Curiously, further investigations have shown that the proposed algorithm (Section 6.2) tends to run slightly faster than a GPU-based implementation of the approach described in Appendix D. However, such performance gain is only marginal, thus there are no clear advantages in favoring one algorithm over the other.

Interesting enough, the technique described in Appendix D can be seen as a particular implementation of the algorithm presented by Meijer and Akl (MEIJER; AKL, 1987). The latter produces optimal-prefix sums through a hierarchy of intercommunicating processors, while the former adapts this processor topology to a more robust pool of independent processors that share memory; the same topology used by Blelloch (BLELLOCH, 1990). It is also worth noting that Meijer and Akl's method predates Blelloch's algorihtm, even though no reference is made by Blelloch to the Meijer and Akl's work.

The remainder of the contents of this Chapter will provide an in-depth, illustrative description of the initially proposed algorithm only. Nevertheless, the information accu-

mulated in the following pages can be browsed as a reference to understand the operational semantics behind the already available, but rather unknown algorithm described in Appendix D.

6.2 Moving from a Balanced Tree Prescan to a Scan

Blelloch's algorithm (BLELLOCH, 1990) can be easily modified to perform a scan instead of a prescan. The proposed algorithm works for the addition operator, although it might work *for other associative operators that happen to have inverse operators with the same neutral element*. This work does not provides a formal proof for this conjecture.

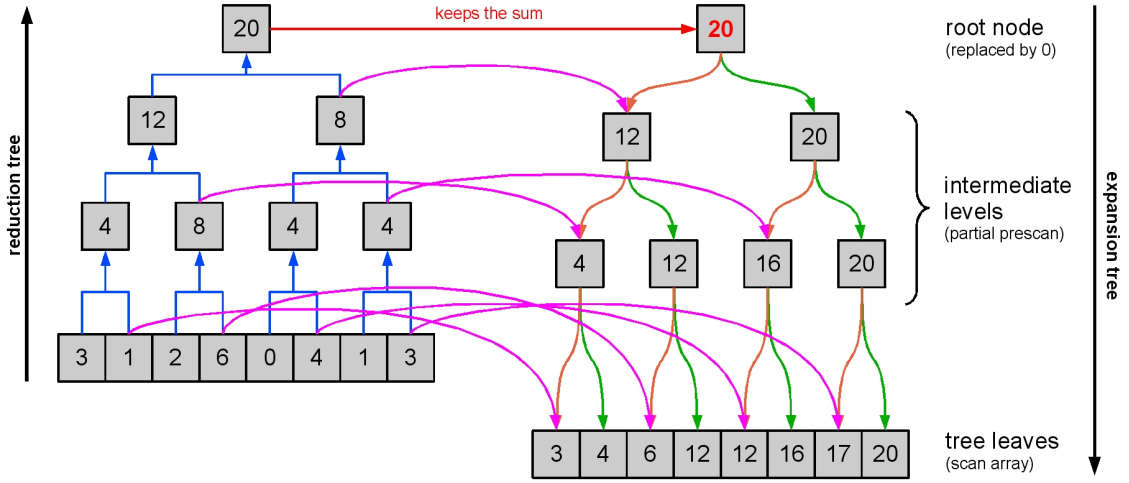

Figure 6.1: The reduction phase is as usual. As for the expansion, the root node is replaced by the root node of the reduction tree. From the expansion's root node, two new nodes are generated. The root's value is propagated to its right child (green arrow), while the left child is the subtraction of the root's value (orange arrow) with its right sibling (pink arrow) on the reduction tree. The process is repeated for each generated node until the entire tree is generated. The leaves of last level comprises a scan on the input array.

The reduction stage is performed as usual. However, the root element remains as the sum of the input array and is *not replaced* by the neutral element. The expansion stage then propagates the parent node to its *rightmost child*. Each remaining children is then evaluated as the *subtraction* of the parent node with *all of its right siblings* in the reduction tree. Figure 6.1 illustrates the steps of the algorithm.

Now, for the GPU case, reduction is performed as described in Section 5.1.4 (however, its last pass can not be skipped, since the entire sum is required in the expansion stage). For convenience, a copy of Figure 5.12 is reproduced in Figure 6.2.

The expansion stage is similar, but instead of replacing its root node by zero, it is replaced by a copy of the root node of the reduction tree. As for the remaining, the parent's value is propagated to the right child, and the left child is the result of the parent's value minus its right sibling's value from the reduction tree, as show in Figure 6.3.

The last expansion pass is performed just like before, reading both the input array and an auxiliary one, writing the results in the remaining auxiliary array, as illustrated in Figure 6.4.

With this simple procedure, one can avoid the need of converting a prescan to a scan.

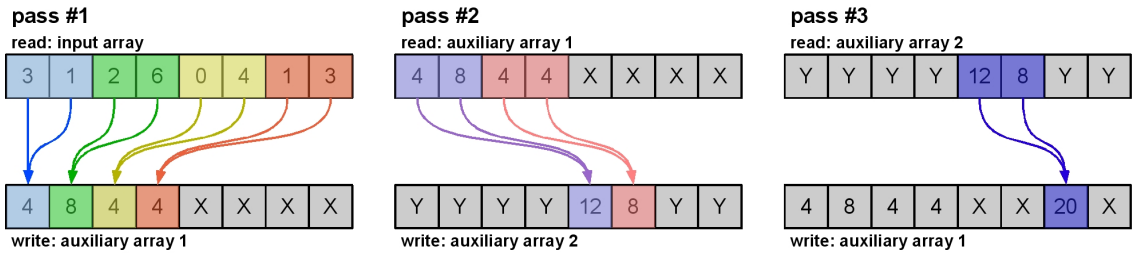

Figure 6.2: The reduction stage on a GPU implementation is the same exemplified in Figure 5.12. However, in order to perform a scan, the last pass can not be skipped.

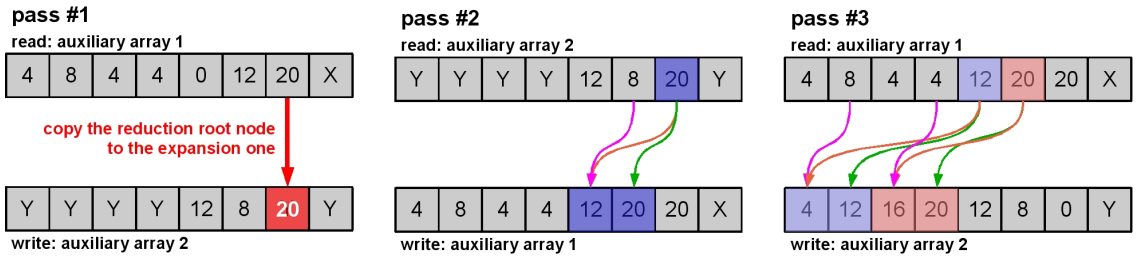

Figure 6.3: The expansion stage in GPU is similar to the one of Figure 5.13. However, instead of replacing the expansion tree root node by zero, it is replaced by a copy of the reduction tree root node. At each remaining pass, elements with even indices receive a copy of their corresponding parent node (green arrows). Odd elements are the result of their parent's value (orange arrows) subtracting their right siblings in the reduction phase (pink arrows).

Recall that, in a summed-area table, such conversions happens to both rows and columns, requiring two additional steps over its entire area. Although it is a simple modification, no reference for such a procedure was found in the literature. Like the original algorithm of the Section 5.1.3, this new method is not restricted to binary trees and can be extended for any $k \geq 2$.

Results from recursive doubling (RD), balanced tree (BT) and the proposed improvement (BT*) on the balanced tree are shown in the following tables. Such tables provides the time to generate a full summed-area table (in milliseconds) in a GeForce FX5200, a GeForce 6800GT and a GeForce 8800GTX, respectively (see the top-row of each table). All computers were running Windows XP Professional with Service Pack 2, the latest NVIDIA drivers (169.21 WHQL) and OpenGL 2.1.2. As for the shaders, they were developed in NVIDIA Cg, and the runtime version used was Cg 2.0 (January 2008). The overall speedup of the proposed improvement is about 2.7 times when compared with recursive doubling, and 1.4 times when compared with the traditional balanced tree with conversions from prescan to scans.

The main bottleneck in the performance is caused by the amount of context switching required for the multi-stage approach of the summed-area table generation. For such a reason, the balanced tree approach tends to have lower performance on small images, since it performs twice the number of context switching than recursive doubling. Note that the proposed balanced tree (BT*) algorithm significantly improves the overall performance when compared to the traditional approach that requires conversion from a prescan to a scan (BT).

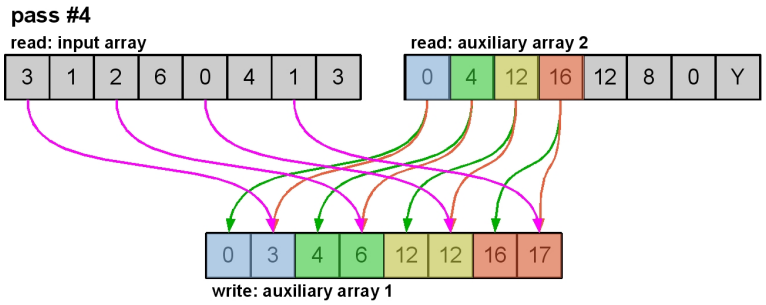

Figure 6.4: The last step of the expansion stage in GPU is similar to the one of Figure 5.14. The input array is read together with the auxiliary one. Even elements are propagated from their parent's value (green arrows), while odd elements are computed as the subtraction of their parent's value on the auxiliary array (orange arrows) with their right siblings on the input (pink arrows).

FX5200		Time (ms)					
image size		k=2	k=3	k=4	k=5	k=6	k=8
128x128	RD	4.172	4.720	5.046	6.213	5.801	7.646
	BT	6.385	5.977	5.936	6.155	6.646	7.228
	BT*	4.514	3.982	4.067	4.276	4.702	5.298
256x256	RD	18.238	21.595	20.572	24.605	29.687	30.096
	BT	16.191	16.905	17.512	19.419	20.700	24.073
	BT*	11.810	12.504	13.028	14.908	16.486	19.741
400x300	RD	37.984	39.146	44.792	46.714	55.173	56.446
	BT	29.440	29.388	30.841	33.003	36.182	42.043
	BT*	22.086	21.891	23.692	25.423	28.713	34.260
512x512	RD	86.825	87.630	100.635	103.763	122.883	126.709
	BT	61.365	63.564	66.969	72.698	78.429	90.878
	BT*	44.795	47.052	49.808	55.941	62.170	74.923
640x480	RD	104.075	102.197	117.050	133.612	143.003	167.199
	BT	69.599	71.110	73.669	79.965	89.281	102.190
	BT*	52.974	53.538	57.305	63.551	71.985	85.494
800x600	RD	173.848	174.851	187.212	214.230	229.859	296.214
	BT	111.439	109.717	115.687	127.208	139.632	161.861
	BT*	85.819	84.899	89.518	101.803	113.660	136.750
1024x768	RD	301.383	309.596	314.886	389.801	386.764	499.813
	BT	198.989	198.211	211.684	228.001	242.267	284.121
	BT*	137.019	139.190	150.716	165.457	181.608	223.527
1024x1024	RD	398.329	422.940	421.657	523.650	523.215	667.012
	BT	266.042	276.124	293.529	315.354	334.954	380.918
	BT*	181.540	196.563	211.854	233.648	251.439	296.819
1280x1024	RD	563.628	562.069	614.966	712.459	700.383	887.591
	BT	296.262	293.702	308.499	338.993	372.071	432.453
	BT*	234.464	228.797	246.796	274.688	307.084	367.447

Table 6.1: Performance results for summed-area table generation from a GeForce FX5200 with 128MB RAM installed on an AGP bus of an AMD Athlon Thunderbird 1.4GHz CPU with 256MB RAM.

6800GT		Time (ms)					
image size		k=2	k=3	k=4	k=5	k=6	k=8
128x128	RD	9.841	7.406	7.447	7.446	7.309	7.405
	BT	26.819	24.199	21.818	22.227	22.270	21.055
	BT*	14.915	12.215	10.028	9.958	9.464	9.119
256x256	RD	10.364	7.932	7.808	8.054	8.269	6.056
	BT	27.891	24.791	22.954	22.354	23.523	20.474
	BT*	15.354	12.664	10.282	10.413	10.447	8.302
400x300	RD	10.502	8.461	8.729	8.753	9.141	9.182
	BT	29.388	26.341	23.800	23.977	23.201	21.702
	BT*	16.131	13.141	10.864	10.861	10.925	8.879
512x512	RD	13.107	9.888	10.551	10.618	11.429	11.448
	BT	30.750	25.743	24.844	25.571	25.923	24.097
	BT*	16.798	12.739	11.725	12.032	12.155	11.165
640x480	RD	13.727	10.489	11.130	11.754	12.167	13.067
	BT	31.366	26.594	25.789	26.369	26.434	26.684
	BT*	17.452	12.475	12.157	12.558	12.492	13.464
800x600	RD	16.352	12.718	13.342	14.297	14.932	17.574
	BT	33.455	28.161	28.403	28.474	28.084	30.672
	BT*	18.865	14.050	13.250	14.127	14.048	15.515
1024x768	RD	21.599	16.776	17.202	19.878	19.834	24.188
	BT	37.391	33.410	32.287	32.630	33.251	35.156
	BT*	20.948	17.274	15.355	16.601	16.754	19.257
1024x1024	RD	25.599	20.403	20.471	24.198	23.994	29.827
	BT	41.110	37.832	34.530	36.958	36.840	39.964
	BT*	22.298	19.011	17.005	18.685	18.913	22.308
1280x1024	RD	65.919	51.634	66.189	69.605	65.142	79.610
	BT	63.831	57.216	57.867	63.631	63.690	71.413
	BT*	38.004	30.302	31.801	36.393	36.929	44.354
1600x1200	RD	106.649	80.298	89.866	89.921	94.824	106.335
	BT	81.330	74.925	73.797	81.175	82.709	93.992
	BT*	48.546	42.065	42.194	48.319	49.734	60.422
2048x2048	RD	111.356	116.923	122.770	128.908	135.353	149.227
	BT	108.068	113.227	112.040	115.456	119.740	123.664
	BT*	69.570	71.657	73.807	76.021	78.301	83.070

Table 6.2: Performance results for summed-area table generation from a GeForce 6800GT with 256MB RAM installed on an AGP bus of an Intel Pentium 4 2.8GHz CPU with 1GB RAM.

8800GTX		Time (ms)					
image size		k=2	k=3	k=4	k=5	k=6	k=8
128x128	RD	2.846	2.732	2.814	2.669	2.647	2.348
	BT	6.927	6.393	6.349	6.493	6.561	6.398
	BT*	2.994	2.506	2.138	2.427	2.677	2.359
256x256	RD	1.705	1.687	1.682	1.531	1.527	1.303
	BT	7.412	6.915	6.937	6.706	6.330	6.113
	BT*	3.163	2.876	2.734	2.502	2.244	1.843
400x300	RD	2.870	3.067	2.610	2.663	2.445	1.414
	BT	10.139	9.432	9.476	8.996	8.778	8.223
	BT*	4.806	4.025	3.999	3.755	3.491	3.030
512x512	RD	3.299	2.930	2.971	3.480	3.510	4.012
	BT	9.595	9.356	9.196	8.675	7.889	8.673
	BT*	3.933	3.623	3.518	3.026	2.119	2.699
640x480	RD	3.464	3.126	3.071	3.270	3.666	4.232
	BT	7.759	5.886	5.812	5.333	6.223	6.182
	BT*	4.199	3.255	3.132	2.838	2.773	2.869
800x600	RD	5.326	6.015	6.076	6.251	6.371	6.897
	BT	5.964	4.818	4.712	4.769	4.780	8.891
	BT*	3.061	1.953	1.770	1.943	1.838	6.000
1024x768	RD	7.559	5.495	5.596	5.805	5.577	6.099
	BT	9.241	8.268	8.099	6.427	6.443	7.391
	BT*	5.642	4.559	4.252	2.914	2.924	3.602
1024x1024	RD	6.098	5.974	5.809	6.108	6.618	8.257
	BT	5.270	5.380	5.402	5.351	5.456	5.660
	BT*	2.920	3.040	2.968	2.999	3.194	3.366
1280x1024	RD	6.373	6.368	6.636	7.687	8.139	8.572
	BT	5.809	5.728	5.272	5.574	6.082	6.176
	BT*	3.626	3.626	3.123	3.393	3.970	4.026
1600x1200	RD	10.753	9.591	13.213	14.070	14.156	16.077
	BT	8.250	6.872	6.439	6.888	7.474	7.234
	BT*	5.034	4.402	5.446	5.229	5.865	6.112
2048x2048	RD	24.173	24.630	27.258	30.664	34.477	36.559
	BT	27.218	27.029	30.361	35.597	35.990	36.789
	BT*	7.290	7.040	11.710	16.241	16.788	16.981
4096x4096	RD	86.159	88.056	88.502	110.566	115.983	118.023
	BT	57.156	61.349	63.148	67.281	73.849	77.548
	BT*	26.549	28.549	30.460	36.264	43.693	45.887

Table 6.3: Performance results for summed-area table generation from a GeForce 8800GTX with 768MB RAM installed on an AGP bus of an AMD Athlon XP 64-bit 2.21GHz CPU with 2GB RAM.

