Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

High-Dynamic-Range Visualization of Density Maps

Marc Rivinius

Course of Study: Informatik

Examiner: Prof. Dr. Daniel Weiskopf

Supervisor: Liang Zhou, Ph.D.

Commenced: April 27, 2017

Completed: October 27, 2017

Abstract

Density maps are an important means of data representation and have been widely used in various visualizations, e.g., scatter plots, parallel coordinates, and trajectories. Typically, density maps have high dynamic-ranges which are beyond the displayable intensities on a monitor. The common operators to map the data values to displayable intensities (for example, linear, logarithmic, and gamma mappings) do not work in all situations and produce unsatisfactory results, where features may be lost or misleading visualizations may be created. Therefore, we propose a perceptual-based model to better visualize high-dynamic-range density maps: we map high-dynamic-range data to a displayable range through a perceptual tone mapping operator; on top of that, we apply glare simulation to highlight high-density regions which are found by our automatic bright pixel detector. The glare is used to highlight high-density regions, while the tone mapping preserves structural details. In addition, we evaluate different tone mapping operators on density maps in typical data visualizations, which has not been studied to the best of our knowledge. For the whole approach, an efficient GPU-based implementation and an easy-to-use application with intuitive user interactions are provided. We demonstrate the effectiveness of our method through a wide range of density map visualizations.

Contents

1 Introduction **9**

2 Background and Related Work **13**
2.1 Definitions and Notation 13
2.2 Density Map Visualization 14
2.3 Tone Mapping Techniques 15
2.4 Glare Simulation 16
2.5 Blob Detection 17

3 Evaluation of Tone Mapping Operators on Density Maps **19**

4 Perceptual High-Dynamic-Range Density Map Visualization Model **25**
4.1 Pipeline 25
4.2 Tone Mapping 26
4.3 Bright Pixel Detection 30
4.4 Glare Simulation 33

5 User Interaction **37**

6 GPU-Based Implementation **41**
6.1 Data Structures 41
6.2 Image Files 43
6.3 Implementation of the Pipeline 43
6.4 Tone Mapping 44
6.5 Bright Pixel Detection 44
6.6 Glare Simulation 46

7 The HDR-Toolkit System **51**
7.1 User Interface 51
7.2 Image Files 53
7.3 Other Features 53

8 Evaluation and Examples **55**

9 Conclusion and Future Work **61**
9.1 Conclusion 61
9.2 Future Work 61

A Zusammenfassung **65**

Bibliography **67**

List of Abbreviations

CPU central processing unit. 41

FFT fast Fourier transform. 14

GPU graphics processing unit. 41

HDR high-dynamic-range. 9

LDR low-dynamic-range. 9

PSF point spread function. 16

TMO tone mapping operator. 9

UI user interface. 51

XAML Extensible Application Markup Language. 51

1 Introduction

Density maps, which are generated by accumulating data values, are an important data source in visualization. Various visualizations make use of density maps, for example, scatter plots, parallel coordinates, and trajectories. Due to the high-dynamic nature of density maps, some kind of mapping methods have to be applied to visualize density maps on a monitor using the displayable intensities. Common mapping approaches use simple math functions which do not take human visual perception into account and therefore may result in misleading visualizations as important features are lost or relative value differences cannot be perceived. Instead, we treat the problem as a special high-dynamic-range (HDR) imaging problem with a focus on human visual perception. HDR is a long-standing research topic in computer graphics and techniques developed there can be applied to density maps. The process of transforming HDR images to displayable low-dynamic-range (LDR) images is called tone mapping, and the techniques to do that are called tone mapping operators (TMOs) [Ča+08]. However, TMOs are typically applied to images with natural scenes, which are very different from images generated in data visualizations, where features of high spatial frequencies are of high dynamic-range (e.g., a scatter plot which describes the 2D histogram of data consists of pixel-sized entries containing data item counts, which may range from zero up to the number of all items in the data). Therefore, a good TMO for density map visualization has to be found, with which the LDR image is close to the HDR image as it would be perceived by a human: All features of an HDR image are preserved and relative value differences are kept.

Human perception of luminance roughly follows the Weber–Fechner law, which implies logarithmic relationship between the physical luminance and the perceived brightness [SW89]. Therefore, even with a perceptually based TMO, perceiving differences between high intensity pixels is difficult due to the limited physical luminance of display devices. For example, a latest 4K UHD monitor[1] has a highest luminance of $370\,\mathrm{cd\,m^{-2}}$. In real life, we are able to recognize that the sun or other light sources having higher luminance than objects that are not light sources because of glare. Artificial glares can be added into a computer synthesized image to simulate the human perception of high-luminance objects [Spe+95]. These could also be applied to the density map images to highlight pixels of very high densities.

To find the bright regions in an image which are glared, no hand-made mask (like in the implementation of Frisvad [Fri09]) or just a simple luminance threshold (as suggested by

[1]Samsung LU28D590DS/ZA monitor, see http://www.samsung.com/us/computer/monitors/LU28D590DS/ZA-specs.

Spencer et al. [Spe+95]) should be used. The first technique requires manual work and the second technique requires knowledge about the image. An automatic blob detection in combination with a luminance threshold seems to be more appropriate than the previous techniques.

In this thesis, a novel HDR method to visualize density maps is proposed. We map HDR data to a displayable range through a perceptual tone mapping operator. High-density regions that have similar luminance are found automatically by a scale-space-based bright pixel detector. Glare simulations are then applied to these detected regions to resolve ambiguous luminance. We study HDR imaging techniques together with the simulation of glare in the context of data visualization. We experiment with different parameter settings and study their effects on various types of density map datasets. The contributions of this work are

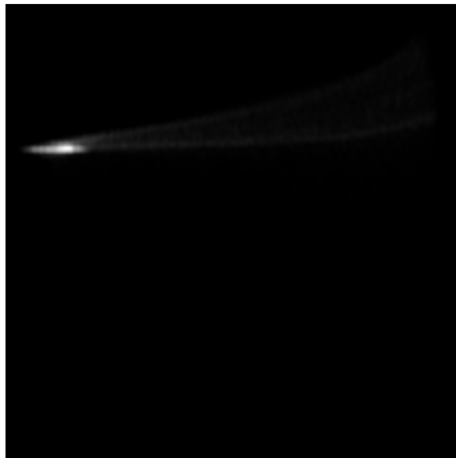
1. a perceptual model based on human vision for density visualization,
2. an efficient GPU-based implementation, and
3. an evaluation of tone mapping operators on visualization data.

Our method has the following benefits. Features are preserved perceptually in density maps: Many details can be seen, while the perceived intensity difference is kept by glares. The method is data-independent; therefore, the approach can be applied to results of any visualization pipeline. The user interaction is easy and intuitive—in fact, only four sliders are required to fully manipulate the method.

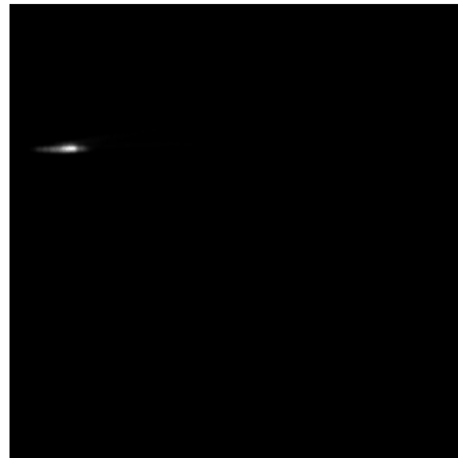
In Figure 1.1, a comparison of our method to common mapping techniques (see Figures 1.1a to 1.1c) and a perceptually based TMO (see Figure 1.1d) on "Isabel" data[2] can be seen. With our method (see Figure 1.1e), it is clearly visible that more details are visible compared to the common mapping techniques thanks to the TMO. This includes large parts of the image that are not even visible using the common techniques. Additionally, the high-density pixels can be easily seen with our method, just as with the common techniques.

The foundations and related work are presented in Chapter 2. Different tone mapping operators are evaluated on density maps in Chapter 3. The perceptual model to visualize such data is proposed in Chapter 4. The possible interactions of the approach are described in Chapter 5. Implementation details of the system that realizes our method are outlined in Chapter 6, an evaluation of the approach and example results can be found in Chapter 8, and the work is concluded with Chapter 9. A summary in German can be found in Appendix A.

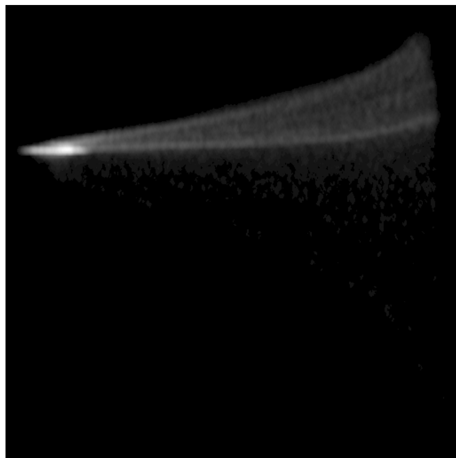
[2]Hurricane Isabel data from the IEEE Visualization 2004 Contest, see http://vis.computer.org/vis2004contest/data.html.



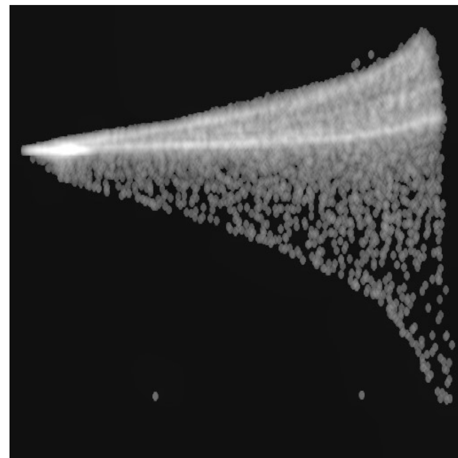
(a) Using linear mapping.



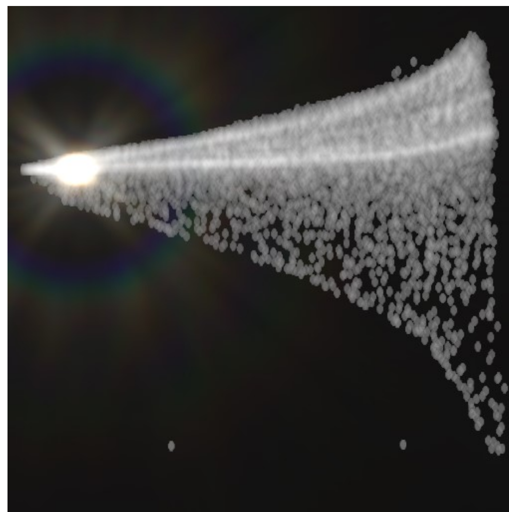
(b) Using logarithmic mapping.



(c) Using gamma mapping.



(d) Using a perceptual TMO [MMS06].



(e) Using our proposed method.

Figure 1.1: Comparison of different visualization methods.

2 Background and Related Work

The necessary foundation for the rest of this work is given in this chapter. Definitions and notation are explained first in Section 2.1. Related work of density map visualization, tone mapping, and glare simulation are briefly discussed in Sections 2.2 to 2.4.

2.1 Definitions and Notation

Coordinates Coordinates are denoted as an explicit pair of coordinates (x, y) or a vector of coordinates $\mathbf{x}$. This notation allows the use of vector operations like the scalar product $\langle \cdot, \cdot \rangle$, multiplication with scalar values, the Euclidean norm $\|\cdot\|$, or the maximum norm $\|\cdot\|_\infty$. All images are noted in function notation, meaning the pixel at location $\mathbf{x}_j = (x_n, y_m)$ of an image f is $f(\mathbf{x}_j) = f(x_n, y_m)$, which is $f_j = f_{n,m}$ in subscript notation sometimes used for discrete images.

High-Dynamic-Range Image The HDR input image of width N_x and height N_y can be defined as a mapping from the image domain

$$\mathbb{D} = [0, N_x - 1] \times [0, N_y - 1] \subset \mathbb{N}^2 \tag{2.1}$$

to a color co-domain. The color is assumed to be in the RGB color space, as colors from other color spaces like the HSV color space can be converted to RGB colors. The HDR image is then

$$I : \mathbb{D} \to \mathbb{R}^3,\ \mathbf{x} \mapsto I(\mathbf{x}) = (r, g, b). \tag{2.2}$$

The RGB values in HDR images are usually inferred from the scene that is represented by the image. This means these values can represent the scene linearly, which is not possible with a limited dynamic-range and fixed precision. This means HDR images can be used to store any two-dimensional color data without any modification.

Low-Dynamic-Range Image The HDR input image has to be converted in an LDR image to be viewed on standard displays or prints. The process of reducing the dynamic-range of an image is called tone mapping. The functions to accomplish this are called TMOs. An LDR image can be defined analogously to HDR images as

$$\hat{I} : \mathbb{D} \to [0, 1]^3 \subset \mathbb{R}^3,\ \mathbf{x} \mapsto \hat{I}(\mathbf{x}) = (\hat{r}, \hat{g}, \hat{b}). \tag{2.3}$$

The color values can then be linearly mapped to the integer values that represent the color intensity of displays or printers.

Luminance The luminance of a color in linear RGB color space is defined as

$$L : \mathbb{R}^3 \to \mathbb{R},\ (r, g, b) \mapsto 0.2126 \cdot r + 0.7152 \cdot g + 0.0722 \cdot b \tag{2.4}$$

and represents the brightness of an image [Int15b]. Only luminance is modified in most tone mapping operators. As a simplified notation $L(\mathbf{x})$ is used for the luminance of an image $I(\mathbf{x})$ and $\hat{L}(\mathbf{x})$ for the tone mapped luminance that is used to compute the tone mapped image $\hat{I}(\mathbf{x})$.

Fourier Transform The two-dimensional continuous Fourier transform is defined as

$$\mathcal{F}[f(x,y)]\,(u,v) = \iint_{-\infty}^{\infty} f(x,y) \cdot \exp\left(-2\pi i(ux+vy)\right)\,\mathrm{d}x\,\mathrm{d}y \tag{2.5}$$

and the inverse Fourier transform as

$$\mathcal{F}^{-1}[f(u,v)]\,(x,y) = \iint_{-\infty}^{\infty} f(u,v) \cdot \exp\left(2\pi i(ux+vy)\right)\,\mathrm{d}u\,\mathrm{d}v \tag{2.6}$$

for an input f. The Fourier transform is evaluated at position (u, v) and the inverse transform is evaluated at (x, y). The discrete versions are the discrete two-dimensional Fourier transform

$$\mathcal{F}[f(x,y)]\,(u,v) = \sum_{x=0}^{N_x-1} \sum_{y=0}^{N_y-1} f(x,y) \cdot \exp\left(-2\pi i\left(\frac{ux}{N_x} + \frac{vy}{N_x}\right)\right) \tag{2.7}$$

and the inverse discrete Fourier transform

$$\mathcal{F}^{-1}[f(u,v)]\,(x,y) = \frac{1}{N_x \cdot N_y} \sum_{u=0}^{N_x-1} \sum_{v=0}^{N_y-1} f(u,v) \cdot \exp\left(2\pi i\left(\frac{ux}{N_x} + \frac{vy}{N_x}\right)\right). \tag{2.8}$$

The two-dimensional Fourier transform is separable and is usually implemented with two consecutive one-dimensional fast Fourier transforms (FFTs), which requires the image dimensions to be powers of two.

2.2 Density Map Visualization

A density map maps data dimensions to a density value. Many visualization techniques use density maps, e.g., scatter plots [Cha83], scatter plot matrices [EDF08], parallel coordinates [NH06], and trajectory plots [AA05]. Most often, density maps are discrete and are visualized by either first accumulating data values and then mapping to optical properties, i.e., luminance and color, or mapping data values to optical properties first and then performing blending of optical properties. Continuous density maps can be computed using advanced models for scatter plots [BW08] or parallel coordinates [HW09]. Although methods that generate density maps have been well studied, mappings from data values to optical properties are not. Usually, simple linear or logarithmic mappings are used in

existing visualization methods. Human visual perception is typically not considered by these methods. Even the more advanced techniques [BW08; HW09] to display density maps use just a simple logarithmic mapping from density to color (like Equation (2.12)). Due to lack of research, our method attempts to improve density map visualizations with a human visual perception-based model.

2.3 Tone Mapping Techniques

The data that has to be visualized is given as an HDR image and could have arbitrary values and precision. Tone mapping has to be used to transform these HDR values to fixed-precision integer values that can be used as colors in displays or printers. Typical LDR values are saved with $8\,\mathrm{bit}$ per color channel with values ranging from 0 to 255. As the final step of transforming values from the range $[0,1]$ to these displayable values is always the same, tone mapping is defined as transforming HDR images (see Equation (2.1)) to LDR images (see Equation (2.2)). Usually, a transfer function has to be used to get from linear values in the range $[0,1]$ to perceptually linear values in the range $[0,1]$. A color transfer function [Int15a] like

$$C : \mathbb{R} \supset [0,1] \to [0,1] \subset \mathbb{R},\, c \mapsto \begin{cases} 12.92 \cdot c\ , & c \leq 0.0031308 \\ 1.055 \cdot c^{1/2.4} - 0.055\ , & \text{otherwise} \end{cases} \tag{2.9}$$

or a *gamma correction* function

$$C_\gamma : \mathbb{R} \supset [0,1] \to [0,1] \subset \mathbb{R},\, c \mapsto c^{1/\gamma}\ , \tag{2.10}$$

with $\gamma = 2.2$ applied to all color channels before transforming them to integer values is used. These new values are then multiplied by 255 and quantized.

Not only has a tone mapping operator to bring the input colors and luminance in the display range, it also tries to preserve or enhance certain image properties. These properties can include contrast, high detail, or a visually appealing appearance. Some operators use information from a local neighborhood to achieve that, while others only work with some global information.