6.3 Deriving Large Prefix Sums From SATs

It is known that a prefix sum can assist the generation of a summed-area table. Additionally, a partially generated summed-area table can be used to produce a prefix sum. Such procedure is interesting when there are constraints regarding the maximum size of the input array. Current GPU technology does not allow 1D texture with more than 8192 texels, but a 2D texture can have up to 8192×8192 texels.

Harris also suffered from such limitation on his CUDA implementation (HARRIS, 2007). The array size is limited due to the number of computation allowed in a single thread block. In his work, he used only binary trees ($k = 2$), leading to a limit of 1024 elements. If he has used a larger value for k, the number of additions would have grow as well, reducing even more the maximum size of the array. He surpassed this limitation by dividing the large array into smaller ones. A scan operation is performed on each of these smaller arrays, individually. The resulting sums (the last element of the scan) are placed in a new array and another scan operation is performed (a *superscan*). Each partial scan is then visited again, accumulating to its cells an appropriate element of the *superscan*. Note that Harris performs prescans and they must be converted into scans for the algorithm to work properly. Such conversions could be avoided using the algorithm proposed in the previous section.

The proposed method is similar to Harris' approach, but it has no constraints regarding the choices for k. The only one is related to the maximum texture size allowed by the hardware (as well as the maximum viewport size). First, the input array must be arranged in a matrix that can hold all of its elements (and respects any imposed dimension constraints). For a given input array A with size n, and a matrix M with h rows and w columns, a mapping of an index i on the array to its corresponding matrix row y and column x can be expressed as follows:

$$i = (y - 1) \cdot w + x \tag{6.1}$$

$$x = mod(i, w) + 1 \tag{6.2}$$

$$M(x, y) = \left\lceil \frac{i}{w} \right\rceil \tag{6.3}$$

where $mod(a, b)$ is the remainder of the division of a by b. With such parametrization, its corresponding matrix is defined as:

$$E(x) = \begin{cases} A(i) & \text{if } 1 \leq i \leq n \\ \omega & \text{if } (n+1) \leq i \leq (w \cdot h) \end{cases} \tag{6.4}$$

and automatically deals with padding, required in some arrangements, using the neutral element ω of the scan operation.

From such matrix, a scan operation is performed on each of its rows, generating a partial summed-area table. At this point, instead of repeating the scan process to all of its columns, only the *last column* is used. After this isolated scan operation (*superscan*), each cell of the partially generated SAT is accumulated with its corresponding predecessor row on the *superscan*. A complete execution of the algorithm is shown in Figure 6.5.

One can then use the same parametrization of Equations 6.1, 6.2 and 6.3 to perform operations on the array using its matrix representation without the need of fetching the entire matrix and building a new array. Table 6.4 compares the results of the technique described above with the ones obtained using Harris's method in CUDA. Speedup starts

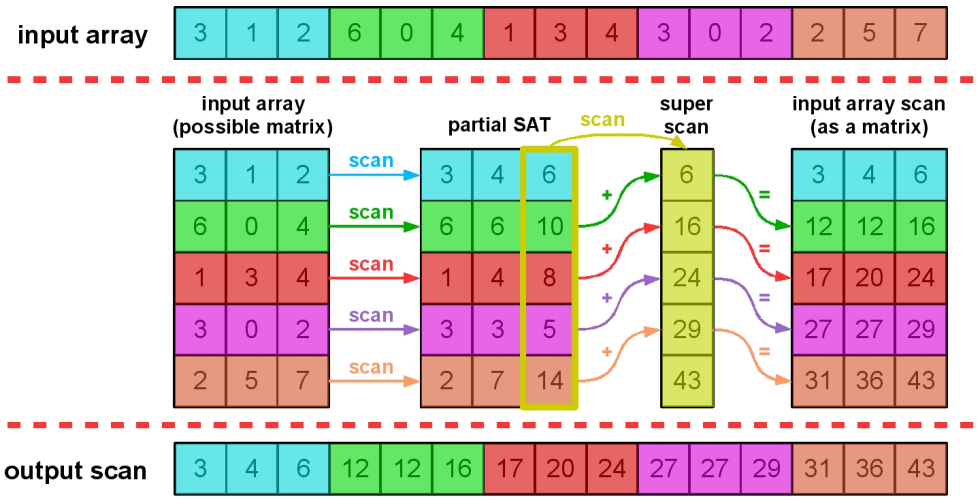

Figure 6.5: Generating a prefix-sum of a large arrays using summed-area tables. First, the array is decomposed in a matrix (table) and a scan operation is applied to all of its rows, resulting in a partial-summed area table. Next, only the last resulting column is submitted to another scan operation. The resulting column (superscan) is the accumulated with the entire partial SAT.

to appear only on the bigger arrays, but Harris's approach generates only a prescan (HARRIS, 2007). Times are given in milliseconds, corresponding to executions on a GeForce 8800GTX. For the proposed technique results, the optimal value for k from Table 6.3 was used.

	Time (ms)	
array size	HARRIS, 2007	Partial SAT
65536	0.137006	0.834288
262144	0.326900	1.056221
1048576	1.118091	1.359830
4194304	4.062923	3.810231
16777216	15.854781	13.738720

Table 6.4: Comparative results of Harris's method and the proposed approach to generate prefix-sum of large arrays. Performance was measured using a GeForce 8800GTX with 768MB RAM. Note that the performance of the proposed technique tends to loose to Harris's method for some array sizes. This is because of the context switching overhead between the auxiliary textures.

7 PHOTOGRAPHIC OPERATOR WITH SUMMED-AREA TABLES

This chapter presents a novel technique to approximate the Reinhard *et al.*'s photographic tone reproduction operator on GPU (REINHARD et al., 2002). The results are very similar to the ones obtained using a software-based implementation of the local operator but the presented technique is significantly faster than previous GPU implementations of the algorithm, making it suitable for real time applications. The proposed approach is based on the following key observations:

- Reinhard *et al.* used normalized differences of Gaussian-filtered luminance images to identify the largest approximately isoluminant region around each pixel (REINHARD et al., 2002). This is performed by checking whether these normalized differences are smaller than a specified threshold. Note that the actual (approximately) isoluminant region around a pixel can have arbitrary shapes, whereas the used Gaussian kernels are rotationally symmetric and do not perfectly fit these regions. Thus, the algorithm computes an approximate scale for each region. Similar results can be obtained using normalized differences of box-filtered luminance images;

- Since a convolution with a box filter is just an average over a rectangular region, it can be efficiently implemented using a summed-area table (CROW, 1984), for a fraction of the cost of a Gaussian convolution. Compared to the approach described by Goodnight *et al.* (GOODNIGHT et al., 2003), the proposed technique achieves an average speedup of about five to ten times, while producing very similar results. Compared with Krawczyk *et al.*'s approximation (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005), the speedup achieved with the proposed technique is about three times, lowering the memory requirements, and also significantly reducing the occurrence of halo artifacts that Krawczyk *et al.*'s technique may introduce.

7.1 Method Overview

The most computationally expensive operation of the local photographic operator is the evaluation of the Gaussian-filtered images, used in Blommaert and Martens' model for brightness perception (BLOMMAERT; MARTENS, 1990; REINHARD et al., 2002). To accelerate the convolutions, the proposed approximation replaces the Gaussian kernels from Equation 4.9 with box kernels, as below:

$$V(x, y, s) = L_r(x, y) \otimes BOX(x, y, s) \tag{7.1}$$

where $\otimes$ is the convolution operator and $BOX(x, y, s)$ is a box-kernel with scale s.

Such replacement is plausible for the operator since it does not display the filtered images; instead, it evaluates differences on the Gaussian hierarchy to find the largest region for which no sudden luminance variation occurs (the optimal measurement of the local contrast for a given pixel). Using summed-area tables, such regions can be filtered much faster than the original Gaussian-filtered ones.

A box-filter can effectively perform this task as well, but since it weights the contributions of all pixels in the neighborhood equally, the tone-mapped images produced might be more prone to halo artifacts than the ones produced with Gaussian-filters of the same scale. Fortunately, it is possible to minimize their occurrence by reducing the threshold used to estimate the local contrast. However, even with the same threshold of $\epsilon = 0.05$, results are similar to the ones produced with the original operator.

A summed-area table is then generated from the relative luminance image from Equation 4.6. Once it is ready, it can be used to filter rectangular regions on it, *centered* on arbitrary pixels. Moreover, the cost to evaluate a higher scale is the same as a smaller one, since filtering on a SAT has cost $O(1)$.

The resulting images of this approximation reduce the occurrence halos artifacts that Krawczyk *et al.*'s method tends to introduce (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005). Additionally, the proposed technique is approximately three times faster than Krawczyk *et al.*'s algorithm. When compared to Goodnight *et al.*'s results, the approximation is not restricted to a small number of adaptation zones to run at interactive rates, since the cost to evaluate the filtering is constant. In this case, the speedup is about ten times (GOODNIGHT et al., 2005). Quality comparison with these techniques is available in Section 7.4.1. Performance is discussed in Section 7.4.2. A complete description of the workflow of the proposed method is available in Section 7.4.

7.2 Box-filter: Mip-Map versus SAT

Together with summed-area tables, mip-mapping also provides a fast way to perform box-filtering. However, the latter does not center the filtering kernel at the individual pixels, leading to excessive blurring across high-contrast edges, preventing an accurate estimation of the local contrast, thus potentially introducing halos artifacts. Examples of box-filtering using SAT and Mip-Map are illustrated in Figure 7.1.

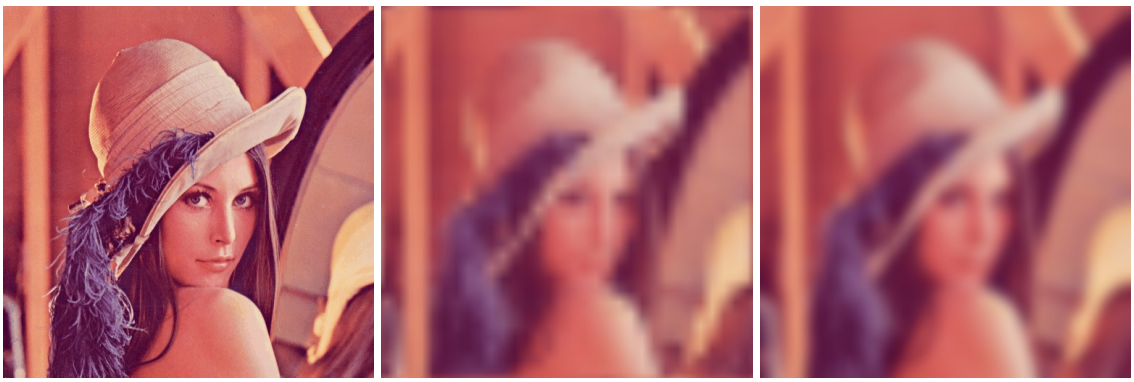

Figure 7.1: The original image (left) is submitted to a box-filter of radius 10 using mip-mapping (middle) and summed-area tables (right). Note that mip-mapping does not center the convolution in arbitrary pixels.

7.3 Precision Constraints

As mentioned in Section 5.3, summed-area tables may suffer from lack of precision, and when used with HDR content, this issue becomes more relevant. Due to the nature of HDR images, some pixels can have high intensity values, while others can have very low ones. When such pixels are close to each other, a summed-area table can suffer from several precision problems. In the proposed technique, a summed-area table is generated for the relative luminance image (Equation 4.6), which has lower values than the original luminance. However such compression might not mitigate the lack of precision. When this happens, loss of precision can not be simply ignored since it can potentially introduce artifacts as the ones shown in Figure 7.2.

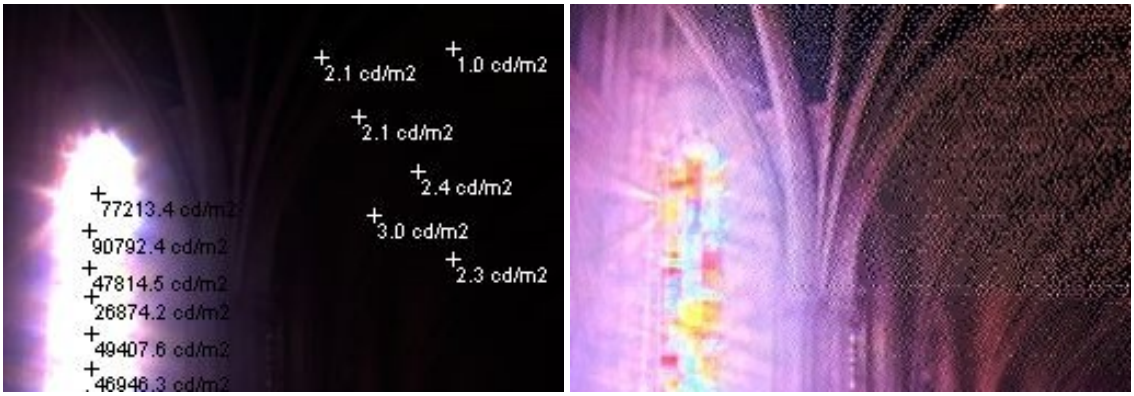

Figure 7.2: On the left, a zoomed version of the Nave image, with marks to show the luminances at the window and internal wall. On the right, the tone-mapped image using the SAT-based approximation without caring with precision problems.

To avoid such loss of precision, the solution proposed by Hensley *et al.* was employed (HENSLEY et al., 2005) and effectively removed the artifacts, as presented in Section 5.3. The process requires two additional stages before the SAT can be effectively generated. The first one is to produce the mip-map levels of the relative luminance texture in order to retrieve its average value. The second one consists on subtracting this average from all pixel values. The remaining of the method is kept unchanged, only being careful to decode the values when fetches on the summed-area table are performed. These two extra steps introduce small additional computational effort when compared to the time required to generate the SAT.

7.4 The Local Photographic Operator Using SAT

As mentioned in Section 4.6, the key value $\widetilde{L}$ of an input image I can be obtained using mip-mapping, followed by an exponentiation. For such, the corresponding log-luminance image, L_{log} of I, is obtained by evaluating the $log(L(x, y) + \delta)$ (Equation 4.5) for each of its pixels. An example of such log-luminance image is shown in Figure 7.3 (right).

From L_{log} (Figure 7.3, right), all of its mip-map levels are generated. The last mip-map level holds the averaged sum of Equation 4.5 (but still without the exponential function). The mip-map levels from the log-luminance image from Figure 7.3 (right) are illustrated in Figure 7.4.

To obtain the key value $\widetilde{L}$ of the image I, the exponential function is applied to the

Figure 7.3: The input HDR RGB image (left), followed by its corresponding luminance image (middle), accordingly to Equation 4.2, and the log-luminance image (right).

Figure 7.4: The mip-map levels of the log-luminance image L_{log}. The single texel of the last mip-map level is the averaged sum of Equation 4.5, but still without the exponential function.

single texel of the last mip-map level of L_{log}. The result of this operation is equivalent to the entire expression in Equation 4.5. With the key value, $\widetilde{L}$, the relative luminance image L_r can be generated, according to Equation 4.6. Figure 7.5 shows the resulting relative luminance image L_r of the input image I from Figure 7.3.

Figure 7.5: The relative luminance image, obtained using Equation 4.6.

As described in Section 7.1, from this point, a summed-area table can be generated from L_r. However, to avoid the precision problems discussed in Section 7.3, the relative luminance image L_r is submitted to a new mip-mapping process. The same texture memory from the previous mip-map of L_{log} can be used. The resulting texel of the last mip-map level will then store the average intensity $\overline{L_r}$ of the entire relative luminance

image L_r. Figure 7.6 illustrates the mip-map levels produced from the relative luminance image from Figure 7.5.

A new pass over the relative luminance image L_r is performed, subtracting from each of its pixels the average value $\overline{L_r}$, producing a new image L_{rsub}. Figure 7.7 illustrates the resulting pixel's subtraction from Figure 7.5 according to its average value (the last mip-map level from Figure 7.6).

Figure 7.6: The mip-map levels of the relative luminance image L_r. The single texel of the last mip-map level is the average $\overline{L_r}$ of the entire relative luminance image.

Figure 7.7: The relative luminance image L_r from Figure 7.5 subtracted from its average $\overline{L_r}$ (the single texel of the last mip-map level from Figure 7.6).

A summed-area table is then produced from the subtracted image L_{rsub} (Figure 7.7), mitigating possible precision constraints. Once the SAT is ready, the tone-mapping stage can finally start.

For each pixel, the box-filtered values for the required scales, $V(x, y, s)$, from Equation 7.1), are obtained by fetching their corresponding key cells (A, B, C and D) on the SAT, as defined in Section 5.2. It is important to remember to add $\overline{L_r}$ back after filtering, since the SAT was generated from L_{rsub}. Figure 7.8 shows the filtered relative luminance images $V(x, y, s)$ for each required scale.