Some simple global tone mapping operators can be defined using the minimum luminance L_{min} and maximum luminance L_{max}. These operators include linear mapping

$$\hat{I}(\mathbf{x}) = \frac{I(\mathbf{x}) - L_{\text{min}}}{L_{\text{max}} - L_{\text{min}}}, \tag{2.11}$$

logarithmic mapping

$$\hat{I}(\mathbf{x}) = \frac{\log(I(\mathbf{x}) + 1)}{\log(L_{\text{max}} + 1)}, \tag{2.12}$$

the simple tone mapping operator of Reinhard et al. [Rei+02]

$$\hat{I}(\mathbf{x}) = \frac{I(\mathbf{x})}{1 + L(\mathbf{x})}, \tag{2.13}$$

and *gamma correction*

$$\hat{I}(\mathbf{x}) = I(\mathbf{x})^{1/\gamma}. \tag{2.14}$$

The three operators of Equations (2.11) to (2.13) bring the luminance to the range $[0, 1]$. In the original version of Equation (2.13) proposed by Reinhard et al., the image and the luminance are both multiplied by the desired key of the image between zero and one and divided by the log-average luminance of the image.

The *gamma correction* of Equation (2.14) can be used to map values from the range $[0, 1]$ to the range $[0, 1]$ in a non-linear way. Another tone mapping operator like linear mapping should be applied before the *gamma correction* to bring the values to the desired range. Generally, values of $\gamma > 1$ are used to bright up images and values of $\gamma < 1$ are used to darken images.

A detailed evaluation of tone mapping operators was done by Čadík et al. [Ča+08]. Some of the TMOs are evaluated specifically for density maps by this work in Chapter 3. Notably good results were achieved using the local operator of Reinhard et al. [Rei+02]. This tone mapping operator is similar to the global operator of Equation (2.13), but the divisor is calculated individually for each pixel using local averages computed with Gaussian filters of different sizes. This operator can produce artifacts, which is noted as the main downside of this operator.

Another tone mapping technique that yields good result was introduced by Mantiuk, Myszkowski, and Seidel [MMS06]. They proposed a framework based on the human perception of contrast. The input image is first transformed into the contrast space. A transducer function is devised to convert physical contrast to a linear perceptual response space with just-noticeable-difference as unit. Therefore, the image can be easily manipulated in the response space; and then, the image is transformed from the response space back to a displayable image by solving an optimization problem. Naturally, one application of the framework is tone mapping and it yields good results for density maps as seen in Chapter 3. Thanks to the use of low-pass images, the framework avoids artifacts in the tone mapped image.

2.4 Glare Simulation

Glare can be used to make bright pixels in images appear brighter [Rit+09; Spe+95]. Spencer et al. [Spe+95] divided glare into *bloom* and *flare*. *Bloom* or *veiling luminance* is a glow that reduces the contrast around bright objects. *Flare* is composed of the *lenticular halo* and the *ciliary corona*. The concentric colored rings around bright objects are called

lenticular halo and the rays or needles radiating from bright objects are called *ciliary corona*.

Spencer et al. proposed functions to simulate these features trying to match phenomenological results. The functions form a point spread function (PSF) or glare filter that has to be convolved with the image. The PSF can be customized using a few parameters (the light adaption state of the viewer and the age of the viewer). The glare filter should be only applied to bright pixels.

Newer approaches [Kak+04; Rit+09] are based on wave-optics. These approaches model an optical system and calculate the diffraction caused by an aperture. This diffraction is used to calculate a glare filter. To simulate glare as seen by humans, the aperture can be an image containing the parts of the eye that contribute to glaring [see Rit+09, Table 1]. Kakimoto et al. only used the pupil, eyelids, and eyelashes in their model of a human aperture. Ritschel et al. added lens particles, lens gratings, and vitreous particles to their model.

Ritschel et al. also proposed a way to create a dynamic aperture by simulating the movement of the particles, changes of the pupil size, and blinking. Using this dynamic aperture, the simulated glare changes slightly with time, which mimics real glare perceived by humans. Dynamic glare could be perceived brighter than static glare [Rit+09]. To simulate glare produced by cameras, the aperture can be a diaphragm and an optionally attached camera filter instead. This can produce the star-shaped glare produced by some cameras.

2.5 Blob Detection

Lindeberg [Lin98] proposed a technique to detect blobs using the scale-space

$$S(\mathbf{x}, \sigma) = g(\mathbf{x}, \sigma) * L(\mathbf{x}) \tag{2.15}$$

of a luminance-image L. The three-dimensional scale-space is defined by a convolution of the image with a Gaussian function

$$g(\mathbf{x}, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\langle \mathbf{x}, \mathbf{x} \rangle}{2\sigma^2}\right) \tag{2.16}$$

with standard deviation σ. To find blobs, the normalized Laplacian response

$$\hat{\Delta} S(\mathbf{x}, \sigma) = \sigma^2 \Delta S(\mathbf{x}, \sigma) \tag{2.17}$$

of the scale-space can be used. The normalization by σ^2 has to be done to counteract the decrease of the magnitude of the Laplacian caused by the smoothing of the Gaussian function. This way, the normalized Laplacian responses of two different scales are comparable.

A blob is then located at a local extremum with respect to x, y and σ. Such an extremum of $\hat{\Delta} S$ found at position $(\mathbf{x}_j, \sigma_k)$ corresponds to a blob at position $\mathbf{x}_j$ in the image with a radius proportional to σ_k.

3 Evaluation of Tone Mapping Operators on Density Maps

Several tone mapping operators were applied to different kinds of images using the Luminance HDR software [ACK+17] to evaluate the effectiveness of the operators. The software supports several simple tone mapping operators (see Equations (2.11), (2.12) and (2.14)) and other operators [Ash02; DD02; Dra+03; FLW02; Fer+11; MDK08; MMS06; Mai+11; Pat+00; RD05; Rei+02]. The default parameters of the software were used for all tone mapping operators. The results can be seen in Figures 3.1 to 3.3 for "Isabel" data (scatter plots), "power plant" data[1] (parallel coordinates), and "flight" data[2] (trajectories). All non-zero regions should be visible using a good tone mapping operator. Many details should be visible as well. It should also introduce as few artifacts as possible, and preserve colors well. These properties are important for visualizing data, because as much as possible should be seen, while no new structures should be added.

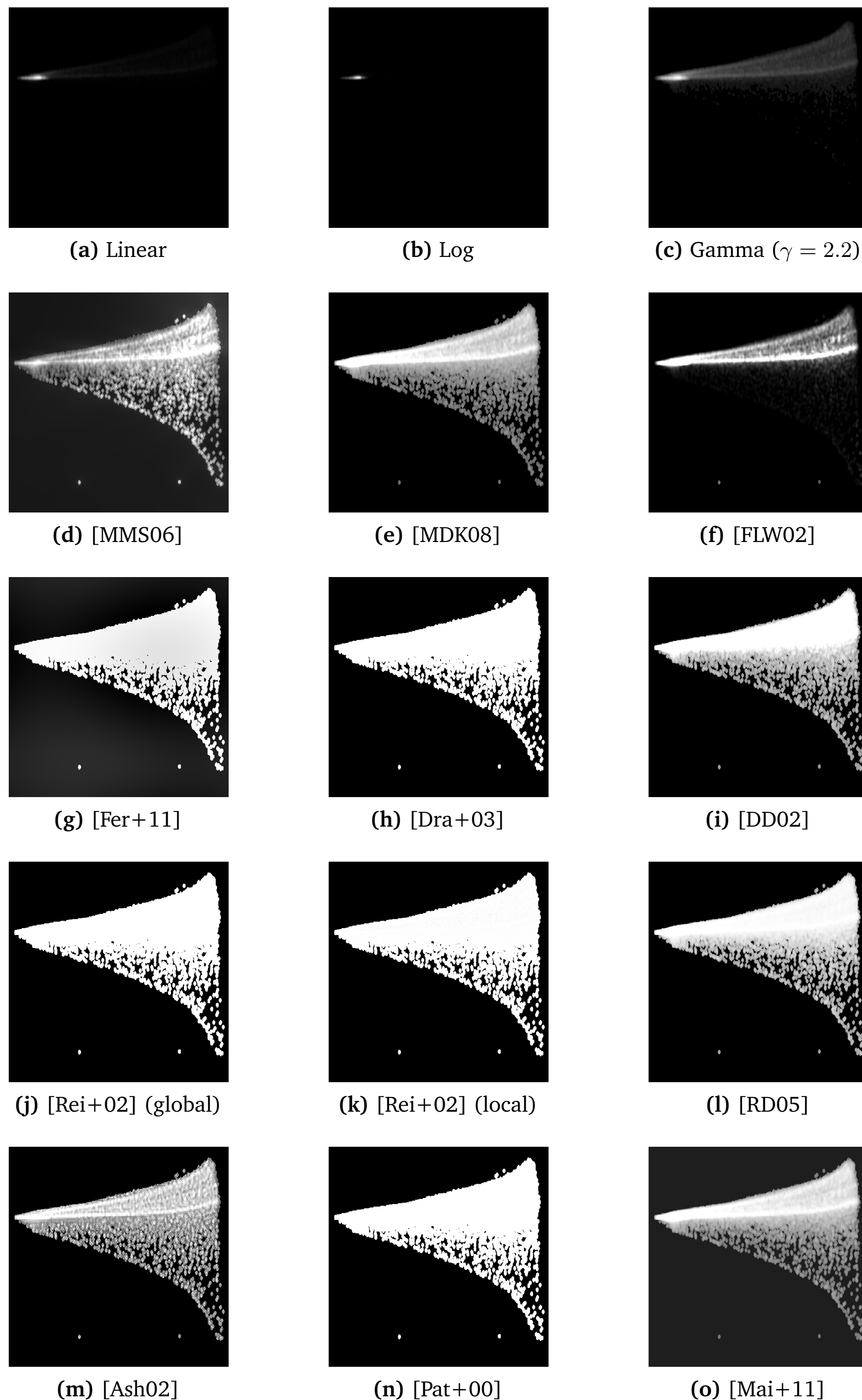
Using the simple operators, only the brightest pixels can be seen well, and the overall results are relatively dark. This is especially true for the linear and logarithmic mappings. The operator of Fattal, Lischinski, and Werman [FLW02] produces similarly dark results. The operators based on the works of Ferradans et al. [Fer+11], Drago et al. [Dra+03], Reinhard et al. [Rei+02], Reinhard and Devlin [RD05], and Pattanaik et al. [Pat+00] produce mostly uniform regions, where no details can be seen. The operator of Durand and Dorsey [DD02] produces similar uniform regions in Figures 3.1 and 3.3. The result of the operator of Ashikhmin [Ash02] seem to introduce over-sharpening artifacts.