The normalized differences are evaluated as originally defined in Equation 4.11 (Figure 7.9). When the largest scale s_{max} for each pixel is found (satisfying Equation 4.12), it is used to estimate the measurement of local contrast of that pixel ($V(x, y, s_{max})$). Figure 7.10 shows the largest scale s_{max} for each pixel (White means bigger neighborhood) and its corresponding estimation of local contrast, $V(x, y, s_{max})$.

The measurement of local contrast $V(x, y, s_{max})$ is used to perform the luminance compression, resulting in normalized displayable intensity L_d, according to Equation 4.8. The last step then scales the original HDR color components using the compressed luminance L_d, as described in Equation 4.3, with $L' = L_d$. The resulting image is then the tone-mapped version of the input image I. Figure 7.11 shows a comparison between the tone-mapped image using the original operator (left) and the approximation described in this section (right).

Figure 7.8: The relative luminance image filtered at different scales (adaptation zones), obtained using box-filtering (Equation 7.1). The kernel sizes are the same from Figure 7.8.

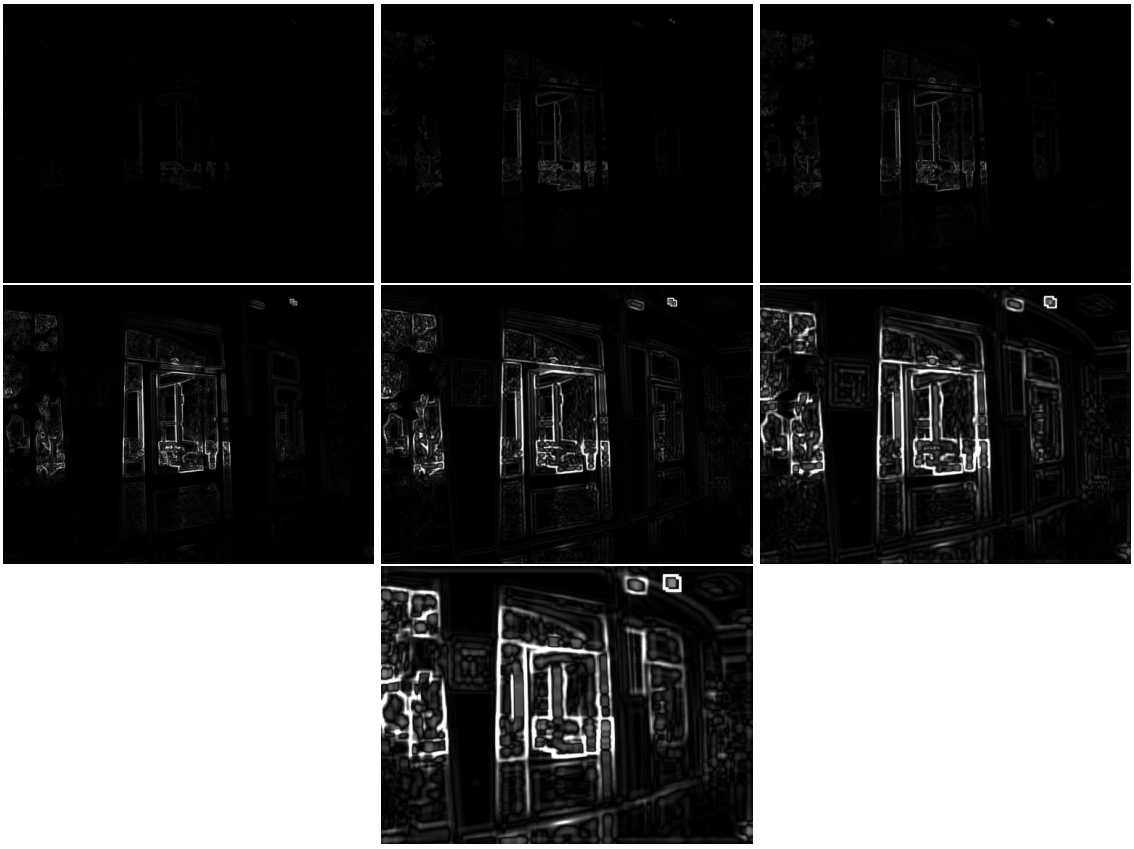

Figure 7.9: The seven normalized differences of the adaptation zones from Figure7.8, accordingly to Equation 4.11.

Figure 7.10: The representation of the size of the maximum approximately isoluminant neighborhood of each pixel (s_{max}). White means bigger neighborhood. The image on the right shows the estimated local contrast of each pixel ($V(x, y, smax)$). This values will be used to perform dynamic range compression (Equation 4.8). Results obtained using box-filtering, according to the images from Figure 7.8 and 7.9. Note the similarities with the ones using Gaussian filters from Figure 4.10.

Figure 7.11: A comparison between the result produced by the original operator (left), which uses Gaussian-filtering, and the proposed approximation which uses box-filtering (right). Note that the approximated result is very similar to the original one.

7.4.1 Image Quality Comparison

Although Goodnight et al's method can reproduce the same results of the Reinhard *et al.*'s technique (GOODNIGHT et al., 2003; REINHARD et al., 2002), it does not run at interactive rates when using more than two adaptation zones for current display resolutions. Krawczyk *et al.*'s method introduces noticeable halos due to the excessive blur on the relative luminance filtered images (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005). In the proposed technique, the occurrence of halos is minimized, since the filtering for each scale is centered at each pixel. Compared with existing GPU implementations, the proposed technique produces better results.

The use of the threshold $0.025 < \epsilon < 0.05$ can make some difference for certain images. Figures 7.12 and 7.13, compare the quality of the results produced by the proposed technique and by Krawcyzk *et al.*'s approach, using the S-CIELAB color difference metric described in (ZHANG et al., 1997). The S-CIELAB is a spatial extension to the CIE L*a*b* DeltaE color difference metric that incorporates the pattern-color sensitivity measurements by Poirson and Wandell (ZHANG et al., 1997). In these error images, white means bigger error.

Figures 7.14 to 7.22 provide illustrative comparisons between the original local photographic operator (left) (REINHARD et al., 2002), Krawczyk *et al.*'s approximation (middle) (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005), and the proposed SAT-based approximation presented in this thesis (right). For all these images, $\epsilon = 0.05$ was used. Krawczyk *et al.*'s method potentially introduces noticeable halos. The proposed approximation is less prone to such artifacts and the achieved image quality is very similar to the original local photographic operator. Note the occurrence of halos in Krawczyk *et al.*'s results. Figure 7.23 shows additional results of images processed with the proposed approximation of the photographic operator.

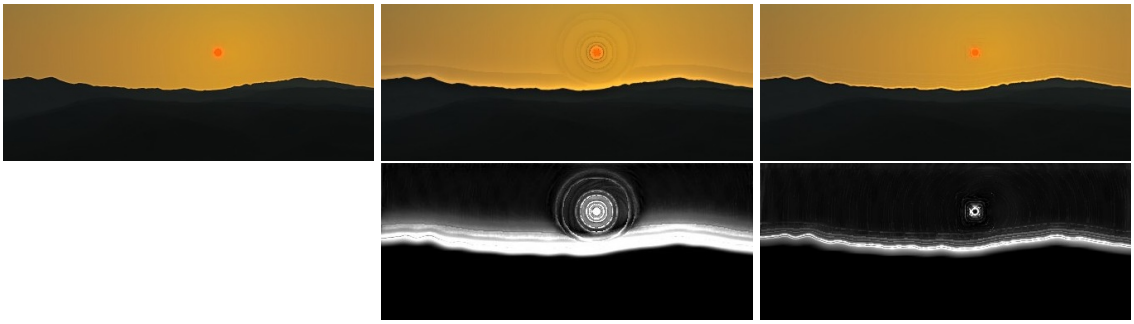

Figure 7.12: Comparison of the results produced by Krawczyk *et al.*'s and the proposed technique, using $\epsilon = 0.05$. Top row: tone-mapped image using Reinhard *et al.* original operator (left); Krawczyk *et al.*'s technique (middle), and the proposed method (right). Bottom row: the respective per-pixel perceptual error computed with the S-CIELAB color difference metric, using the result from the original operator as reference. White means bigger errors.

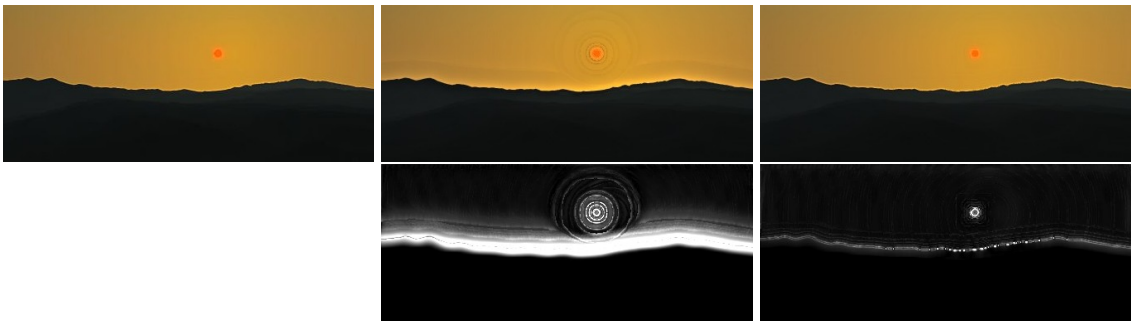

Figure 7.13: Comparison of the results produced by Krawczyk *et al.*'s and the proposed technique, using $\epsilon = 0.025$. Top row: tone-mapped image using Reinhard *et al.* original operator (left); Krawczyk *et al.*'s technique (middle), and the proposed method (right). Bottom row: the respective per-pixel perceptual error computed with the S-CIELAB color difference metric, using the result from the original operator as reference. White means bigger errors.

Figure 7.14: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^6 : 1$.

Figure 7.15: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^9 : 1$.

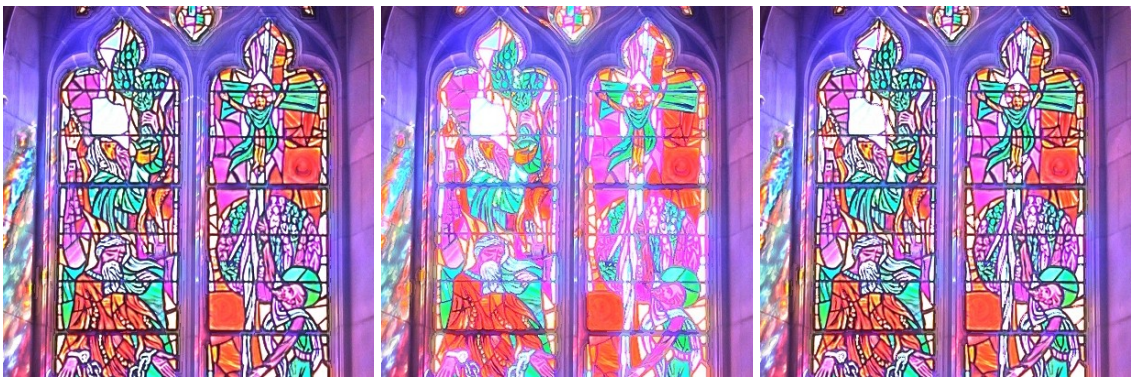

Figure 7.16: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^10 : 1$.

Figure 7.17: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^9 : 1$.

Figure 7.18: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^5 : 1$.

Figure 7.19: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^6 : 1$.

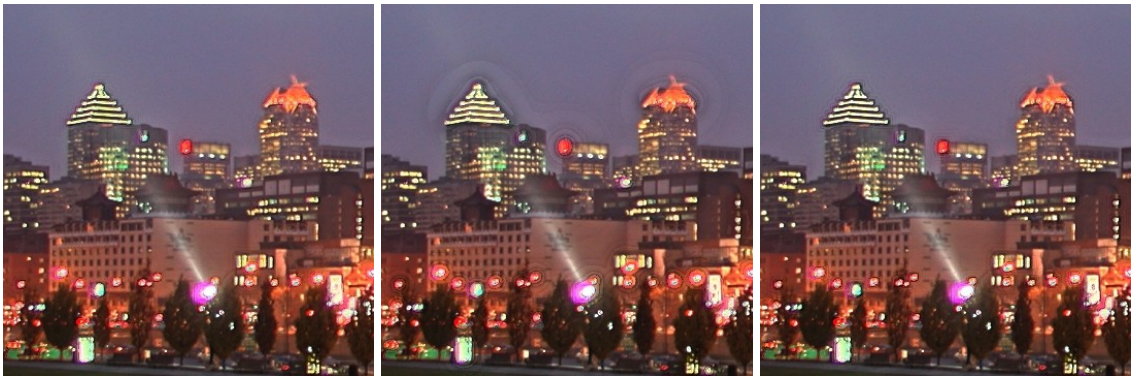

Figure 7.20: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^{1}0 : 1$.

Figure 7.21: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^{1}1 : 1$.

Figure 7.22: Image quality comparison between the original Reinhard *et al.*'s local operator (left), Krawczyk *et al.*'s approximation (middle), and the proposed approximation described in this thesis (right). For all images, $\epsilon = 0.05$. Dynamic range of the input image: $10^{1}2 : 1$.

Figure 7.23: Additional examples of tone-mapped images using the proposed approximation of the local photographic operator. Dynamic range of the input images are, $10^1 1 : 1$, $10^9 : 1$ and $10^7 : 1$, respectively.

7.4.2 Performance Comparison

The proposed approximation described in this Chapter is significantly faster than previous GPU implementations of the same operator. The performance results were measured using the improved balanced tree algorithm described in Section 6.1, with the best values for k, according to the Tables 6.1, 6.1 and 6.1. The computer configurations for each graphics card are the same from the ones of Section 6.1.

Krawczyk *et al.*'s approximation (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005) is about four times faster than Goodnight *et al.*'s approach (GOODNIGHT et al., 2003). For such reason, the performance measurements of Table 7.1 compare the proposed technique results only with Krawczyk *et al.*'s approach. The approximation described in this chapter achieves a speedup of about three times in a GeForce 8800GTX, when compared with Krawczyk *et al.*'s approximation (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005).

Krawczyk *et al.*'s method (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005) requires five textures of 1-channel each, with the same dimensions of the input image. The proposed technique described in this thesis requires only 2 and $1/3$ (2 full-sized textures for the balanced tree scan and $1/3$ to store the required mip-map levels). Also note that as k becomes higher, the texture size of one of the auxiliary textures lowers (Section 5.1.4).

		FPS		speedup
		SAT-based	KRAWCZYK, 2007	
128x128	FX5200	62	34	1.823529
	6800GT	41	32	1.281250
	8800GTX	186	114	1.631579
256x256	FX5200	17	12	1.416667
	6800GT	40	32	1.250000
	8800GTX	174	79	2.202532
400x300	FX5200	10	5	2.000000
	6800GT	37	29	1.275862
	8800GTX	172	67	2.567164
512x512	FX5200	5	2	2.500000
	6800GT	31	23	1.347826
	8800GTX	168	59	2.847458
640x480	FX5200	4	2	2.000000
	6800GT	28	22	1.272727
	8800GTX	145	56	2.589286
800x600	FX5200	3	1	3.000000
	6800GT	25	18	1.388889
	8800GTX	152	51	2.980392
1024x768	FX5200	2	1	2.000000
	6800GT	20	14	1.428571
	8800GTX	145	50	2.900000
1024x1024	FX5200	1	1	1.000000
	6800GT	17	14	1.214286
	8800GTX	163	45	3.622222
1280x1024	6800GT	12	6	2.000000
	8800GTX	149	41	3.634146
1600x1200	6800GT	10	3	3.333333
	8800GTX	94	35	2.685714
2048x2048	6800GT	4	3	1.333333
	8800GTX	77	17	4.529412
4196x4196	8800GTX	13	3	4.333333

Table 7.1: Performance results of the SAT-based approximation for the local photographic operator and Krawczyk *et al.* approximation (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005).

8 CONCLUSION

This thesis presented a new technique to accelerate the photographic local tone-mapping operator (REINHARD et al., 2002). The proposed solution does not exactly reproduce the original operator, but provides an approximation that leads to very similar results and whose performance is up to three to ten times faster than exiting GPU implementations (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005; GOODNIGHT et al., 2003).

The choice for the photographic operator comes from its simplicity, automaticity, robustness and perceptually-driven approach (REINHARD et al., 2002). It can effectively deal with a wide range of HDR content, preserving noticeable details by evaluating differences in a set of Gaussian-filtered images (adaptation zones), based on a model of brightness perception (BLOMMAERT; MARTENS, 1990). Existing GPU implementations limit the number of adaptation zones in order to run at interactive rates (GOODNIGHT et al., 2003), or introduces noticeable halos artifacts due to excessive blur along contrastive edges (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005).

The achieved speedup is obtained by replacing the Gaussian-filtered hierarchy of relative luminances images with a set of box-filtered ones. For such filtering, a summed-area table is employed since it can filter an arbitrary rectangular region of an image in constant time (CROW, 1984). As a result, only one summed-area table has to be generated from the first relative luminance image, from which all other filtered levels can be queried.