This leaves the operators of Mantiuk, Myszkowski, and Seidel [MMS06], Mantiuk, Daly, and Kerofsky [MDK08], and Mai et al. [Mai+11], which all produce good-looking images. Some low-intensity lines can hardly be seen using the operator of Mantiuk, Daly, and Kerofsky [MDK08] for parallel coordinates. The same is true for the operator of Mai et al. [Mai+11]. In the results of the operator of Mantiuk, Myszkowski, and Seidel [MMS06] many details can be seen, even in regions where the other operators fail to do so (for example, in Figure 3.2), but the results look less visually appealing. This operator also brightens up some black regions near brighter regions.

Because the TMO of Mantiuk, Myszkowski, and Seidel [MMS06] is based on human perception and no details are lost, this operator seems to be most suitable to visualize HDR

[1]Combined Cycle Power Plant data from the UCI Machine Learning Repository, see http://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant.

[2]Airline and route data from OpenFlights, see https://openflights.org/data.html.



(a) Linear **(b)** Log **(c)** Gamma ($\gamma = 2.2$)

(d) [MMS06] **(e)** [MDK08] **(f)** [FLW02]

(g) [Fer+11] **(h)** [Dra+03] **(i)** [DD02]

(j) [Rei+02] (global) **(k)** [Rei+02] (local) **(l)** [RD05]

(m) [Ash02] **(n)** [Pat+00] **(o)** [Mai+11]

Figure 3.1: Various tone mapping operators applied to scatter plots. Linear, Log, and Gamma correspond to Equation (2.11), Equation (2.12), and Equation (2.14), respectively.

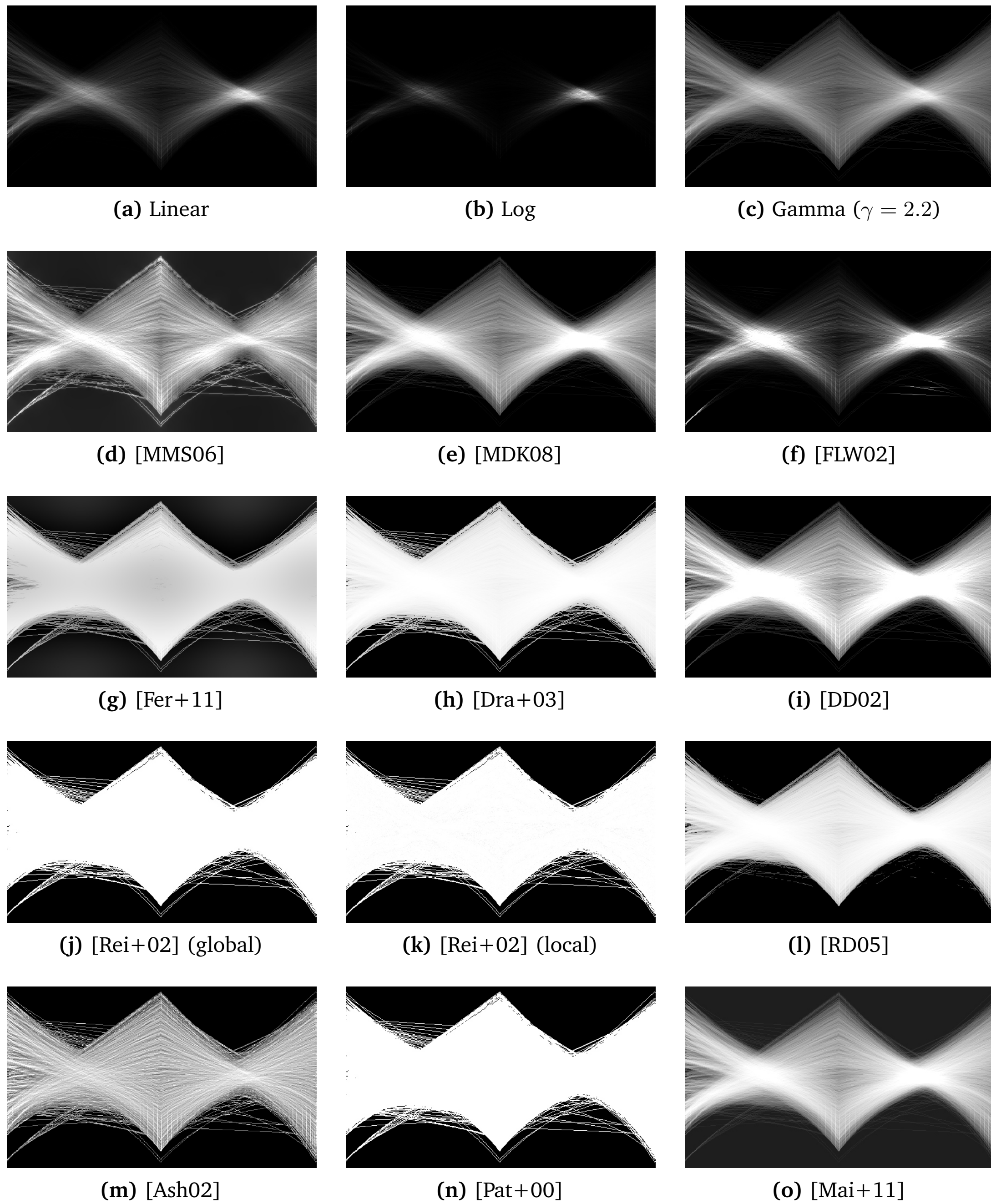
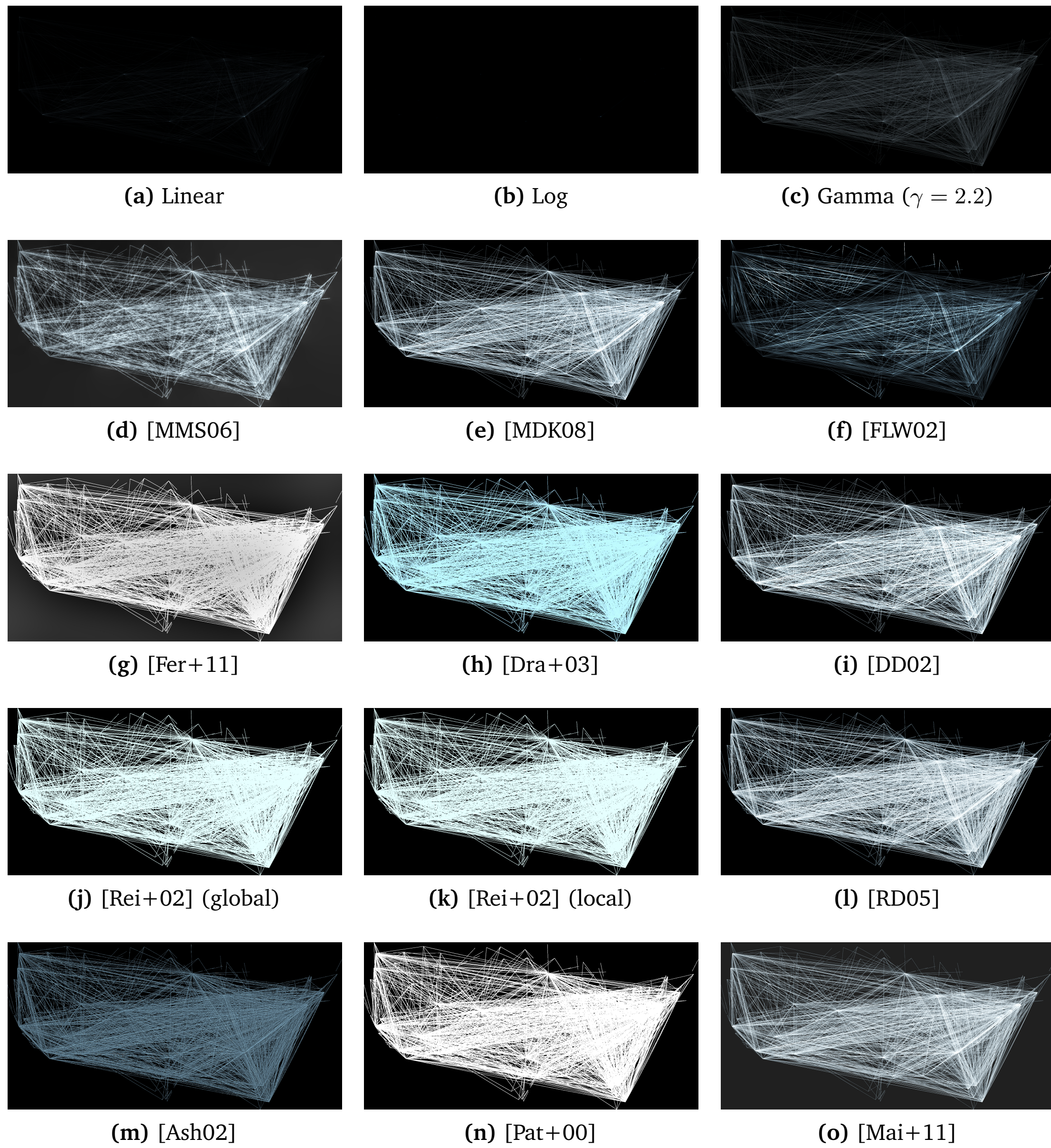


Figure 3.2: Various tone mapping operators applied to parallel coordinates. Linear, Log, and Gamma correspond to Equation (2.11), Equation (2.12), and Equation (2.14), respectively.



(a) Linear **(b)** Log **(c)** Gamma ($\gamma = 2.2$)

(d) [MMS06] **(e)** [MDK08] **(f)** [FLW02]

(g) [Fer+11] **(h)** [Dra+03] **(i)** [DD02]

(j) [Rei+02] (global) **(k)** [Rei+02] (local) **(l)** [RD05]

(m) [Ash02] **(n)** [Pat+00] **(o)** [Mai+11]

Figure 3.3: Various tone mapping operators applied to trajectories. Linear, Log, and Gamma correspond to Equation (2.11), Equation (2.12), and Equation (2.14), respectively.

data. The other operators have some downsides that make them not suitable for this task in some situations. The brightest spots can no longer be easily detected by just looking at the tone mapped image, but this is compensated by detecting these in the original image and applying glare to the spots.

4 Perceptual High-Dynamic-Range Density Map Visualization Model

This approach combines tone mapping and glare simulation to visualize density maps. The techniques are carefully chosen from existing computer graphics methods and extended to the context of visualization. The tone mapping helps a viewer to see details and contrast, while the glare simulation highlights spots with high density. To automate this process, high-density spots are detected automatically using a bright pixel detector. Regions that set themselves apart from other regions can be detected using blob detection. These regions can be used in combination with a brightness threshold to get the bright pixels.

As the tone mapping modifies the contrast of the image and glare simulation adds new structures to an image, the bright pixel detection should be done on the input image. We denote $\boldsymbol{t}(\cdot)$ as the tone mapping operator, $\boldsymbol{g}(\cdot)$ as the glaring operator that uses the bright pixel positions of the original image, and $\odot$ as the blending operator. This leaves two more possibilities to combine tone mapping and glare simulation:

1. glaring followed by tone mapping (see Figure 4.1a):

$$\bar{I}(\mathbf{x}) = \boldsymbol{t}(I(\mathbf{x}) \odot \boldsymbol{g}(I(\mathbf{x}))), \tag{4.1}$$

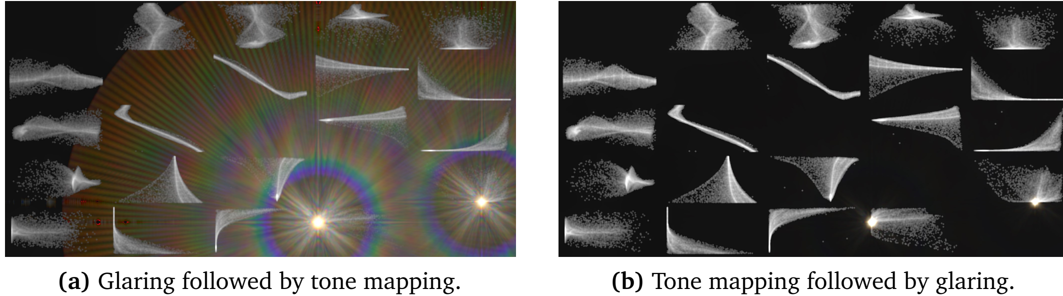
2. tone mapping followed by glaring (see Figure 4.1b):

$$\bar{I}(\mathbf{x}) = \boldsymbol{t}(I(\mathbf{x})) \odot \boldsymbol{g}(\boldsymbol{t}(I(\mathbf{x}))). \tag{4.2}$$

The first approach applies the tone mapping operator to the glared image. This leads to an enhancement of contrast of the glare. The *lenticular halo* and the *ciliary corona* get enhanced and are easily visible. Even the parts that should have very low brightness and should be invisible can be seen. This introduces new structures to the image, which are too pronounced and are not suitable to highlight only the intended bright pixels. The second approach does not have these issues. It is identical to the glaring of LDR images, but additional information from the bright pixel detection in the HDR image can be used.

4.1 Pipeline

The three main components of this approach are tone mapping, bright pixel detection, and glare simulation. Bright pixel detection is further divided into blob detection and



(a) Glaring followed by tone mapping. **(b)** Tone mapping followed by glaring.

Figure 4.1: Comparison of the order of tone mapping and glaring using "Isabel" data.

masking, while glare simulation is divided in aperture generation, diffraction approximation, convolution, and blending.

The pipeline of this approach can be seen in Figure 4.2. Steps are shown as black boxes connected to the output of the step (below the black box) and one or more inputs of the step (above the black box or at the same level). The results of steps are shown in white boxes together with an example image. The image I is the input of the pipeline and the tone mapped and glared image $\bar{I}$ is the output. The steps tone mapping, blob detection, masking, and blending can each be modified with one parameter. The details of the main components of the approach and the individual steps are described in Sections 4.2 to 4.4.

4.2 Tone Mapping

The perceptual framework of Mantiuk, Myszkowski, and Seidel [MMS06] is used for tone mapping because it is based on the human perception of contrast in images. The contrast-based approach ensures that details and structures can still be seen in the tone mapped image. In early version of the model, the tone mapping operator of Reinhard et al. [Rei+02], which is constructed to make images look realistic, was used. A comparison of these two tone mapping operators can be seen in Figure 4.3.

Because of the perceptual basis and better results with the test data, the perceptual framework of Mantiuk, Myszkowski, and Seidel is used in this approach. The proposed contrast-based framework incorporates the following steps:

Step 1. Calculate the luminance L from an image I.

Step 2. Calculate the contrast G for neighboring luminance pixels.

Step 3. Calculate the response R from the contrast using a transducer function.

Step 4. Calculate the modified response $\hat{R}$.

Step 5. Calculate the modified contrast $\hat{G}$ from the modified response using the inverse transducer function.