Such filtering replacement is possible because the operator only uses the Gaussian-filtered images to estimate the largest neighborhood for each pixel where no sudden luminance variation occurs. A box-filter can effectively performs this task, but since it weights the contributions of all pixels in the neighborhood equally, the tone-mapped images are more prone to halo artifacts than the ones produced with Gaussian-filters of the same scale. Fortunately, it is possible to minimize the occurrence of halos by reducing the threshold used to estimate the local contrast. Mip-mapping also provides a fast way to perform box-filtering, but it does not center the filtering on arbitrary pixels and might lead to excessive blurring across high contrast edges (CROW, 1984; HENSLEY et al., 2005).

Summed-area tables can be created using either recursive doubling (DUBOIS; RODRIGUE, 1977) or balanced trees (BLELLOCH, 1990). The former has cost $O(nlog(n))$ while the cost of the latter is $O(n)$, but it requires twice the number of passes used in recursive doubling. In the computer graphics literature, authors tend to rely on recursive doubling for unclear reasons (HENSLEY et al., 2005; LAURITZEN, 2007), and only recently some researches started to use the balanced tree approach (SENGUPTA; LEFOHN; OWENS, 2006; SENGUPTA et al., 2007; HARRIS, 2007).

This thesis presented an algorithmic improvement for the balanced tree approach. Although it does not reduce the asymptotic complexity of the algorithm, it avoids unnecessary computation. The improvement allows a balanced tree algorithm to perform scan

operations without the need of deriving them from prescan ones. The modification does not change the overall structure of the original algorithm. Additionally, an alternative way to perform the scan of larger arrays, similarly to the one presented by Harris (HARRIS, 2007), was described. Using a partially generated summed-area table, only its last column is submitted to another scan operation which is then combined with the partial results. Although relying on simple ideas, none of these improvements were found in the literature, and any application that uses prefix sums can benefit from them.

Due to the considerable speedup and the good approximation obtained, this work provides an attractive solution for HDR rendering applications that require high performance without compromising image quality.

9 FUTURE WORK

Not only HDR rendering and prefix sum applications can benefit from the contributions of this work. Since the proposed SAT-based approximation for the photographic operator implements an instance of Blommaert and Martens (BLOMMAERT; MARTENS, 1990) brightness perception model, any application that uses similar models can also benefit from the results of this thesis. When performance is desirable, one can replace the exact model with the approximation described in this thesis. Thus, there are at least three classes of applications that can immediately benefit from the proposed approximation: illustration generation, feature line detection, and image retargeting.

Figure 9.1: Example of automatic human facial illustration generation. The original image (left) processed using Gooch *et al.*'s technique (GOOCH; REINHARD; GOOCH, 2004) using Gaussian filters to evaluate Blommaert and Martens's model for brightness perception (BLOMMAERT; MARTENS, 1990) (middle), and using the box-filter-based approximation of the Blommaert and Martens's model presented in this thesis (right).

Gooch *et al.* introduced a technique for automatic generation of human facial illustration (GOOCH; REINHARD; GOOCH, 2004) that uses Blommaert and Martens's model (BLOMMAERT; MARTENS, 1990). Their work is not restricted to non-photorealistic rendering (NPR) or data transfer: using psychophysical studies, the authors concluded that illustrations can assist and accelerate recognition and learning tasks, since they reproduce only the most relevant features of a face. Although performance was not crucial in their work, one can envision artistic effects, such as rotoscopy, which basically consists on drawing lines over images. The process can then automatically suggests relevant lines

to guide an artist, be plugged in a video stream, or be used as a NPR post-effect in a game. For such classes of applications, performance is desirable, and they can benefit from the results of this thesis. However, since the content may change at each frame, some effort might be required to avoid flickering artifacts over time. Figure 9.1 shows some partial results in the context of human facial illustrations, obtained by replacing the Gaussian filters by the box-filter approximation. Figure 9.2 shows more general results, not restricted to faces.

Figure 9.2: Example of automatic illustrations generated using Gooch *et al.*'s technique (GOOCH; REINHARD; GOOCH, 2004). Left column: the original colored images. Middle column: the resulting illustration using Gaussian filters to evaluate Blommaert and Martens's model for brightness perception (BLOMMAERT; MARTENS, 1990). Right column: the resulting illustration using the box-filter-based approximation of the Blommaert and Martens's model presented in this thesis.

There are several applications that benefit from feature line detection. They are basically used for NPR rendering, for visualization and to assist a manual task in an interactive application, thus a fast way to detect these lines is desirable. Moreover, the determination of lines that are most noticeable can be a powerful tool. Legeai uses perceptual information to identify feature lines (LEGEAI, 2005). An interesting fact about his research is that he is also using Blommaert and Martens brightness perception model (BLOMMAERT; MARTENS, 1990) in order to extract potential feature lines in a perceptual way. Since the evaluation of this model is time demanding, it can be accelerated using the proposed approximation for the brightness perception model. Figure 9.3 shows some results produced by Legeai's technique.

The third class of potential applications for using the results of this thesis is image retargeting. Due to the diversity and popularity of digital display equipments that exist in many resolutions and sizes, from cell phones, portable video-games, palm computers and widescreen displays, to cite a few, it is desirable to display existing imaging content on them in a fashioned way. Real-time performance is also desirable when dealing with video streams on such devices. A simple resizing may not produce a pleasant result, since it should not rely only on geometric constraints, but must consider imaging content as well. The most remarkable work on imaging retargeting was recently introduced by Avidan and Shamir (AVIDAN; SHAMIR, 2007). They presented an operator called seam

carving, where a seam is basically defined as a connected path of pixels, optimality identified using an energy function. An image is then resized by removing (or inserting) seams. For the energy function, they found that a simple gradient magnitude analysis often produces good results, although they have also considered saliency and eye-tracking analysis. However, they did not used any brightness perception function, such as the Blommaert and Martens model. Such function seems to be a good choice to guide the process since it will potentially remove seams that are noticeably more irrelevant. See Figure 9.4 for an example of image retargeting.

From the investigation on human facial illustration and feature line detection, the approximation of Blommaert and Martens brightness perception model provided in this work also seems to be an attractive alternative for further research on interactive illustrations for visualization.

Finally, it would be desirable to obtain a formal proof to the correctness of the improved balanced tree algorithm for scan operations, ensuring that it produces correct results with any associative operator that has an inverse operator with the same neutral element.

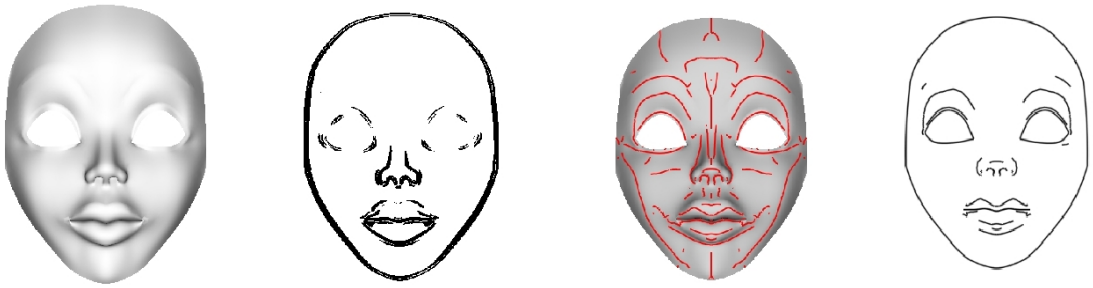

Figure 9.3: Legeai's results for feature line detection (LEGEAI, 2005). From left to right: diffuse model using Gouraud shading; brightness map of the Gouraud shaded image; crest line representation of the Gouraud shaded image; and a brightness map applied to the crest line image in order to identify the most relevant feature lines. Source: (LEGEAI, 2005).

Figure 9.4: An example of image retargeting using the seam carving technique from Avidan and Shamir (AVIDAN; SHAMIR, 2007). The original image (left) resized using a traditional approach (top-right) and using the seam carving method (bottom-right). The red paths of the input image are examples of seams. Source: (AVIDAN; SHAMIR, 2007).

APPENDIX A COMPLEXITY ANALYSIS ON PREFIX SUM ALGORITHMS

This appendix presents a detailed analysis of the complexity of the three algorithms described in this work to generate prefix sums: brute force, recursive doubling and balanced trees. The analysis comprises only the sequential execution, *i.e.*, using a single processor. First, each algorithm is analyzed for the case of arrays of size n, and then they are used to derive the complexity for the summed-area table computation. For simplicity, the ceiling function is omitted, for the cases when $log_k(n) \notin \mathbb{N}$.

For both recursive doubling and balanced tree algorithms, a number k must be chosen and it is assumed to be:

$$2 \leq k \ll n \tag{A.1}$$

The complexity analysis of converting a prescan into a scan and vice versa is not detailed. A shift operation in an array of size n takes $O(n)$ time and the update of a single element in the end of an array is $O(1)$; similarly, an element-wise operation of two arrays of the same size has cost $O(n)$.

A.1 Brute Force

The cost of a brute force prefix sum algorithm is $O(n^2)$.

The brute force algorithm is simple. For each element i of the output, i values are read from the input array and $i-1$ operations are performed. For convenience, since $i > i-1$, the operation chosen here to derive the complexity is the number of readings per element. This statement leads to the following expression:

$$\sum_{i=1}^{n} i \tag{A.2}$$

which is an instance of an arithmetic series such as:

$$\sum_{i=m}^{n} i = \frac{(n-m+1)(n+m)}{2} \tag{A.3}$$

and relying on such property, A.2 can be rewritten as:

$$\sum_{i=1}^{n} i = \frac{(n-1+1)(n+1)}{2} = \frac{n(n+1)}{2} \tag{A.4}$$

thus leading to the conclusion that

$$\sum_{i=1}^{n} i = \frac{n^2 + n}{2} \tag{A.5}$$

which clearly is $O(n^2)$.

A.2 Recursive Doubling

The cost of a recursive doubling prefix sum algorithm is $O(n\ log_k(n))$.

The recursive doubling approach requires $log_k(n)$ passes. At each pass $i \geq 1$, a total of $n - k^{i-1}$ elements are updated and, before switching between the arrays, $k^i - k^{i-1}$ elements are copied in order to ensure data stability. The resulting complexity will then be the sum of the updates and copies. Since the number of copies is smaller than the number of updates, the complexity analysis follows just considering the number of updates. At each step, a total of $n - k^{i-1}$ elements are updated. Each of them retrieves k elements from the previous pass and performs $k - 1$ operations. Since the number of operations remains constant for all the elements — and from the assumption on A.1 — it will be omitted and the analysis follows solely considering the number of updated elements. This statement leads to the following expression:

$$\sum_{i=1}^{log_k(n)} (n - k^{i-1}) \tag{A.6}$$

which can be expressed as:

$$\left[\sum_{i=1}^{log_k(n)} n\right] - \left[\sum_{i=1}^{log_k(n)} k^{i-1}\right] \tag{A.7}$$

and the first sum can be rewritten as:

$$\sum_{i=1}^{log_k(n)} n = n \left[\sum_{i=1}^{log_k(n)} 1\right] \tag{A.8}$$

in which the expression inside the brackets is an instance of the following summation property:

$$\sum_{i=m}^{n} 1 = n - m + 1 \tag{A.9}$$

and matching the resulting bracketed expression in A.8 with the property above gives

$$\sum_{i=1}^{log_k(n)} 1 = log_k(n) - 1 + 1 = log_k(n) \tag{A.10}$$

thus plugging it on A.8 reduces to:

$$\sum_{i=1}^{log_k(n)} n = n\ log_k(n) \tag{A.11}$$

This gives a hint about the complexity of the algorithm. The derivation could be stopped here since the parcels on the second sum of A.7 does not heavily rely on n (just on a much lower $log_k(n)$) and will not have strength enough to nullify any of the terms of A.11. However, for a more formal conclusion, the derivation will continue.

The second sum of A.7 can be rearranged as:

$$\sum_{i=1}^{log_k(n)} k^{i-1} = \sum_{i=1}^{log_k(n)} k^i k^{-1} = k^{-1} \left[\sum_{i=1}^{log_k(n)} k^i \right] = \frac{1}{k} \left[\sum_{i=1}^{log_k(n)} k^i \right] \tag{A.12}$$

where the expression inside the brackets is an instance of a geometric series, defined as:

$$\sum_{i=m}^{n} r^i = \frac{r^{n+1} - r^m}{r - 1} \tag{A.13}$$

where r is the ratio of the progression, and, in A.12, this ratio is k. Matching the geometric series definition above with A.12 gives:

$$\frac{1}{k} \cdot \frac{k^{log_k(n)+1} - k^1}{k-1} = \frac{1}{k} \cdot \frac{k^{log_k(n)} k^1 - k}{k-1} = \frac{1}{k} \cdot \frac{nk - k}{k-1} = \frac{1}{k} \cdot \frac{k(n-1)}{k-1} = \frac{n-1}{k-1} \tag{A.14}$$

and finally, by replacing A.11 and A.14 in A.7 gives:

$$\sum_{i=1}^{log_k(n)} (n - k^{i-1}) = n\ log_k(n) - \frac{n-1}{k-1} \tag{A.15}$$

which, given the assumption of A.1, leads to the conclusion that the updates are $O(n\ log_k(n))$.

Now, to finish the analysis, one must add the cost of the copies and updates, $O(n) + O(n\ log_k(n))$, which is $O(n\ log_k(n))$.

A.3 Balanced Tree

The cost of a balanced tree prefix sum algorithm is $O(n)$.

The balanced-tree approach requires the building of two trees: one for the reduction stage and another for the expansion one. For a given number of leaves n, the number of cells of a single tree is given by the summation expression below:

$$\sum_{i=0}^{log_k(n)} \frac{n}{k^i} \tag{A.16}$$

but recall that two of them must be generated, so:

$$2 \left[\sum_{i=0}^{log_k(n)} \frac{n}{k^i} \right] \tag{A.17}$$

and also recall that the leaves of the reduction tree do not need to be generated, since they comprise the input array of size n - and are already computed - they can be removed from the expression above as follows:

$$2\left[\sum_{i=0}^{log_k(n)} \frac{n}{k^i}\right] - n \tag{A.18}$$

The number of operations in the reduction and expansion stages are not the same: the reduction tree reads k elements and performs $k-1$ operations while in the expansion tree the number of readings depends on the index of the children nodes. From a given child index $1 \leq d \leq k$, a total of d fetches and $d-1$ operations are performed, thus leading to the worst case when $d = k$. Assuming the worst case always, the number of operations performed at each node in both trees will be $k-1$ and remains constant. Relying on A.1, the analysis then follows solely considering the total of elements produced.

Recall the expression inside the brackets in A.18. It can be rearranged as follows:

$$\sum_{i=0}^{log_k(n)} \frac{n}{k^i} = n\left[\sum_{i=0}^{log_k(n)} \frac{1}{k^i}\right] = n\left[\sum_{i=0}^{log_k(n)} (k^i)^{-1}\right] \tag{A.19}$$

and since $(k^i)^{-1} = (k^{-1})^i$, it can be conveniently expressed as:

$$\sum_{i=0}^{log_k(n)} (k^{-1})^i \tag{A.20}$$

which is an instance of a geometric series, defined in A.13, where r is the ratio of the progression, and, in A.20, this ratio is $k-1$. Matching the expression above with the geometric series definition produces:

$$\sum_{i=0}^{log_k(n)} (k^{-1})^i = \frac{(k^{-1})^{log_k(n)} - (k^{-1})^0}{k^{-1} - 1} = \frac{k^{-1}k^{-log_k(n)+1} - 1}{k^{-1} - 1} \tag{A.21}$$

and since $k^{-log_k(n)} = n^{-1}$, the previous statement simplifies to:

$$\frac{k^{-1}n^{-1} - 1}{k^{-1} - 1} \tag{A.22}$$

for which the occurrences of 1 in the expression above can be replaced by kk^{-1}, since:

$$kk^{-1} = \frac{k}{k} = 1 \tag{A.23}$$

leading to the following:

$$\frac{k^{-1}n^{-1} - kk^{-1}}{k^{-1} - kk^{-1}} = \frac{k^{-1}(n^{-1} - k)}{k^{-1}(1-k)} = \frac{n^{-1} - k}{1-k} \tag{A.24}$$

which is equivalent to

$$\left(\frac{1}{n} - k\right)\left(\frac{1}{1-k}\right) = \left(\frac{1-kn}{n}\right)\left(\frac{1}{1-k}\right) = \frac{1-kn}{n(1-k)} \tag{A.25}$$

and can be plugged back on A.19, simplifying it:

$$n\left[\sum_{i=0}^{log_k(n)} (k^i)^{-1}\right] = n\left(\frac{1-kn}{n(1-k)}\right) = \frac{1-kn}{1-k} = \frac{kn-1}{k-1} \tag{A.26}$$

thus finally being replaced in A.18:

$$2\left(\frac{kn-1}{k-1}\right) - n = \frac{2kn-2}{k-1} - n \tag{A.27}$$

and expressing the above statement in the same denominator gives:

$$\frac{2kn-2-n(k-1)}{k-1} = \frac{2kn-2-kn+n}{k-1} = \frac{kn+n-2}{k-1} = \frac{n(k+1)-2}{k-1} \tag{A.28}$$

which, from the assumption of A.1, is $O(n)$.