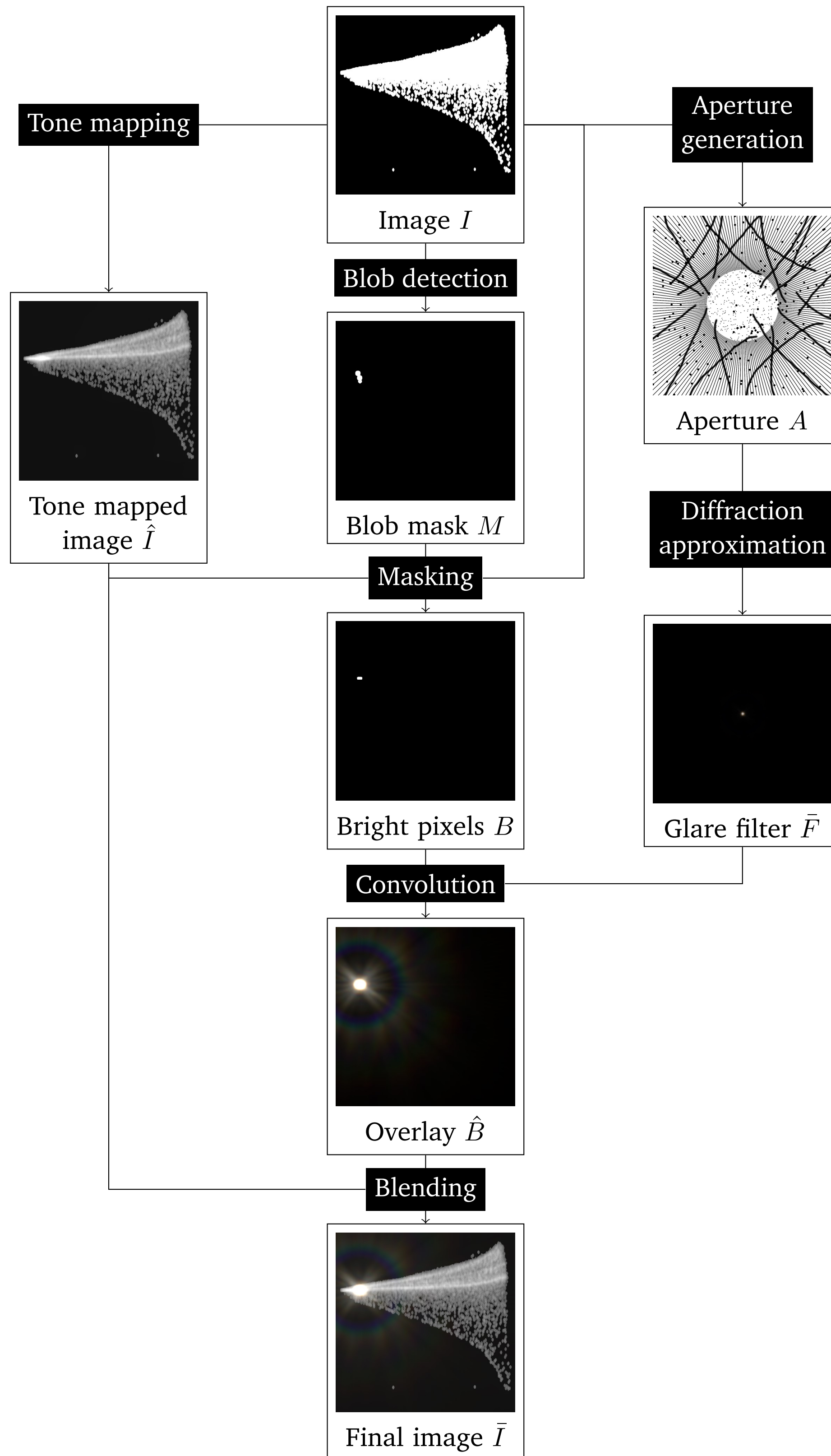
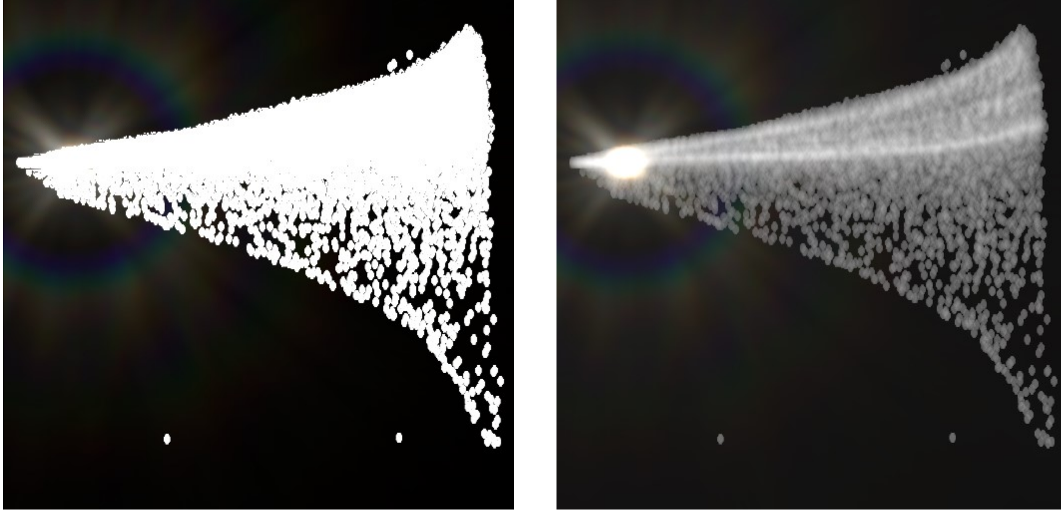


Figure 4.2: The pipeline of the outlined approach.



(a) Using the tone mapping operator of Reinhard et al. [Rei+02].

(b) Using the tone mapping operator of Mantiuk, Myszkowski, and Seidel [MMS06].

Figure 4.3: Comparison of the results of the approach using different tone mapping operators.

Step 6. Calculate the modified luminance $\hat{L}$ by solving an optimization problem on the modified contrast.

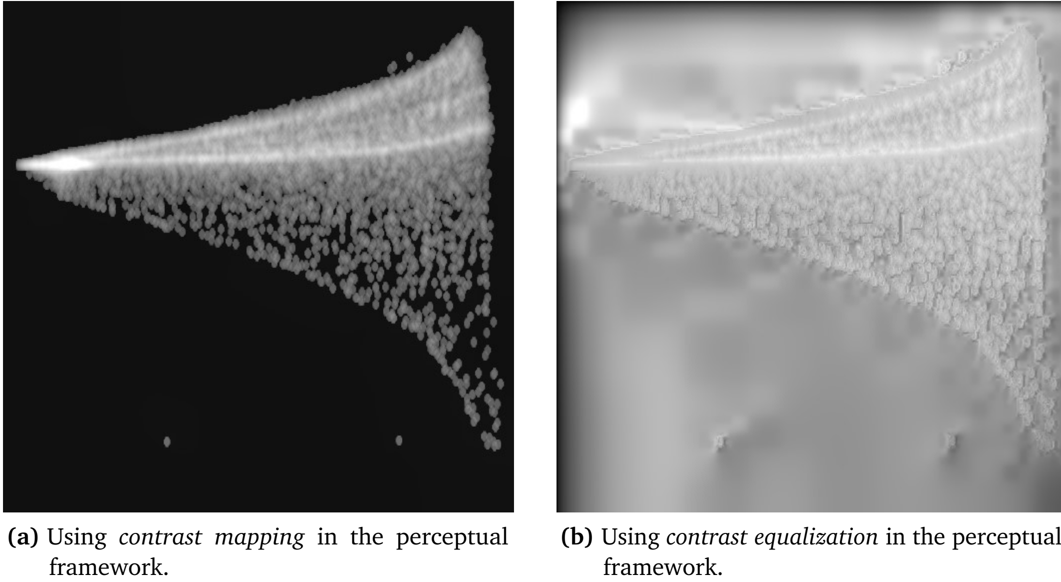
Step 7. Calculate the modified image $\hat{I}$ using the modified luminance and the original image.

From Step 1 to Step 6 Gaussian pyramids are used to represent L, G, R and the modified versions of them. This is because the human visual system can perceive contrast for small regions close to each other or bigger regions farther apart. The Gaussian pyramid for the luminance is created by successively dividing the image width and height by a factor of two until one dimension would be smaller than three pixels. Except for the optimization problem in Step 6 and the construction of the pyramid in Step 1, operations are done independently on each level of the pyramid.

The contrast in Step 2 is defined as the ratio of the luminance of neighboring pixels. As the contrast is in the $\log_{10}$-domain, the ratio can be reformulated as the difference of the logarithm of neighboring pixels. The contrast is calculated for neighboring pixels $\mathbf{u}$ and $\mathbf{v}$ as

$$\begin{aligned} G^k(\mathbf{u},\mathbf{v}) &= \log_{10} \frac{L^k(\mathbf{u})}{L^k(\mathbf{v})} \\ &= \log_{10} L^k(\mathbf{u}) - \log_{10} L^k(\mathbf{v}) \end{aligned} \tag{4.3}$$

where k denotes the level of the Gaussian pyramid. The neighborhood of a pixel depends on the field of application of the framework. It can range from one pixel in x- and y-direction to the whole image domain. In this approach, contrast is computed for one neighbor in



(a) Using *contrast mapping* in the perceptual framework.

(b) Using *contrast equalization* in the perceptual framework.

Figure 4.4: Comparison of two proposed techniques to modify the response values in the perceptual framework of Mantiuk, Myszkowski, and Seidel [MMS06].

x-direction and one neighbor in y-direction. The contrast G_x^k is the contrast of a pixel and the next pixel in x-direction (if there is one). Otherwise, it is zero. G_y^k is defined analogously for the y-direction.

The response of the human visual system in Step 3 is calculated for each contrast value that was computed in the last step. The transducer function is constructed in a way that the response should change by one for a change in just-noticeable-difference. In Step 5 the inverse of this transducer function is used.

For Step 4 multiple ways to modify the response were proposed, like *contrast mapping* and *contrast equalization*. To reduce contrast with *contrast mapping*, the response is multiplied by a constant between zero and one. *Contrast equalization* refers to equalizing the histogram of the contrast response. Mantiuk, Myszkowski, and Seidel claim that *contrast equalization* produces very sharp images and can enhance the visibility of small details, and *contrast mapping* produces sharp images. For some of the tested density maps, the optimization problem solver for the *contrast equalization* method does not converge and produces artifacts as seen in Figure 4.4. *Contrast mapping* does not have these disadvantages, produces satisfying results and is computationally less demanding. Therefore, *contrast mapping* is used in this approach.

In Step 6 an optimization problem has to be solved because the modified contrast does not necessarily correspond to an image. For the proposed optimization problem, an objective function

$$\sum_{k=0}^{N_g-1} \sum_{\mathbf{u}\in\mathbb{D}} \sum_{\mathbf{v}\in\phi(\mathbf{u})} p(\hat{G}^k(\mathbf{u},\mathbf{v})) \cdot (\hat{L}^k(\mathbf{u}) - \hat{L}^k(\mathbf{v}) - \hat{G}^k(\mathbf{u},\mathbf{v}))^2 \tag{4.4}$$

has to be minimized with regard to the modified luminance $\hat{L}$ of pixels at the finest level of the Gaussian pyramid, where N_g is the number of levels of the Gaussian pyramid and $\phi(\cdot)$ is the neighborhood of a pixel. The same neighborhood as in Step 2 has to be used. In the objective function, the factors $p(\cdot)$ are used to penalize a contrast mismatch. A mismatch is less penalized for contrast values, to which the human visual system is less sensitive to. These factors and the size of the pixel neighborhood can be changed to fit the field of application. An efficient solution to the optimization problem was proposed using the biconjugate gradient method. Note that $\hat{L}$ is in the logarithmic domain in this step, as it will be used like that in the next step.

With the proposed color reconstruction of Step 7, the resulting color can be linearly mapped to the colors of a perceptually linearized display. As the tone mapped image has to be glared after the tone mapping, the part that corresponds to a *gamma correction* is omitted at this point. The resulting tone mapped image $\hat{I}$ is then

$$\hat{I}(\mathbf{x}) = \frac{I(\mathbf{x})}{L(\mathbf{x})} \cdot 10^{\left(\frac{\hat{L}(\mathbf{x})-l_{\text{min}}}{l_{\text{max}}-l_{\text{min}}}\right)\cdot r - r} \tag{4.5}$$

where l_{min} is the 0.1-percentile of $\hat{L}$, l_{max} is the 99.9-percentile of $\hat{L}$, and $r = 2.3$ is the dynamic range of the output device [Man+16].

4.3 Bright Pixel Detection

The blob detection of Lindeberg [Lin98] is used to detect blobs in the original image. The detected blob position and blob size are used to build a blob mask, which is used to mask the tone mapped image. This way, the information of the original image and the color values of the tone mapped image are used. The blob detection and masking are described in the following sections.

4.3.1 Blob Detection

The normalized Laplacian response $\hat{\Delta} S$ can be used to detect blobs in scalar images. The luminance L of the original image I is used for blob detection. For discrete images, the scale-space S has to be discretized as well. The spatial dimensions are discretized like the original image and sampling in the scale dimension should be linear for fine scales and exponential for coarser scales [see Lin98, Footnote 2]. The chosen scales

$$\sigma_j = \sqrt[4]{2^j} \tag{4.6}$$

satisfy this. σ_j behaves almost linearly[1] for small j, while the overall behavior is exponential.

[1]A linear regression for σ_j with $j \in [0, 8] \subset \mathbb{N}$ was found using the MATLAB function `regression`. The normalized mean squared error of this regression is 0.9636 (evaluated with the function `goodnessOfFit`). Therefore, σ_j behaves like a linear function for small j.

Local extrema in the discretized scale-space can be found by comparing a value to its 26 neighbors: eight on the same scale, nine on the next lower scale, and nine on the next higher scale. On boundaries of S less than 26 neighbors exist and have to be compared. To not take the sign of $\hat{\Delta} S$ into account, absolute values are used and the local maximum has to be found [Lin98].

Detecting all local extrema in $\hat{\Delta} S$ leads to blobs, of which many only cause a small response. As proposed by Lindeberg, blobs are thresholded and circles are drawn for blobs above the threshold. Local extrema with a value of $\hat{\Delta} S$ smaller than the threshold T_M are ignored, while the remaining ones are classified as blobs. For a local extremum at position $(\mathbf{x}_j, \sigma_k)$, a solid circle with radius $\sqrt{2} \cdot \sigma_k$ and center $\mathbf{x}_j$ will be drawn on the the blob mask M. The blob mask has a value of one if a pixel is within a blob and zero otherwise.

This radius is chosen, so the zero-crossings of the Laplacian of the Gaussian align with a circle of radius r [Laz11]. The Laplacian of the Gaussian

$$\Delta g(\mathbf{x}, \sigma) = \frac{\langle \mathbf{x}, \mathbf{x} \rangle - 2\sigma^2}{2\pi\sigma^6} \exp\left(-\frac{\langle \mathbf{x}, \mathbf{x} \rangle}{2\sigma^2}\right) \tag{4.7}$$

has zero-crossings when $\langle \mathbf{x}, \mathbf{x} \rangle$ equals $2\sigma^2$. Let the circle (without loss of generality) be centered around the origin, then $\langle \mathbf{x}, \mathbf{x} \rangle$ equals r^2. Then the radius is $\sqrt{2}$ times the scale it was detected on. Other radii could be used as well. For example, sometimes a radius of 3 times the scale are used to cover the 3σ-region of a Gaussian function [Kon+13].

Using this relation between scales and radii of blobs, a maximum scale can be defined. In this approach, the maximum scale is set to the first scale, on which a blob's diameter exceeds $20\,\%$ of the minimum of the image width and height. The smallest scale used is σ_0. The number of scales is then

$$N_\sigma = \arg\min_{j \in \mathbb{N}} (2 \cdot \sigma_j \geq 0.2 \cdot \min\{N_x, N_y\}) + 1, \tag{4.8}$$

which can be used to define the scale-space domain

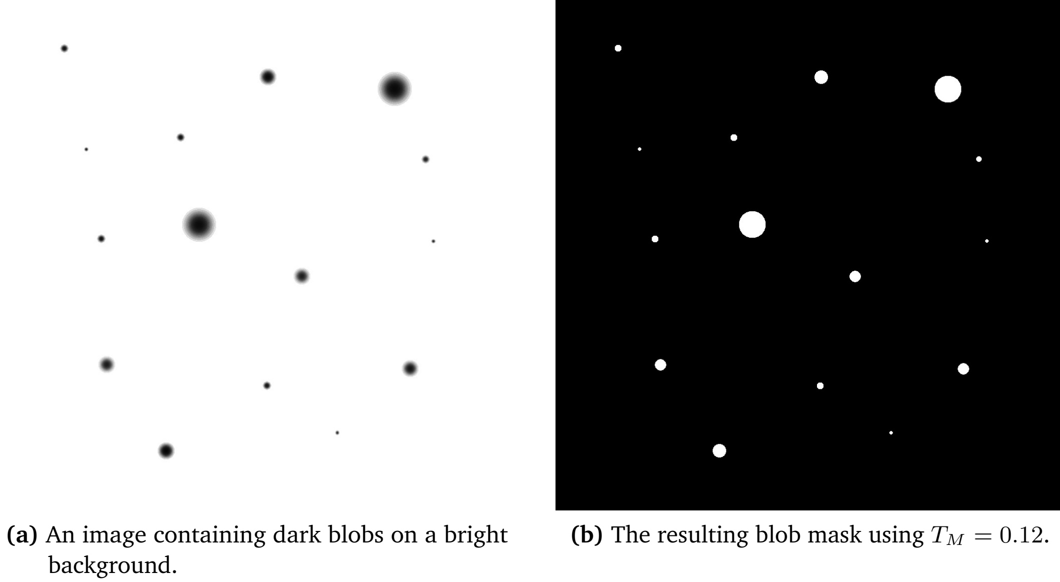
$$\mathbb{S} = \mathbb{D} \times \{\sigma_j \mid \mathbb{N} \ni j < N_\sigma\}\,. \tag{4.9}$$

The blob mask M is defined to be one for all pixels near a local extremum of the scale-space. The circular neighborhoods across all scales are inspected to find extrema and the finds are thresholded:

$$M(\mathbf{x}) = \begin{cases} 1 & \bigvee_{j=0}^{N_\sigma - 1} \bigvee_{\mathbf{u} \in \varphi(\mathbf{x}, \sigma_j)} H(\mathbf{u}, \sigma_j) \geq (L_{\text{max}} - L_{\text{min}}) \cdot T_M \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

with a circular neighborhood

$$\varphi(\mathbf{x}, \sigma_j) = \left\{\mathbf{u} \in \mathbb{D} \mid \sqrt{2} \cdot \sigma_j \geq \|\mathbf{x} - \mathbf{u}\|\right\} \tag{4.11}$$



(a) An image containing dark blobs on a bright background.

(b) The resulting blob mask using $T_M = 0.12$.

Figure 4.5: Example of blob detection.

that is checked for extrema, and the local extremum detection

$$H(\mathbf{u}, \sigma_j) = \begin{cases} |\hat{\Delta} S(\mathbf{u}, \sigma_j)| & \bigwedge_{(\mathbf{v}, \sigma_k) \in \psi(\mathbf{u}, \sigma_j)} |\hat{\Delta} S(\mathbf{u}, \sigma_j)| > |\hat{\Delta} S(\mathbf{v}, \sigma_k)| \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

for the local neighborhood

$$\psi(\mathbf{u}, \sigma_j) = \{(\mathbf{v}, \sigma_k) \in \mathbb{S} \setminus \{(\mathbf{u}, \sigma_j)\} \mid 1 \geq \|\mathbf{u} - \mathbf{v}\|_\infty \wedge 1 \geq |j - k|\}\,. \tag{4.13}$$

The blob threshold T_M gets rescaled by $(L_{\text{max}} - L_{\text{min}})$ to be a parameter that does not depend on the range of the input image. This scaling is chosen to fit the discretized implementation of the Laplacian operator (see Equation (6.1)) and the definition of the scale-space. The smoothing by the convolution with the Gaussian function does not change the range of the image, as it is counteracted by the normalization of the Laplacian operator. The magnitude of the discrete Laplacian is then $|4 \cdot L_{\text{max}} - 4 \cdot L_{\text{min}}|$ in the worst case.