A.4 SAT Generation Algorithms

A summed-area table is obtained running multiple instances of prefix sums on each of its rows and then on each of its resulting columns. Given an prefix sum algorithm $A_{scan}(n)$ for arrays of size n, the complexity of an algorithm to generate a summed-area table with dimensions $w \times h$ can be expressed as:

$$h \cdot O(A_{scan}(w)) + w \cdot O(A_{scan}(h)) \tag{A.29}$$

and is also useful to keep in mind that:

$$N = w \cdot h \tag{A.30}$$

being N the number of cells of the SAT.

Here follows the complexity analysis for the summed-area table case using the algorithms presented in this appendix:

- *A SAT can be generated using a brute force prefix sum algorithm in O(N(w+h))*:

  plugging the brute force algorithm on A.29 gives:

  $$h \cdot w^2 + w \cdot h^2 = h \cdot w \cdot w + w \cdot h \cdot h = w \cdot h(w+h) \tag{A.31}$$

  thus, from A.30, the expression above can be written as:

  $$N(w+h) \tag{A.32}$$

  and cannot be reduced anymore, leading to $O(N(w+h))$.

- *A SAT can be generated using a recursive doubling prefix sum algorithm in* $O(N\ log_k(N))$:

  plugging the recursive doubling algorithm on A.29 gives:

  $$h \cdot (w\ log_k(w)) + w \cdot (h\ log_k(h)) = h \cdot w\ log_k(w) + w \cdot h\ log_k(h) \tag{A.33}$$

  thus, from A.30, the expression above can be written as:

  $$N\ log_k(w) + N\ log_k(h) = N\ (log_k(w) + log_k(h)) \tag{A.34}$$

  and from the following property of the logarithms:

  $$log_k(a) + log_k(b) = log_k(a \cdot b) \tag{A.35}$$

  the former expression can be conveniently expressed as:

  $$N\ log_k(w \cdot h) \tag{A.36}$$

  and again, from A.30, it reduces to:

  $$N\ log_k(N) \tag{A.37}$$

  which is $O(N\ log_k(N))$.

- *A SAT can be generated using a balanced tree prefix sum algorithm in* $O(N)$:

  plugging the balanced tree algorithm on A.29 gives:

  $$h \cdot w + w \cdot h \tag{A.38}$$

  thus, from A.30, the expression above can be written as:

  $$2N \tag{A.39}$$

  which is $O(N)$.

APPENDIX B HDR IMAGE FORMATS

Ward introduced the use of high dynamic range images in computer graphics and developed the first HDR image format in 1985, the RGBE8 format, together with his rendering application RADIANCE (WARD, 1994c,b). This allowed the use of more physically accurate measurements for radiance intensities and the resulting synthesized images are said to be scene-referred in contrast to traditional device-referred images.

The RGBE8 format offers a compact representation (32-bit per pixel) and a large range of intensities, covering about 76 orders of magnitude. However, the RGBE8 has a relative precision of 1% (the quantization step between two close intensities), which is the limit of the acceptable human perceptible level (REINHARD et al., 2006). Also, 76 orders of magnitude is exaggerated when compared to real world scenes, which typically covers 14 orders of magnitude ($10^8 cd/m^2$ being sunlight and $10^{-6} cd/m^2$ being starlight).

The SGI/TIFF 32-bit LogLuv format (WARD, 1998) covers 38 orders of magnitude and has a relative precision of 0.3%. However, the format deviates from the traditional RGB color space and this seems to have prevented its widespread use. At a cost of extra memory, the ILM OpenEXR 48-bit format holds about 11 orders of magnitude with a relative precision of 0.1%, but this may not suffice to store all the intensity range of a real world scene (Industrial Light and Magic, 2002).

The Portable Floatmap format can deal with 79 orders of magnitude and has a negligible relative precision of only 0.000003%, but requires 96-bit per pixel and any noise present in the last half of the mantissa can compromise further compression (BOURKE, 2003).

There are more HDR image formats, such as Microsoft/HP scRGB, Kodak Cineon/DPX, Pixar/TIFF Log Encoding and JPEG 2000, to cite a few. As any other format, they deal with limitations on storage, magnitude range and relative precision. A detailed description about these different formats can be found in (WARD, 2005; HOLZER, 2006).

For the reasons above, the RGBE8 format (.hdr file) remains as the most widely used, and the vast majority of HDR content available today uses this format. The nature of the input HDR image format does not affects the evaluation process of tone-mapping operators. The content is decoded to floating-point values and the operator deals only with this representation, without the need of knowing the image's format encoding details.

APPENDIX C HDR ACQUISITION AND RENDERING

In a high dynamic range image, each pixel stores proportional radiance measurements from its corresponding scene. In low dynamic range images, these values rarely matches the true radiance of the original scene, due to the nonlinear nature of the quantization processes performed by the acquisition device or imaging encoding (DEBEVEC; MALIK, 1997). Such quantization can be seen as a simple instance of dynamic range compression (tone-mapping) for the particular target device capabilities.

Debevec *et al.* developed an approach to recover HDR images from low dynamic range ones (DEBEVEC; MALIK, 1997). They noticed that a set of photographs of a scene, each taken at a different exposure time, captures almost all the intensities of the scene. These LDR pictures can then be assembled into a single high dynamic range image, which they called a radiance map (see Figure C.1).

Later, Debevec showed how to capture natural lights to use them for image-based lighting (DEBEVEC, 1998). Relying on the knowledge from his previous work, a series of photographs of a mirrored ball inserted in a scene are taken and assembled into a single HDR image, as shown in Figure C.2 (DEBEVEC; MALIK, 1997). The resulting image is an angular map, that he called a light probe, and can be used in reflection-mapping as well as an omni-directional light source in a global illumination technique. To avoid the presence of the camera (and photographer) in the angular map, the same acquisition process is repeated at a different angle — 90 degrees — of the mirrored ball. An image registration process is then applied to both sets of images, removing any unwanted object. This process also leads to a better sampling and representation of incoming radiance arriving backwards the sphere of the first set of pictures. A light probe can be converted to others omni-directional map formats, such as cube-maps and latitude-longitude maps (see Figure C.2).

These works comprise a mark in HDR and realistic rendering (DEBEVEC; MALIK, 1997; DEBEVEC, 1998). Following their ideas, several researchers developed sophisticated rendering algorithms, enhancing the overall image quality relying on captured HDR natural lights (RAMAMOORTHI; HANRAHAN, 2001; SLOAN; KAUTZ; SNYDER, 2002; NG; RAMAMOORTHI; HANRAHAN, 2004; SLOMP; OLIVEIRA, 2005; PEERS et al., 2007). Figure C.3 provides examples of some of these works.

High dynamic range rendering is becoming an increasingly popular technique in computer graphics (Valve Software, 2005; Unigine Corp, 2005; BLYTHE, 2006; Crytek, 2007; Epic Games, 2007; Irrlicht Team, 2007) and such popularity is closely related to a set of features found in the current programmable graphics hardware: high performance, low cost and floating point texture support (CEBENOYAN, 2005; BLYTHE, 2006). HDR imaging tends to become even more popular with the widespread use of high-definition television and video, medias with increasingly storage capacities, next generation of video

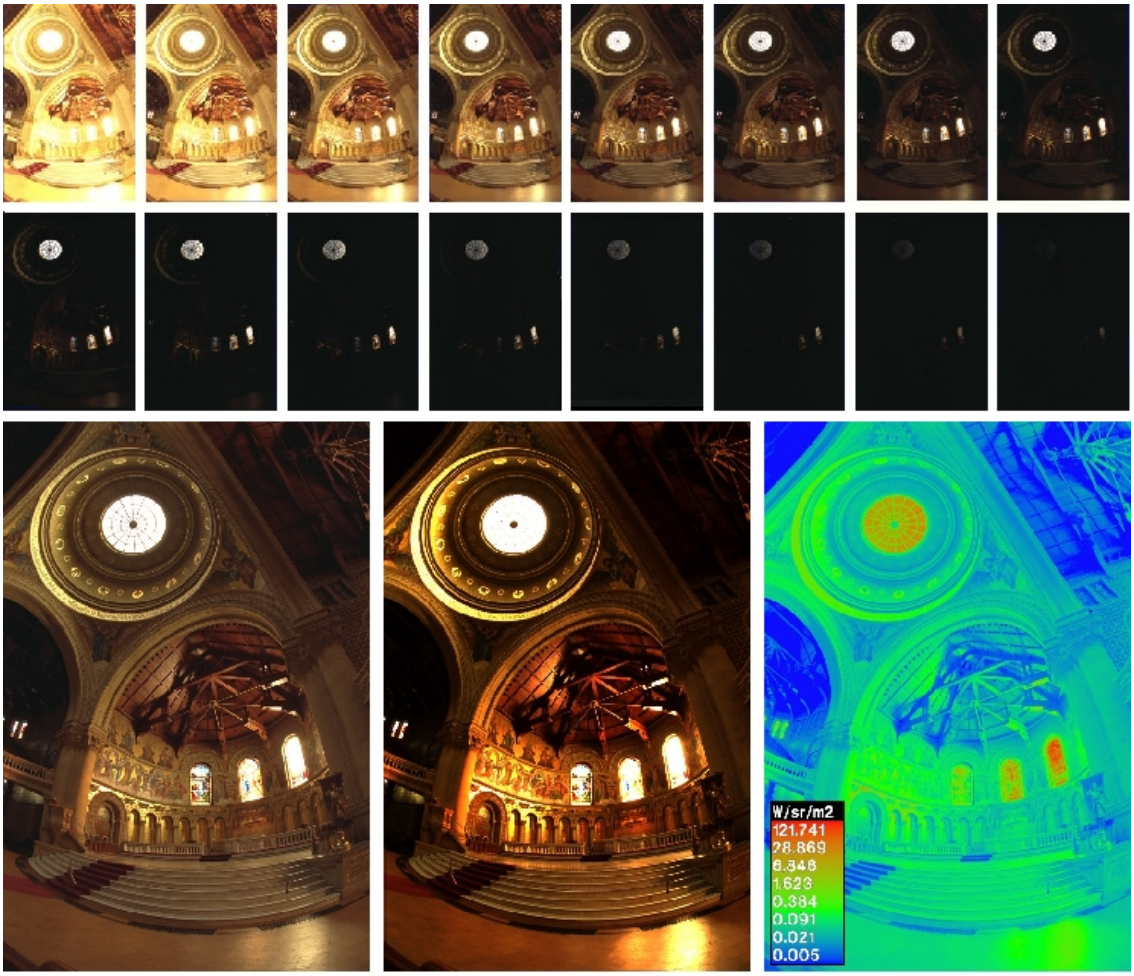

Figure C.1: Radiance map generated from photographs. First, a set of photographs are taken, each at an increasingly exposure time (all small images on top) and then are assembled into a single HDR image (bottom, middle). At bottom-left, a photograph taken with a common camera, subjected to the later encoding. At bottom-right, a false-color representation and the corresponding radiance scales from the radiance map (bottom, middle). Source: (DEBEVEC; MALIK, 1997).

games and recent developments on display devices technologies (Toshiba Corporation; Canon Inc, 2008; Dolby Laboratories, 2008). Figure C.4 compares the image quality between low and high dynamic range rendering in a gaming scenario.

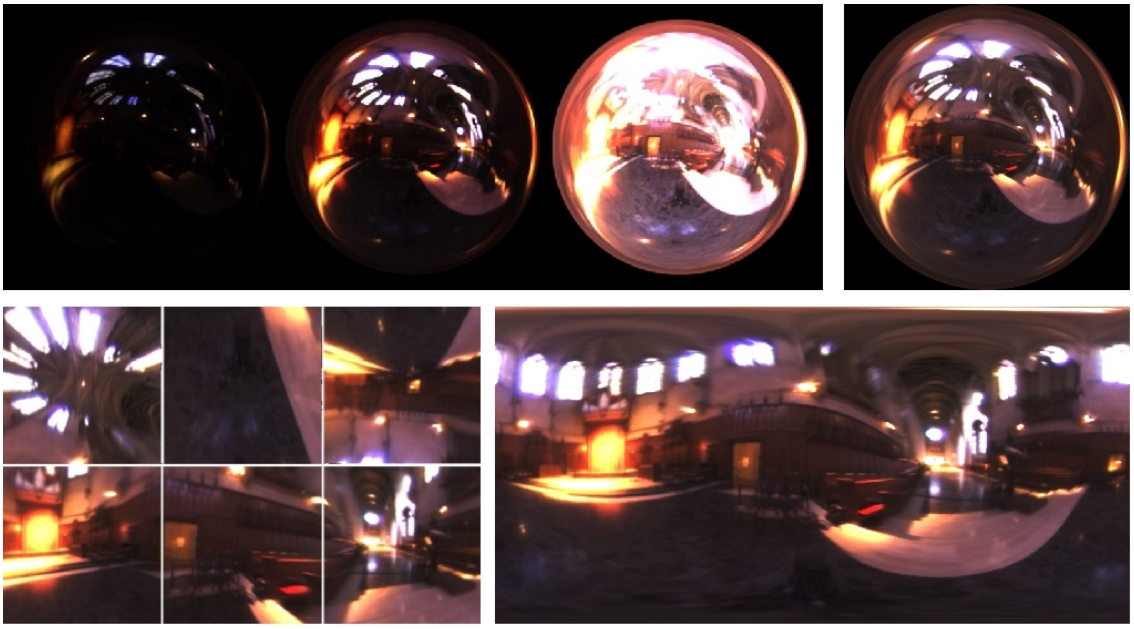

Figure C.2: Natural light can be captured and stored in a light probe (top-right), generated from a set of photographs of a mirrored ball (top-left). A light probe can be converted to an alternate format, such as cube-map faces (bottom-left) or a latitude-longitude map (bottom-right). Source: (DEBEVEC, 1998).

Figure C.3: Rendering with natural lights: Devebec used natural light to render solely synthetic objects (upper-left) and to insert virtual objects into real scenes (upper-right; the spheres in the middle of the table are virtual, while the remaining scene is real) (DEBEVEC, 1998). On the bottom row, some works that benefits from image-based lighting. From left to right: irradiance maps for fast diffuse lighting (RAMAMOORTHI; HANRAHAN, 2001); precomputed radiance transfer with translucency effects (SLOAN; KAUTZ; SNYDER, 2002); high frequency triple-product rendering (NG; RAMAMOORTHI; HANRAHAN, 2004); precomputed radiance transfer with thickened specular effects (SLOMP; OLIVEIRA, 2005).

Figure C.4: Two screenshots from the game FarCry: the image on the left does not use HDR rendering, while the one on the right performs, using a tone-mapping operator to compress the dynamic range of the image. Source: Wikipedia.

APPENDIX D AN OPTIMAL PREFIX-SUMS ALGORITHM

2

Basic Techniques

The task of designing parallel algorithms presents challenges that are considerably more difficult than those encountered in the sequential domain. The lack of a well-defined methodology is compensated by a collection of techniques and paradigms that have been found effective in handling a wide range of problems. This chapter introduces these techniques as they apply to a selected set of combinatorial problems, which are interesting on their own and often appear as subproblems in numerous computations.

It is important to view the introduced techniques as general guidelines for designing parallel algorithms, rather than as a manual of directly applicable methods. The combinatorial problems discussed include the prefix sums (Section 2.1), parallel prefix (Section 2.2), the convex hull (Section 2.3), merging (Section 2.4), insertions into 2-3 trees (Section 2.5), computing the maximum (Section 2.6), and coloring the vertices of a directed cycle (Section 2.7). Most of these parallel algorithms are used frequently in the remainder of this book.

2.1 Balanced Trees

The PRAM algorithm to compute the sum of n elements presented in Section 1.3.2 is based on a balanced binary tree whose leaves are the given n elements

and whose internal nodes represent additions. This algorithm is an example of the general strategy of *building a balanced binary tree on the input elements and traversing the tree forward and backward to and from the root*. An internal node u usually holds information concerning the data stored at the leaves of the subtree rooted at u. The success of such a strategy depends partly on the existence of a fast method to determine the information stored in an internal node from the information stored in its children. We use the computation of the prefix sums of n elements to provide another illustration of this strategy.

2.1.1 AN OPTIMAL PREFIX-SUMS ALGORITHM

Consider a sequence of n elements $\{x_1, x_2, \ldots, x_n\}$ drawn from a set S with a binary associative operation, denoted by $*$. The prefix sums of this sequence are the n partial sums (or products) defined by

$s_i = x_1 * x_2 * \ldots * x_i$, $1 \leq i \leq n$.

A trivial sequential algorithm computes s_i from s_{i-1} with a single operation by using the identity $s_i = s_{i-1} * x_i$, for $2 \leq i \leq n$, and hence takes $O(n)$ time. Clearly, this algorithm is inherently sequential.

We can use a balanced binary tree to derive a fast parallel algorithm to compute the prefix sums. Each internal node represents the application of the operation $*$ to its children during a forward traversal of the tree. Hence, each node v holds the sum of the elements stored in the leaves of the subtree rooted at v. During a backward traversal of the tree, the prefix sums of the data stored in the nodes at a given height are computed.

We start by giving a recursive version described by the steps needed to obtain the data stored in the nodes at height 1 and the steps needed to obtain the prefix sums after the recursive call on the nodes at height 1 terminates.

ALGORITHM 2.1
(Prefix Sums)
Input: *An array of $n = 2^k$ elements $(x_1, x_2, \ldots, x_n)$, where k is a nonnegative integer.*
Output: *The prefix sums s_i, for $1 \leq i \leq n$.*

begin
1. **if** $n = 1$ **then** {*set* $s_1 := x_1$; **exit**}
2. **for** $1 \leq i \leq n/2$ **pardo**
   Set $y_i := x_{2i-1} * x_{2i}$
3. *Recursively, compute the prefix sums of* $\{y_1, y_2, \ldots, y_{n/2}\}$, *and store them in* $z_1, z_2, \ldots, z_{n/2}$.

4. **for** $1 \leq i \leq n$ **pardo**
   {i *even* : *set* $s_i := z_{i/2}$
   $i = 1$: *set* $s_1 := x_1$
   i *odd* > 1 : *set* $s_i := z_{(i-1)/2} * x_i$}

end

EXAMPLE 2.1:

The prefix-sums algorithm on eight elements is illustrated in Fig. 2.1. The time units referred to in what follows are those indicated in the figure. During the first time unit, the four elements $y_1 = x_1 * x_2$, $y_2 = x_3 * x_4$, $y_3 = x_5 * x_6$, and $y_4 = x_7 * x_8$ are computed. The second time unit corresponds to computing $y'_1 = y_1 * y_2$ and $y'_2 = y_3 * y_4$ with a recursive call to handle these two inputs. The element $y''_1 = y'_1 * y'_2$ is computed during time unit 3. Hence, at the fourth time unit, the prefix sum of this single input is generated. The reverse process begins by generating, in time unit 5, the prefix sums z'_1 and z'_2 of the two inputs y'_1 and y'_2 generated during the second time unit. Similarly, during time unit 6, the prefix sums z_1, z_2, z_3, and z_4 of the four elements y_1, y_2, y_3, and y_4 are generated. Finally, the prefix sums $\{s_i\}$ of the x_i's are generated in time unit 7. □

On inputs of size $n = 2^k$, the prefix-sums algorithm requires $2k + 1$ time units such that, during the first k time units, the computation advances from the leaves of a complete binary tree to the root, where the leaves hold $x_1, x_2, \ldots, x_n$. During the last k time units, we transverse the tree in reverse order and compute prefix sums by using the data generated during the first k time units.

We note that the whole algorithm can be executed in place, and that we introduced additional variables for clarity.

We are now ready to examine the following theorem, which is stated within the WT presentation level.

Theorem 2.1: *The Prefix-Sums Algorithm (Algorithm 2.1) computes the prefix sums of n elements in time $T(n) = O(\log n)$, using a total of $W(n) = O(n)$ operations.*

Proof: The correctness proof will be by induction on k, where the size of the input is $n = 2^k$.

The base case $k = 0$ is handled correctly by step 1 of the algorithm.

Assume that the algorithm works correctly for all sequences of length $n = 2^k$, where $k > 0$. We will prove that the algorithm computes the prefix sums of any sequence of length $n = 2^{k+1}$.

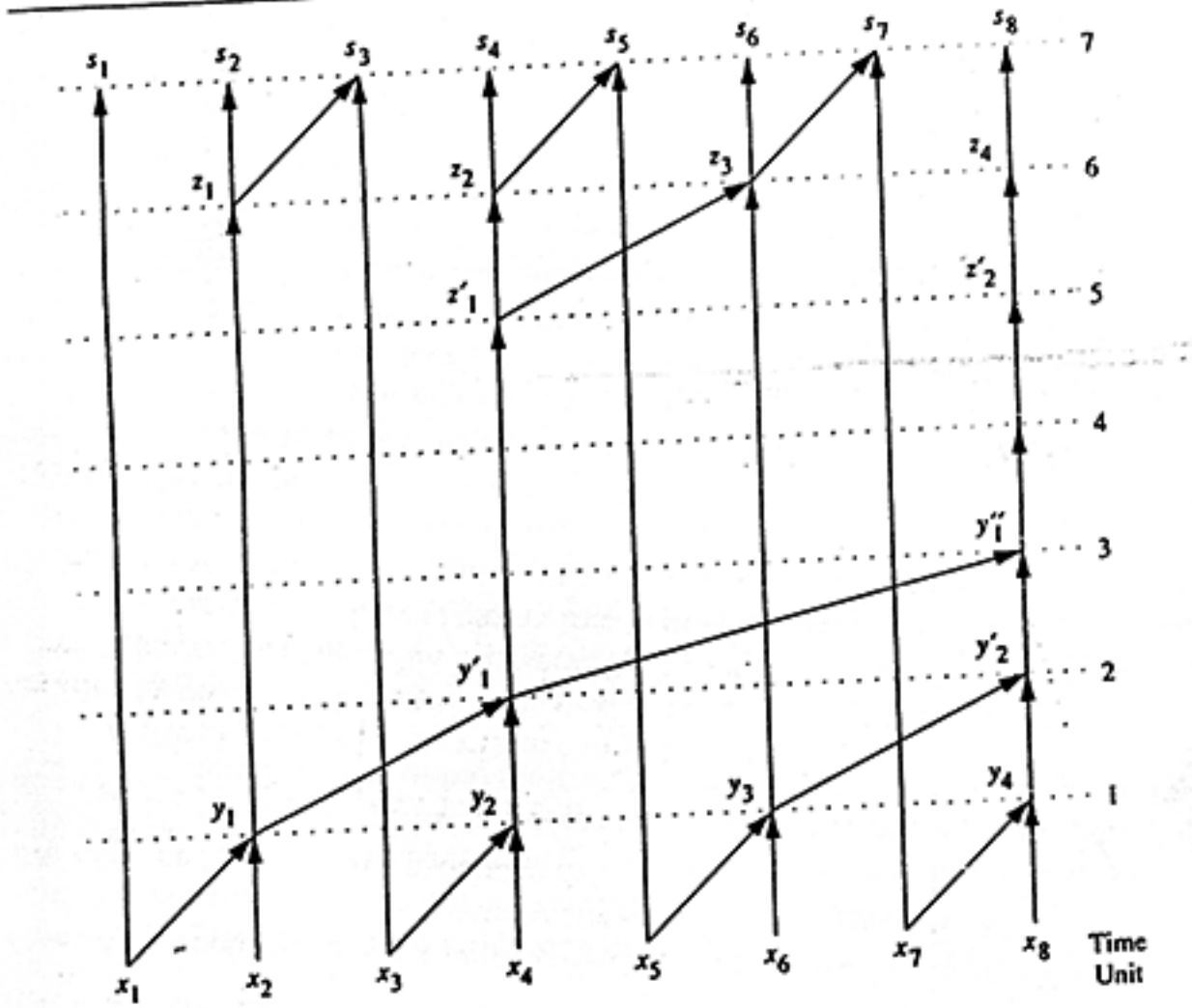

FIGURE 2.1
Prefix sums of eight elements. Each node represents a $*$ operation on the data stored in the tails of the incident arcs. During each time unit, the operations that take place are indicated by the presence of nodes with incoming arrows.

By the induction hypothesis, the variables $z_1, z_2, \ldots, z_{n/2}$, computed at step 3, hold the prefix sums of the sequence $\{y_1, y_2, \ldots, y_{n/2}\}$, where $y_i = x_{2i-1} * x_{2i}$, for $1 \le i \le n/2$. In particular, $z_j = y_1 * y_2 * \cdots * y_j$ and hence $z_j = x_1 * x_2 * \cdots * x_{2j-1} * x_{2j}$. That is, each z_j is nothing but the prefix sum s_{2j}, for $1 \le j \le n/2$. Thus, if i is even—say, $i = 2j$—then we have that $s_i = z_{i/2}$; otherwise, either $i = 1$ or $i = 2j + 1$, for some $1 \le j \le (n/2) - 1$. The case when $i = 1$ is trivial. For the remaining case of $i = 2j + 1$, we have $s_i = s_{2j+1} = s_{2j} * x_{2j+1} = z_{(i-1)/2} * x_i$. Therefore, all these cases are handled appropriately in step 4 of the algorithm. It follows that the algorithm works correctly on all inputs.

As to the resources required, they can be estimated as follows. Step 1

steps using $O(n)$ operations. Therefore, the running time $T(n)$ and the work $W(n)$ required by the algorithm satisfy the following recurrences:

$$T(n) = T\left(\frac{n}{2}\right) + a$$

$$W(n) = W\left(\frac{n}{2}\right) + bn,$$

where a and b are constants. The solutions of these recurrences are $T(n) = O(\log n)$ and $W(n) = O(n)$. □

PRAM Model: Since steps 1, 2 and 4 of Algorithm 2.1 do not require concurrent read or concurrent write capability, this algorithm runs on the EREW PRAM model. A lower bound of $\Omega(\log n)$ on the time it takes a CREW PRAM to compute the Boolean OR of n variables (regardless of the number of operations used) implies that the previous algorithm is *WT optimal*, or *optimal in the strong sense, on the EREW and CREW PRAM*. This lower-bound proof is presented in Chapter 10. □

2.1.2 A NONRECURSIVE PREFIX-SUMS ALGORITHM

We present a nonrecursive version of the prefix algorithm, which will be used in Section 2.1.3 to illustrate the details involved in adapting the WT scheduling principle.

Let $A(i) = x_i$, where $1 \le i \le n$. Let $B(h,j)$ and $C(h,j)$ be sets of auxiliary variables, where $0 \le h \le \log n$ and $1 \le j \le n/2^h$. The array B will be used to record the information in the binary tree nodes during a forward traversal, whereas the array C will be used during the backward traversal of the tree. Figure 2.2 illustrates the forward traversal, and Fig. 2.3 illustrates the backward traversal, for $n = 8$.

ALGORITHM 2.2
(Nonrecursive Prefix Sums)
Input: *An array A of size $n = 2^k$, where k is a nonnegative integer.*
Output: *An array C such that $C(0, j)$ is the jth prefix sum, for $1 \le j \le n$.*
begin
1. for $1 \le j \le n$ pardo
   Set $B(0, j) := A(j)$
2. for $h = 1$ to $\log n$ do
   for $1 \le j \le n/2^h$ pardo
   Set $B(h, j) := B(h - 1, 2j - 1) * B(h -$

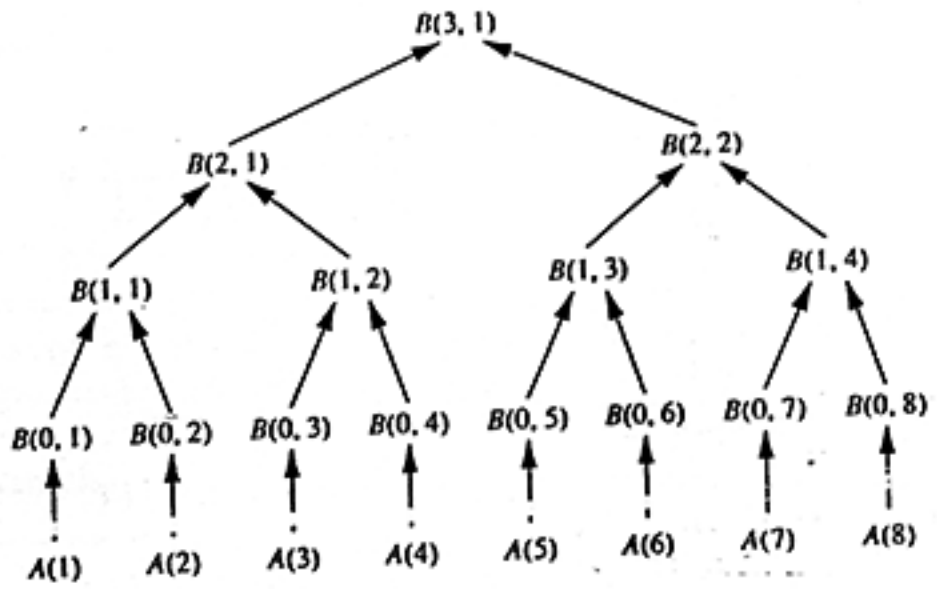

FIGURE 2.2
The bottom-up (forward) traversal of the binary tree used in the nonrecursive prefix sums algorithm. $B(0, j)$ is initially set to $A(j)$, and each internal node represents the operation $*$.

3. **for** $h = \log n$ **to** 0 **do**
   for $1 \le j \le n/2^h$ **pardo**
   $\begin{cases} j \text{ even} & : \text{Set } C(h,j) := C\left(h+1, \frac{j}{2}\right) \\ j = 1 & : \text{Set } C(h,1) := B(h,1) \\ j \text{ odd} > 1 & : \text{Set } C(h,j) := C\left(h+1, \frac{j-1}{2}\right) * B(h,j) \end{cases}$

end

EXAMPLE 2.2:

Let $n = 8$ (see Figs. 2.2 and 2.3). We initially set $B(0,j) = A(j)$, for all $1 \le j \le 8$. The $B(0,j)$s correspond to the leaves of the binary tree. The variables $B(1,j)$, where $1 \le j \le 4$, correspond to the internal nodes at height 1; the variables $B(2,j)$, where $1 \le j \le 2$, correspond to the internal vertices at height 2. The root of the binary tree is stored in $B(3, 1)$. We then traverse this binary tree backward. We start by setting $C(3, 1) = B(3, 1)$. At the next step, we generate $C(2, 1) = B(2, 1)$ and $C(2, 2) = B(3, 1)$. Hence, $C(2, 1)$ and $C(2, 2)$ hold the prefix sums corresponding to the inputs $B(2, 1)$ and $B(2, 2)$. Similarly, $C(1, 1)$, $C(1,2)$, $C(1,3)$, and $C(1,4)$ are the prefix sums of $B(1, 1)$, $B(1, 2)$, $B(1, 3)$, and $B(1, 4)$. Therefore the $C(0, j)$s hold the prefix sums of the original inputs. □

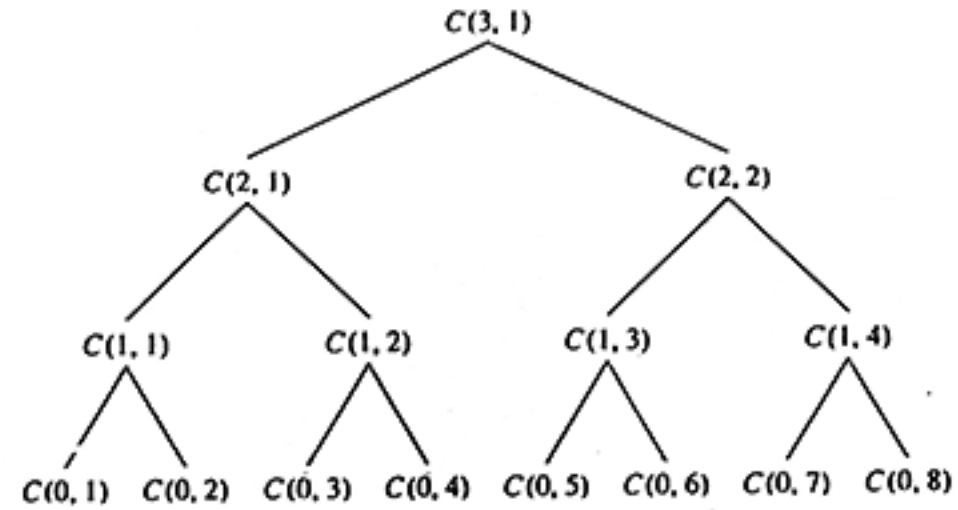

FIGURE 2.3
The elements of the array C as generated by the top-down (backward) traversal of the binary tree corresponding to the nonrecursive prefix-sums algorithm.

2.1.3 *ADAPTATION OF THE WT SCHEDULING PRINCIPLE

As indicated in Section 1.5, the adaptation of the WT scheduling principle requires the determination of the number of operations at each parallel step and a solution to the corresponding processor allocation problem.

Let $n = 2^k$. There are $2 \log n + 2 = 2k + 2$ parallel steps in Algorithm 2.2. Let $W_{1,1}$ be the number of operations performed at step 1, and let $W_{i,m}$ be the number of operations performed at step i in Algorithm 2.2 during the mth iteration, $i = 2, 3$. Then, $W_{1,1} = n$, $W_{2,m} = n/2^m = 2^{k-m}$ for $1 \le m \le k$, and $W_{3,m} = 2^m$ for $0 \le m \le k$. The total number of operations is given by $W(n) = W_{1,1} + \sum_{m=1}^{k} W_{2,m} + \sum_{m=0}^{k} W_{3,m} = n + 2^k \sum_{m=1}^{k} 2^{-m} + \sum_{m=0}^{k} 2^m = n + n(1 - (1/n)) + 2n - 1 = O(n)$. The processor allocation problem is addressed next.

Let our PRAM have $p = 2^q \le n$ processors $P_1, P_2, \ldots, P_p$, and let $l = n/p = 2^{k-q}$. The input array is divided into p subarrays such that processor P_s is responsible for processing the sth subarray $A(l(s-1)+1), A(l(s-1)+2), \cdots, A(ls)$. At each height h of the binary tree, during either a forward or a backward traversal, the generation of the $B(h,\cdot)$ and $C(h,\cdot)$ values is divided in a similar way among the p processors. This division is performed in a way similar to the details we worked out for the parallel algorithm to compute the sum of n numbers in Example 1.13.

In Algorithm 2.3, which follows, conditions 2.1 and 3.1 hold whenever the number of operations at that particular iteration is greater than or equal

to p. These operations are then distributed equally among the p processors. In the case where the number of operations is less than p (steps 2.2 and 3.2), we assign one operation per processor starting from the lowest-indexed processor. Note that, for any given value of h in the loops defined in steps 2 and 3, the possible number of concurrent operations is $n/2^h = 2^{k-h}$.

The algorithm to be executed by the sth processor is given next. Figure 2.4 illustrates the corresponding processor allocation in the case of $n = 8$ and $p = 2$.

ALGORITHM 2.3

(Algorithm for Processor P_s)

Input: *An array A of size $n = 2^k$, and an index s that satisfies $1 \le s \le p = 2^q$, where $p \le n$ is the number of processors.*

Output: *The prefix sums $C(0, j)$ for $\frac{n}{p}(s-1) + 1 \le j \le \frac{n}{p}s$.*

begin

1. **for** $j = 1$ **to** $l = n/p$ **do**
   Set $B(0, l(s-1) + j) := A(l(s-1) + j)$
2. **for** $h = 1$ **to** k **do**
   2.1. **if** $(k - h - q \ge 0)$ **then**
   for $j = 2^{k-h-q}(s-1) + 1$ **to** $2^{k-h-q}s$ **do**
   Set $B(h, j) := B(h-1, 2j-1) * B(h-1, 2j)$
   2.2. **else** {**if** $(s \le 2^{k-h})$ **then**
   Set $B(h, s) := B(h-1, 2s-1) * B(h-1, 2s)$}
3. **for** $h = k$ **to** 0 **do**
   3.1. **if** $(k - h - q \ge 0)$ **then**
   for $j = 2^{k-q-h}(s-1) + 1$ **to** $2^{k-q-h}s$ **do**

$$\begin{cases} j \text{ even} & : \text{Set } C(h, j) := C\left(h+1, \frac{j}{2}\right) \\ j = 1 & : \text{Set } C(h, 1) := B(h, 1) \\ j \text{ odd} > 1 & : \text{Set } C(h, j) := C\left(h+1, \frac{j-1}{2}\right) * B(h, j) \end{cases}$$

   3.2. **else** $\Big[$**if** $(s \le 2^{k-h})$ **then**

$$\begin{cases} s \text{ even} & : \text{Set } C(h, s) := C\left(h+1, \frac{s}{2}\right) \\ s = 1 & : \text{Set } C(h, 1) := B(h, 1) \\ s \text{ odd} > 1 & : \text{Set } C(h, s) := C\left(h+1, \frac{s-1}{2}\right) * B(h, s) \end{cases}\Big]$$

end

EXAMPLE 2.3:

Let $n = 8$ and let $p = 2$. Consider the algorithm corresponding to processor P_2. At step 1, P_2 sets $B(0, 5) = A(5)$, $B(0, 6) = A(6)$, $B(0, 7) = A(7)$, and

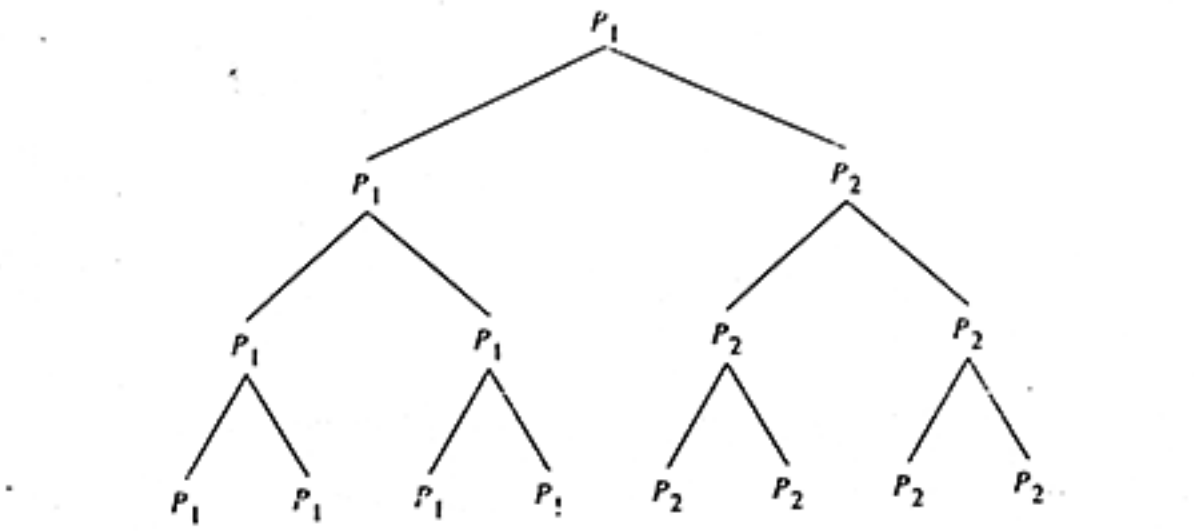

FIGURE 2.4
Processor allocation for computing the prefix sums of eight elements with two processors. The processor indicated at each node is responsible for computing the data generated at that node.

Algorithm 2.3, P_2 will be active for $h = 1, 2$ and idle for $h = 3$. Processor P_2 generates $B(1, 3)$, $B(1, 4)$ and $B(2, 2)$ during the loop defined by step 2. Similarly, during the backward traversal of the binary tree, P_2 will be idle for $h = 3$, and active for $h = 2, 1, 0$. Hence, P_2 generates the sums $C(2, 2)$, $C(1, 3)$, $C(1, 4)$, $C(0, 5)$, $C(0, 6)$, $C(0, 7)$, and $C(0, 8)$ (see Fig. 2.3). □

2.1.4 REVIEW

The basic scheme to build a balanced binary tree on the inputs and to traverse the binary tree to or from the root leads to efficient algorithms for many simple problems. This scheme is one of *the most elementary and the most useful parallel techniques*. We shall use this technique frequently in the rest of this book. Broadcasting a value to all the processors, and compacting the labeled elements of an array, are two simple examples that can be handled efficiently by this scheme (see Exercises 1.8 and 2.3).

The scheme using the balanced binary tree can be generalized to arbitrary balanced trees, where the number of children of an internal node could be nonconstant. As in the binary tree case, a fast algorithm is needed to determine the data stored in an internal node from the data stored in that node's children. It turns out that such a strategy can be carried out successfully for a number of specialized combinatorial problems. Computing the maximum of n elements is one such example discussed in Section 2.4.

APPENDIX E RESUMO ESTENDIDO EM PORTUGUÊS

O uso de imagens em ampla faixa dinâmica (*high dynamic range*, ou abreviadamente, HDR) em computação gráfica vem se intensificando desde o início da década. O problema central com tais imagens é o fato de que a faixa de luminância da imagem não necessariamente corresponde com a faixa de luminância que o dispositivo de exibição é capaz de reproduzir. A solução desse problema encontra-se no emprego de operadores de reprodução de tonalidades (*tone-mapping operators* ou *tone reproduction operators*, abreviadamente, TMO).

O funcionamento de um operador de reprodução de tonalidades é, de certo modo, similar ao processo de adaptação à luminosidade realizado pelo sistema visual humano. Assim como dispositivos de exibição, o sistema visual humano não é capaz de distinguir, simultaneamente, todo a faixa de luminosidade presente no mundo real. Entretanto, diferentemente dos dispositivos de exibição, o olho humano é capaz de se ajustar dinamicamente à determinadas condições de iluminação. Devido a tal imutabilidade na faixa de luminância em que dispositivos são capazes de operar, o projeto de operadores de reprodução de tonalidades torna-se demasiadamente complexo.

Existem duas categorias de operadores, globais e locais. Um operador global procura comprimir a faixa de luminância de uma imagem aplicando o mesmo conjunto de parâmetros sobre todos os pixels da imagem, uniformimente. Já em um operador local, um conjunto ótimo de parâmetros é estabelecido, individualmente, para cada pixel da imagem, um processo que tipicamente involve investigar uma vizinhança de tamanho variável ao redor de cada pixel. Operadores globais são computacionalmente baratos de se executar, embora acabem por suprimir diversos detalhes de mais alta frequência presentes na imagem. Os operadores locais são capazes de reter tais detalhes, mas a um custo computacional muito mais elevado.

Na prática, aplicações interativas tais como jogos, visualizadores e simuladores ficam limitados a operadores globais. Mesmo com o imenso poder de processamento paralelo presente no atual hardware gráfico programável (GPU), a implementação de operadores locais em GPU mostra-se incapaz de manter alto grau de interatividade. O objetivo desta tese é fornecer uma solução satisfatória, implementada em GPU, para a versão local do operador fotográfico digital de reprodução de tonalidades apresentado por Reinhard et al. (REINHARD et al., 2002). A abordagem utilizada faz uso de tabelas de áreas acumuladas (*summed-area tables*, ou abreviadamente, SAT) (CROW, 1984) para aproximar de modo eficiente a computação do modelo de percepção de brilho proposto por Blommaert e Martens (BLOMMAERT; MARTENS, 1990), que corresponde à parte mais custosa do operador fotográfico digital de Reinhard *et al.* Além disso, a tese também proporciona possíveis melhorias aos algoritmos de somas prefixadas, peça-chave na geraçáo eficiente de SATs em GPU.

O operador fotográfico digital de reprodução de tonalidades baseia-se em um processo de revelação de filmes fotográficos conhecido como Sistema de Zonas (*Zone System*) (WHITE; ZAKIA; LORENZ, 1984). O Sistema de Zonas opera através de um mapeamento entre a tonalidade média da cena fotografada e a refletividade de luz média do papel de revelação. Para tanto, o fotógrafo precisa determinar, subjetivamente, o que ele acredita ser o tom de cinza médio da cena (*middle-gray*). Após a revelação, se a fotografia aparecer muito clara ou muito escura, o fotógrafo pode revelá-la novamente ajustando adequadamente o tom de cinza médio. De qualquer modo, tal mapeamento sempre está sujeito à perdas de detalhe contrastivo. Quando isso ocorre, o fotógrafo pode empregar um processo de subexposição-e-superexposição (*dodge-and-burning*) de modo a adaptar localmente o contraste em cada região; um processo bastante exaustivo e de difícil automatição.

Reinhard *et al.*, na versão local de seu operador, automatiza tal processo para fotografias digitais. A busca por detalhes contrastivos é guiada pelo modelo de percepção de brilho apresentado por Blommaert e Martens, que é, por sua vez, implementado através de diferenças de filtros Gaussianos. Tal procedimento mostra-se adequado para a simulação da característica de inibição lateral (*lateral inhibition*) presente no sistema visual humano, que faz com que uma mesma tonalidade pareça mais clara ou escura dependendo das tonalidades presentes ao seu redor. As filtragens Gaussianas necessárias ao operador ocorrem em diversas escalas, tipicamente entre 1 pixel e 47 pixels. Mesmo através da decomposição de filtros Gaussianos e do poder de processamento das atuais GPUs, essas filtragens consomem tempo computacional demasiado, o que impede o emprego do operador em aplicações interativas.

Tentativas para aceleração da versão local do operador fotográfico digital foram investigadas. A primeira delas, apresentada por Goodnight *et al.* utiliza decomposição de filtros Gaussianos e é capaz de reproduzir autenticamente os resultados do operador original, embora apenas um número limitado de filtros faz-se adequado de modo a manter interatividade, limitando significativamente a quantidade de detalhes contrastivos que o operador original é capaz de reproduzir (GOODNIGHT et al., 2003). Outra abordagem, porposta por Krawczyk *et al.*, aproxima tais filtragens através de um processo de sub-amostragem e super-amostragem de modo a condensar o tamanho dos filtros Gaussianos (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005); embora isso acelere o algoritmo de forma substancial, o processo de sub-amostragem acaba por borrar demasiadamente bordas de alto contraste e, consequentemente durante a super-amostragem, mais borramento é introduzido, causando o aparecimento de halos na imagem resultante.

A técnica proposta neste trabalho ameniza consideravelmente a ocorrencia de halos, além de acelerar ainda mais o operador. O componente principal da nova abordagem é a substituição dos filtros Gaussianos por filtros da média que, por sua vez, podem ser eficientemente computados através do emprego de tabelas de áreas acumuladas. Os resultados obtidos são muito próximos e perceptualmente indistingüíveis daqueles produzidos pelo operador original.

Uma tabela de áreas acumuladas é uma tabela cumulativa em que cada elemento corresponde ao somatório de todos os outros elementos acima e à esquerda do elemento em questão na tabela original (CROW, 1984). A principal propriedade de uma SAT é o fato de permitir a obtenção de somas ou filtragens em regiões retangulares arbitrárias em tempo constante $O(1)$ — ou, mais especificamente, através da inspeção de apenas quatro células da SAT. A geração de SATs em paralelo é adequada ao hardware gráfico programável, através de algoritmos de somatório pré-fixado sobre vetores (*prefix sum*), dentre os quais

detaca-se a abordagem baseada em árvores balanceadas (*balanced trees*) proposta por Blelloch (BLELLOCH, 1990). O processo de síntese de uma SAT então restringe-se à aplicação de somatórios pré-fixados sobre cada uma de suas linhas e, posteriormente, sobre cada uma das colunas resultantes.

De modo similar a uma SAT, um somatório pré-fixado representa um vetor cumulativo no qual cada elemento corresponde ao somatório de todos os elementos à esquerda daquele em questão. O algoritmo paralelo de somatório pré-fixado através de árvores balanceadas opera em duas fases: redução e expansão. Na primeira, cada elemento do vetor é tratado como um nodo final da árvore, e nodos-pai são construídos através da acumulação dos valores de dois nodos-filhos; processo esse que se repete até que o nodo-raíz é obtido. O algoritmo então prossegue para a fase de expansão que, partindo do nodo-raiz, constrói uma segunda árvore que, ao final, fará com que os nodos-finais contenham o resultado do somatório pré-fixado.

O processo de expansão é pouco trivial e requer certos cuidados. O nodo-raíz da árvore de expansão é substituído pelo elemento neturo da operação em questão utilizada no somatório pré-fixado. No caso daeração de SATs, tal operação é a adição em $\mathbb{R}$, sendo zero o elemento neutro. Desse nodo raíz, dois nodos-filhos são produzidos, sendo o valor do nodo-pai propagado ao filho da esquerda. O valor do nodo-filho à direita é calculado através da soma do nodo-pai com o valor de seu nodo-irmão (*i.e.*, à esquerda) *na árvore de* ***redução***. O processo então se repete, recursivamente, a cada nodo-filho até que os nodos finais sejam computados.

É importante perceber que o vetor resultante (*i.e.*, os nodos finais da árvore de expansão) não representam exatamente um somatório pré-fixado completo já que o primeiro elemento sempre será o elemento neutro, propagado aos demais elementos. O processo de conversão de tal vetor resultante (*exclusive prefix sum* ou *prescan*) em um somatório pré-fixado completo (*inclusive prefix sum*, ou *scan*) pode ser facilmente obtido de duas maneiras: (i) somando-se o primeiro elemento do vetor original à cada elemento do vetor resultante; ou (ii) deslocando todo o vetor resultante à esquerda e substituíndo seu último elemento pela soma do último elemento do vetor de original com o penúltimo elemento do vetor resultante deslocado.

Em uma execução seqüencial, tal conversão é de ordem $O(n)$; já uma execução paralela resulta em $O(1)$, mas requer sincronização ou memória adicional de modo a preservar sua semântica. O presente trabalho apresenta uma melhoria ao algoritmo de modo que, durante a fase de expansão, o vetor resultante corresponda à um somatório pré-fixado completo, elmininando a necessidade de conversões. A modificação proposta funciona para o caso do operador de adição em $\mathbb{R}$ e não afeta a estrutura geral do algoritmo.

Em sumo, em vez de substituir e propagar o elemento neutro durante a expansão, o valor bruto acumulado na raíz da árvore de redução é propagado à raiz da árvore de expansão. Ambos os nodos-filhos são produzidos, mas o valor do nodo-pai é propagado ao filho da direita. O valor do nodo-filho à esquerda é estabelecido através da *subtração* do valor do nodo-pai pelo valor do seu respectivo nodo-irmão (à direita) na árvore de redução. Repetindo-se esse processo recursivamente, ao final da fase de expansão, um somatório pré-fixado completo é produzido. Sem a necessidade de tais conversões, o ganho em performance pode chegar até 50% quando empregado na geração de uma SAT.

Uma vez garantida uma implementação eficiente para a geração de SATs em GPU, a modificação proposta neste trabalho sobre a versão local do operador fotográfico de reprodução de tonalidades torna-se simples. As filtragens Gaussianas, de tamanhos variados, sobre a imagem de luminância relativa requeridas pelo operador são substituídas

por consultas à SAT correspondente. O restante do operador permanece intacto, salvo ao parâmetro limiar (*threshold*) que controla a busca pela vizinhança isoluminante ideal: de modo geral, não é necessário ajustar o limiar padrão oferecido pelo operador original, mas resultados mostram que limiares até duas vezes menores proporcionam mais fidelidade quando comparado aos resultados do operador original. A explicação de tal fato vem da observação de que filtros da média resultam em contribuições maiores (uniforme) de luminâncias na vizinhança de cada pixel do que àqueles obtidos através de filtragens Gaussianas. O parâmetro de limiar, presente no operador original, proporciona um método integrado para ajustar o resultado final, se necessário.

Por fim, as modificações propostas sobre a versão local do operador fotográfico digital de reprodução de tonalidades não apenas permitem níveis bastante satisfatórios de interatividade — de duas até dez vezes mais rápido do que as abordagens propostas por Krawczyk *et al.* (KRAWCZYK; MYSZKOWSKI; SEIDEL, 2005) e Goodnight *et al.* (GOODNIGHT et al., 2003), respectivamente — como também proporcionam um alto grau de fidelidade quando comparado ao operador original proposto por Reinhard *et al.* (REINHARD et al., 2002). O presente trabalho oferece uma solução atrativa para aplicações HDR que necessitam alta performance sem comprometer a qualidade das imagens exibidas.

A aproximação das filtragens Gaussianas, requeridas pelo modelo de percepção de brilho de Blommaert e Martens, através de filtros da média computados com o auxílio de tabelas de áreas acumuladas, não se restringe apenas à reprodução de tonalidades. Acredita-se que tal modelo pode ser aplicado em outros contextos para acelerar processos tais como: (i) geração automática de ilustrações (GOOCH; REINHARD; GOOCH, 2004); (ii) detecção de bordas significativas (*feature line detection*) (LEGEAI, 2005); (iii) e redirecionamento de imagens em dispositivos (*retargeting*) (AVIDAN; SHAMIR, 2007).

REFERENCES

ADAMS, A. **The Print**. [S.l.]: Little, Brown and Company, 1983. (The Ansel Adams Photography).

ASHIKHMIN, M. A tone mapping algorithm for high contrast images. In: EUROGRAPHICS WORKSHOP ON RENDERING, EGRW, 13., 2002, Aire-la-Ville, Switzerland, Switzerland. **Proceedings...** Eurographics Association, 2002. p.145–156.

AVIDAN, S.; SHAMIR, A. Seam carving for content-aware image resizing. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.26, n.3, p.10, 2007.

Bit-Tech. **BrightSide DR37-P HDR display**. Online resource, available at: <http://www.bit-tech.net>. Last accessed in Feb. 2008.

BLELLOCH, G. E. **Prefix Sums and Their Applications**. [S.l.]: Synthesis of Parallel Algorithms, 1990.

BLOMMAERT, F. J.; MARTENS, J.-B. **An object-oriented model for brightness perception**. [S.l.]: VSP, an imprint of Brill, 1990. 15–41p. v.5, n.1.

BLYTHE, D. The Direct3D 10 system. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.25, n.3, p.724–734, 2006.

BOURKE, P. **Unofficial PBM format for HDR images**. Online resource, available at: <http://local.wasp.uwa.edu.au/~pbourke>. Last accessed in Feb. 2008.

CEBENOYAN, C. **Floating-Point Specials on the GPU**. [S.l.: s.n.], 2005.

CGSD. **Gamma Correction**. Online resource, available at: <http://www.cgsd.com/papers/gamma.html>. Last accessed in Feb, 2008.

CHEN, J.; PARIS, S.; DURAND, F. Real-time edge-aware image processing with the bilateral grid. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 34., 2007, New York, NY, USA. **Proceedings...** ACM, 2007. p.103.

CHIU, K. et al. Spatially nonuniform scaling functions for high contrast images. In: GRAPHICS INTERFACE, GI, 12., 1993. **Proceedings...** [S.l.: s.n.], 1993. p.245–253.

CRAWFORD, B. H. The Scotopic Visibility Function. **The Physical Society, Section B**, [S.l.], v.62, n.5, p.321, 1949.

CROW, F. C. Summed-area tables for texture mapping. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 11., 1984, New York, NY, USA. **Proceedings...** ACM, 1984. p.207–212.

Crytek. **CryENGINE 2**. Online resource, available at: <http://www.crytek.com>. Last accessed in Feb. 2008.

DEBEVEC, P. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 25., 1998, New York, NY, USA. **Proceedings...** ACM, 1998. p.189–198.

DEBEVEC, P. E.; MALIK, J. Recovering high dynamic range radiance maps from photographs. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 24., 1997, New York, NY, USA. **Proceedings...** ACM Press/Addison-Wesley Publishing Co., 1997. p.369–378.

DEVLIN, K. **A Review of Tone Reproduction Techniques**. 2002.

Dolby Laboratories. **High-dynamic-range displays**. Online resource, available at: <http://www.dolby.com>. Last accessed in Feb. 2008.

DRAGO, F. et al. Adaptive Logarithmic Mapping For Displaying High Contrast Scenes. **Computer Graphics Forum**, [S.l.], v.22, p.419–426, 2003.

DUBOIS, P.; RODRIGUE, G. An analysis of the recursive doubling algorithm. **High Speed Computer and Algorithm Organization**, [S.l.], p.299–307, 1977.

DURAND, F.; DORSEY, J. Fast bilateral filtering for the display of high-dynamic-range images. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 29., 2002, New York, NY, USA. **Proceedings...** ACM, 2002. p.257–266.

EISEMANN, E.; DÉCORET, X. **Occlusion Textures for Plausible Soft Shadows**. [S.l.: s.n.], 2008. 13–23p. v.27, n.1.

Epic Games. **Unreal Engine 3**. Online resource, available at: <http://www.unrealtechnology.com>. Last accessed in Feb. 2008.

FATTAL, R.; LISCHINSKI, D.; WERMAN, M. Gradient domain high dynamic range compression. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 29., 2002, New York, NY, USA. **Proceedings...** ACM, 2002. p.249–256.

FERWERDA, J. A. et al. A model of visual adaptation for realistic image synthesis. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 23., 1996, New York, NY, USA. **Proceedings...** ACM, 1996. p.249–258.

GILCHRIST, A. et al. An anchoring theory of lightness perception. **Psychological Review**, [S.l.], 1999.

GOOCH, B.; REINHARD, E.; GOOCH, A. Human facial illustrations: creation and psychophysical evaluation. **ACM Transactions on Graphics (TOG)**, [S.l.], v.23, p.27–44, 2004.

GOODNIGHT, N. et al. A multigrid solver for boundary value problems using programmable graphics hardware. In: ACM SIGGRAPH/EUROGRAPHICS CONFERENCE ON GRAPHICS HARDWARE, HWWS, 1., 2003, Aire-la-Ville, Switzerland, Switzerland. **Proceedings...** Eurographics Association, 2003. p.102–111.

GOODNIGHT, N. et al. Interactive time-dependent tone mapping using programmable graphics hardware. In: ACM SIGGRAPH COURSES, SIGGRAPH, 32., 2005, New York, NY, USA. **Anais...** ACM, 2005. p.180.

HARRIS, M. **Parallel Prefix Sum (Scan) with CUDA**. [S.l.]: NVIDIA Corporation, 2007.

HARRIS, M. et al. **The CUDA Data Parallel Primitives Library**. Online resource, available at: <http://www.gpgpu.org>. Last accessed in Feb. 2008.

HENSLEY, J. et al. Fast Summed-Area Table Generation and its Applications. **Computer Graphics Forum**, [S.l.], v.24, n.3, p.547–555, September 2005.

HOLZER, B. **High Dynamic Range Image Formats**. Technische Universität Wien: [s.n.], 2006.

HUNT, R. W. G. **The Reproduction of Colour**. [S.l.]: Fountain Press, 1996.

Industrial Light and Magic. **An OpenEXR technical introduction**. Online resource, available at: <http://www.openexr.com>. Last accessed in Feb. 2008.

Irrlicht Team. **Irrlicht Engine**. Online resource, available at <http://irrlicht.sourceforge.net>. Last accessed in Feb. 2008.

KRAWCZYK, G.; MYSZKOWSKI, K.; SEIDEL, H.-P. Perceptual Effects in Real-Time Tone Mapping. In: SPRING CONFERENCE ON COMPUTER GRAPHICS, SCCG, 21., 2005, Budmerice, Slovakia. **Proceedings...** ACM, 2005. p.195–202.

LAURITZEN, A. Summed-Area Variance Shadow Maps. **GPU Gems 3**, [S.l.], 2007.

LEDDA, P. et al. Evaluation of tone mapping operators using a High Dynamic Range display. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.24, n.3, p.640–648, 2005.

LEDDA, P.; SANTOS, L. P.; CHALMERS, A. A local model of eye adaptation for high dynamic range images. In: COMPUTER GRAPHICS, VIRTUAL REALITY, VISUALISATION AND INTERACTION IN AFRICA, AFRIGRAPH, 3., 2004, New York, NY, USA. **Proceedings...** ACM, 2004. p.151–160.

LEFOHN, A. E. et al. Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware. In: IEEE VISUALIZATION, VIS, 14., 2003, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2003. p.11.

LEGEAI, A. **Geometric Feature Lines Evaluation and Selection Using Perceptual Criteria**. [S.l.: s.n.], 2005.

LONDON, B.; UPTON, J. **Photography**. [S.l.]: Addison-Wesley Pub Co, 1998.

MEIJER, H.; AKL, S. G. Optimal computation of prefix sums on a binary tree of processors. **International Journal of Parallel Programming**, Norwell, MA, USA, v.16, p.127–136, April 1987.

Microsoft Corporation. **HDR Lighting (Direct3D 9.0c)**. Online resource, available at: <http://msdn.microsoft.com>. Last accessed in Feb. 2008.

MILLER, N. J.; NGAI, P. Y.; D., M. D. The application of computer graphics in lighting design. **Journal of the IES**, [S.l.], v.14, p.6–26, 1984.

MITCHELL, E. **Photographic Science**. New York: John Wiley and Sons, 1984.

NG, R.; RAMAMOORTHI, R.; HANRAHAN, P. Triple product wavelet integrals for all-frequency relighting. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.23, n.3, p.477–487, 2004.

NVIDIA Corporation. **NVIDIA Shader Library**. Online resource, available at: <http://developer.nvidia.com>. Last accessed in Feb. 2008.

NVIDIA Corporation. **NVIDIA Software Development Kit**. Online resource, available at: <http://developer.nvidia.com>. Last accessed in Feb. 2008.

NVIDIA Corporation. **CUDA**. 2007.

PATTANAIK, S. N. et al. A multiscale model of adaptation and spatial vision for realistic image display. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 25., 1998, New York, NY, USA. **Proceedings...** ACM, 1998. p.287–298.

PATTANAIK, S. N. et al. Time-dependent visual adaptation for fast realistic image display. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 27., 2000, New York, NY, USA. **Proceedings...** ACM Press/Addison-Wesley Publishing Co., 2000. p.47–54.

PATTANAIK, S.; YEE, H. Adaptive gain control for high dynamic range image display. In: COMPUTER GRAPHICS, SCCG, 18., 2002, New York, NY, USA. **Proceedings...** ACM, 2002. p.83–87.

PEERS, P. et al. Post-production facial performance relighting using reflectance transfer. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.26, n.3, p.52, 2007.

PELI, E. Contrast in complex images. **Journal of Optical Society of America**, [S.l.], p.2032–2040, 1990.

POYNTON, C. **Digital Video and HDTV Algorithms and Interfaces**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2003.

RAMAMOORTHI, R.; HANRAHAN, P. An efficient representation for irradiance environment maps. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 28., 2001, New York, NY, USA. **Proceedings...** ACM, 2001. p.497–500.

REINHARD, E.; DEVLIN, K. Dynamic Range Reduction Inspired by Photoreceptor Physiology. **IEEE Transactions on Visualization and Computer Graphics**, Piscataway, NJ, USA, v.11, n.1, p.13–24, 2005.

REINHARD, E. et al. Photographic tone reproduction for digital images. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.21, n.3, p.267–276, 2002.

REINHARD, E. et al. **High Dynamic Range Imaging**: acquisition, display and image-based lighting. [S.l.]: Morgan Kaufmann Publishers, 2006.

Reuters. **Shift to large LCD TVs over plasma**. Online, last accessed in Feb, 2008, MSNBC.

SCHLICK, C. Quantization Techniques for Visualization of High Dynamic Range Pictures. In: EUROGRAPHICS WORKSHOP ON RENDERING, EGRW, 5., 1994. **Proceedings...** Springer-Verlag, 1994. p.7–20.

SEETZEN, H. et al. High dynamic range display systems. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.23, n.3, p.760–768, 2004.

SEETZEN, H.; WHITEHEAD, L. A.; WARD, G. A high dynamic range display using low and high resolution modulators. **Society for Information Display Digest (SID)**, [S.l.], v.34, n.1, p.1450–1453, 2003.

SENGUPTA, S. et al. Scan Primitives for GPU Computing. In: GRAPHICS HARDWARE 2007. **Anais...** Association for Computing Machinery, 2007. p.97–106.

SENGUPTA, S.; LEFOHN, A.; OWENS, J. D. A Work-Efficient Step-Efficient Prefix Sum Algorithm. In: WORKSHOP ON EDGE COMPUTING USING NEW COMMODITY ARCHITECTURES, 1., 2006. **Proceedings...** [S.l.: s.n.], 2006. p.D–26–27.

SHLAER, S. The relation between visual acuity and illumination. **The Journal of General Physiology**, [S.l.], v.21, p.165–188, 1937.

SLOAN, P.-P.; KAUTZ, J.; SNYDER, J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 29., 2002, New York, NY, USA. **Proceedings...** ACM, 2002. p.527–536.

SLOMP, M.; OLIVEIRA, M. M. A Gentle Introduction to Precomputed Radiance Transfer. **RITA, Revista da Informática Teórica e Aplicada**, [S.l.], 2005.

STEVENS, S. S.; STEVENS, J. C. Brightness function: parametric effects of adaptation and contrast. **Journal of the Optical Society of America**, [S.l.], 1960.

Toshiba Corporation; Canon Inc. **SED, the surface-conduction electron-emitter display**. Online resource, available at: <`http://www.toshiba.co.jp`>. Last accessed in Feb. 2008.

TUMBLIN, J.; HODGINS, J. K.; GUENTER, B. K. Two methods for display of high contrast images. **ACM Transactions on Graphics (TOG)**, New York, NY, USA, v.18, n.1, p.56–94, 1999.

TUMBLIN, J.; RUSHMEIER, H. Tone Reproduction for Realistic Images. **IEEE Computer Graphics and Applications**, Los Alamitos, CA, USA, v.13, p.42–48, 1993.

Unigine Corp. **Unigine Engine**. Online resource, available at: <`http://unigine.com`>. Last accessed in Feb. 2008.

UPSTILL, S. D. **The realistic presentation of synthetic images**: image processing in computer graphics. 1985. Tese (Doutorado em Ciência da Computação) — .

Valve Software. **Source engine**. Online resource, available at <http://www.valvesoftware.com>. Last accessed in Feb. 2008.

WARD, G. A contrast-based scalefactor for luminance display. **Graphics Gems IV**, San Diego, CA, USA, p.415–421, 1994.

WARD, G. The RADIANCE lighting simulation and rendering system. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 21., 1994, New York, NY, USA. **Proceedings...** ACM, 1994. p.459–472.

WARD, G. **Real Pixels**. [S.l.]: Academic Press, 1994. (Academic Press Graphics Gems Series).

WARD, G. LogLuv encoding for full gamut, high dynamic range images. **Journal of Graphics Tools (JGT)**, Natick, MA, USA, v.3, n.1, p.15–31, 1998.

WARD, G. **High Dynamic Range Image Encodings**. Online resource, available at: http://www.anyhere.com>. Last accessed in Feb. 2008.

WARD LARSON, G.; RUSHMEIER, H.; PIATKO, C. A visibility matching tone reproduction operator for high dynamic range scenes. In: VISUAL PROCEEDINGS: THE ART AND INTERDISCIPLINARY PROGRAMS OF SIGGRAPH, 24., 1997, New York, NY, USA. **Anais...** ACM, 1997. p.155.

WHITE, M.; ZAKIA, R.; LORENZ, P. **The new zone system manual**. [S.l.]: Morgan & Morgan, 1984.

WIEGAND, C.; WALOSZEK, G. **Color Glossary**. Online resource, available at: <http://www.sapdesignguild.org>. Last accessed in Feb. 2008.

WOODS, J. C. **The zone system craftbook**. [S.l.]: McGraw Hill, 1993.

YEE, Y. H.; PATTANAIK, S. N. Segmentation and adaptive assimilation for detail-preserving display of high-dynamic range images. **The Visual Computer**, [S.l.], v.19, p.457–466, 2003.

ZHANG, X.-M. et al. Applications of a spatial extension to CIELAB. **Society for Information Display Digest (SID)**, [S.l.], p.61–63, 1997.