The blob mask detects bright and dark blobs, as the normalized Laplacian response detects regions that are similar in brightness. Therefore, the mask is used in combination with another threshold to mask the tone mapped image.

4.3.2 Masking

This blob mask is then used to identify the location of bright pixels. For the masking, a simple multiplication of the mask with the original image is used. As seen in Figure 4.5, even dark blobs get detected by the blob detection. To only choose the bright blobs, a luminance threshold T_L is used in addition to the blob mask. The value of a pixel in the

bright-pixel image B is either the color of the tone mapped image at this position, if the masked luminance of the original image is greater or equal to the threshold, or black:

$$B(\mathbf{x}) = \begin{cases} \hat{I}(\mathbf{x}) & M(\mathbf{x}) \cdot L(\mathbf{x}) \geq (L_{\max} - L_{\min}) \cdot T_L + L_{\min} \\ (0,0,0) & \text{otherwise} \end{cases} \tag{4.14}$$

The luminance threshold is rescaled so $T_L = 0$ corresponds to the minimum luminance and $T_L = 1$ to the maximum luminance. Only these bright pixels have to be convolved with the glare filter. After that the glared bright pixels are combined with the tone mapped image.

4.4 Glare Simulation

The wave-optics-based glare simulation is used in this approach. The type of glare is chosen to be the one as seen by humans. For this, the aperture of the human eye has to be simulated. The model of Ritschel et al. for the human eye is used, but no dynamics are simulated.

The diffraction for this aperture is then computed to construct the glare filter. The result of the diffraction is considered for different wavelengths of light to produce the colorful *lenticular halo*. The glare filter is applied to the bright pixels with a convolution, and the glared image has to be blended with the tone mapped image. The aperture generation, diffraction approximation, convolution, and blending are described in the following sections.

4.4.1 Aperture Generation

The same parts (lens fibers, different kind of particles, the pupil, and eyelashes) as used by Ritschel et al. are considered in the model of the human, but only simulated once without movement or dynamic changes. Like Ritschel et al., the different parts of the human eye are assumed to be located on the same plane as an approximation. While the pupil diameter changes depending on luminance [Rit+09; Spe+95], a fixed pupil diameter of $9\,\mathrm{mm}$ is used, as the luminance of a density map does not necessarily correspond to physical luminance. The generated aperture can then be used to generate a glare filter using a diffraction approximation.

4.4.2 Diffraction Approximation

The glare filter can be approximated as the diffraction pattern at the retina caused by the aperture A. The diffraction is approximated with the Fresnel diffraction by Ritschel et al. [Rit+09] to get

$$F_\lambda(\mathbf{u}) = \frac{1}{\lambda^2 d^2} \left| \mathcal{F}[A(\mathbf{x}) \cdot E(\mathbf{x})] \left(\frac{\mathbf{u}}{\lambda d}\right) \right|^2 \tag{4.15}$$

with the complex exponential

$$E(\mathbf{x}) = \exp\left(\frac{\pi i}{\lambda d}\langle \mathbf{x}, \mathbf{x}\rangle\right)$$

or with the Fraunhofer diffraction by Kakimoto et al. [Kak+04] to get

$$F_\lambda(\mathbf{u}) = \frac{1}{\lambda^2 d^2}\left|\mathcal{F}[A(\mathbf{x})]\left(\frac{\mathbf{u}}{\lambda d}\right)\right|^2 \tag{4.16}$$

for a single wavelength of light λ and the distance between pupil and retina d. The light is assumed to be homogeneous and to have unit-amplitude. Equations (4.15) and (4.16) only differ by the term E. Note that Equations (4.15) to (4.17) and (6.6) use coordinates where the origin is in the center of the aperture. The Fresnel diffraction is used in this approach, as it is more suitable because the distance between pupil and retina is small and the results look more realistic [Rit+09].

To not recompute the glare filter for each wavelength of light, the spectral glare filter $\bar{F}$ can be approximated by summing up samples from one monochromatic glare filter F_{λ_0}. For n samples for wavelengths between λ_{min} and λ_{max}, the spectral glare is

$$\bar{F}(\mathbf{x}) = \sum_{j=0}^{n-1} \mathcal{S}(\lambda_j) \cdot F_{\lambda_0}\left(\frac{\lambda_0}{\lambda_j} \cdot \mathbf{x}\right) \tag{4.17}$$

with

$$\lambda_j = \lambda_{\text{min}} + j \cdot \frac{\lambda_{\text{max}} - \lambda_{\text{min}}}{n}$$

where λ_0 is the wavelength of the precomputed glare filter F_{λ_0} and $\mathcal{S}$ is the spectrum of the current wavelength in the RGB color space. λ_0 should be approximately $(\lambda_{\text{min}} + \lambda_{\text{max}})/2$ to minimize the error of the approximation [Rit+09].

The colors $\mathcal{S}$ for the wavelengths can be computed by first computing the XYZ values of the CIE 1931 XYZ color space and then converting the values to the RGB color space. The wavelength dependent color matching functions $\bar{x}$, $\bar{y}$, and $\bar{z}$ can be approximated using multiple piecewise continuous Gaussian functions [Wym+13]. The approximated $\bar{x}$ is used to calculate X, $\bar{y}$ to calculate Y, and $\bar{y}$ to calculate Y. The necessary integration is approximated by setting X to the value of $\bar{x}$, Y to $\bar{y}$, and Z to $\bar{z}$. The XYZ values are then converted to linear RGB values [Int15a] with the matrix-vector multiplication

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 3.2406255 & -1.537208 & -0.4986286 \\ -0.9689307 & 1.8757561 & 0.0415175 \\ 0.0557101 & -0.2040211 & 1.0569959 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \end{pmatrix}. \tag{4.18}$$

With any approach (phenomenological or wave-optics-based) the calculated glare filter will be convolved with an image to simulate glare. To not blur the original image, the filter is only convolved with high-intensity pixels.

4.4.3 Convolution

The glare filter $\bar{F}$ has to be convolved with only the bright pixels B the get a glare overlay

$$\hat{B}(\mathbf{x}) = B(\mathbf{x}) * \bar{F}(\mathbf{x}) \tag{4.19}$$

that has to be blended with the tone mapped image. The convolution theorem can be used to implement the convolution efficiently using two Fourier transforms and an inverse Fourier transform:

$$B(\mathbf{x}) * \bar{F}(\mathbf{x}) = \mathcal{F}^{-1}[\mathcal{F}[B(\mathbf{x})]\,(\mathbf{u}) \cdot \mathcal{F}[\bar{F}(\mathbf{x})]\,(\mathbf{u})]\,(\mathbf{x})\,. \tag{4.20}$$

Alternatively, this can be approximated by rendering billboards with the glare filter on-top of the original image.

4.4.4 Blending

The tone mapped image $\hat{I}$ and the glare overlay $\hat{B}$ have to be combined to get the final image $\bar{I}$. This is done with additive blending. To add more flexibility, an additional parameter is available to multiply the glare overlay by a constant factor before blending. This can be used to increase or decrease the intensity of the glare.

These modifications of the glare intensity are used to counteract the fact that only a part of the image gets glared. The image of bright pixels B is black at places, where no bright pixels were detected. As the glare filter is a point spread function, the brightness of the glare depends on the neighborhood of the bright pixels. This means that the glare is weaker if only a few bright pixels are close together, compared to a larger group of bright pixels.

Another reason for the modification of the glare intensity is the used tone mapping operator. Depending on the image, the glare can be barely seen if it is added to an image with high contrast, while it is perfectly visible on uniform black backgrounds. The modification of the glare intensity can help in these cases as well.

The final image is then

$$\bar{I}(\mathbf{x}) = \hat{I}(\mathbf{x}) + P_B \cdot \hat{B}(\mathbf{x}) \tag{4.21}$$

or simply

$$\bar{I}(\mathbf{x}) = \hat{I}(\mathbf{x}) + \hat{B}(\mathbf{x})$$

in the default case with $P_B = 1$.

The blending step concludes the pipeline of the proposed approach. A *gamma correction* has to be applied to the output image $\bar{I}$ before displaying it. An efficient GPU-based implementation is detailed in Chapter 6.

5 User Interaction

The approach requires only a few user interactions which can be done intuitively through a small number of simple sliders. Tone mapping, blob detection, masking, and blending can be influenced using the approach outlined in Chapter 4. The *gamma correction* that is applied before an image is displayed can be changed as well.

Tone mapping has one parameter, the *contrast factor* P_R, that can be changed. The parameter influences the contrast compression, which alters the look of the tone mapped image. The default value of 0.4 produced good results for all images that were tested.

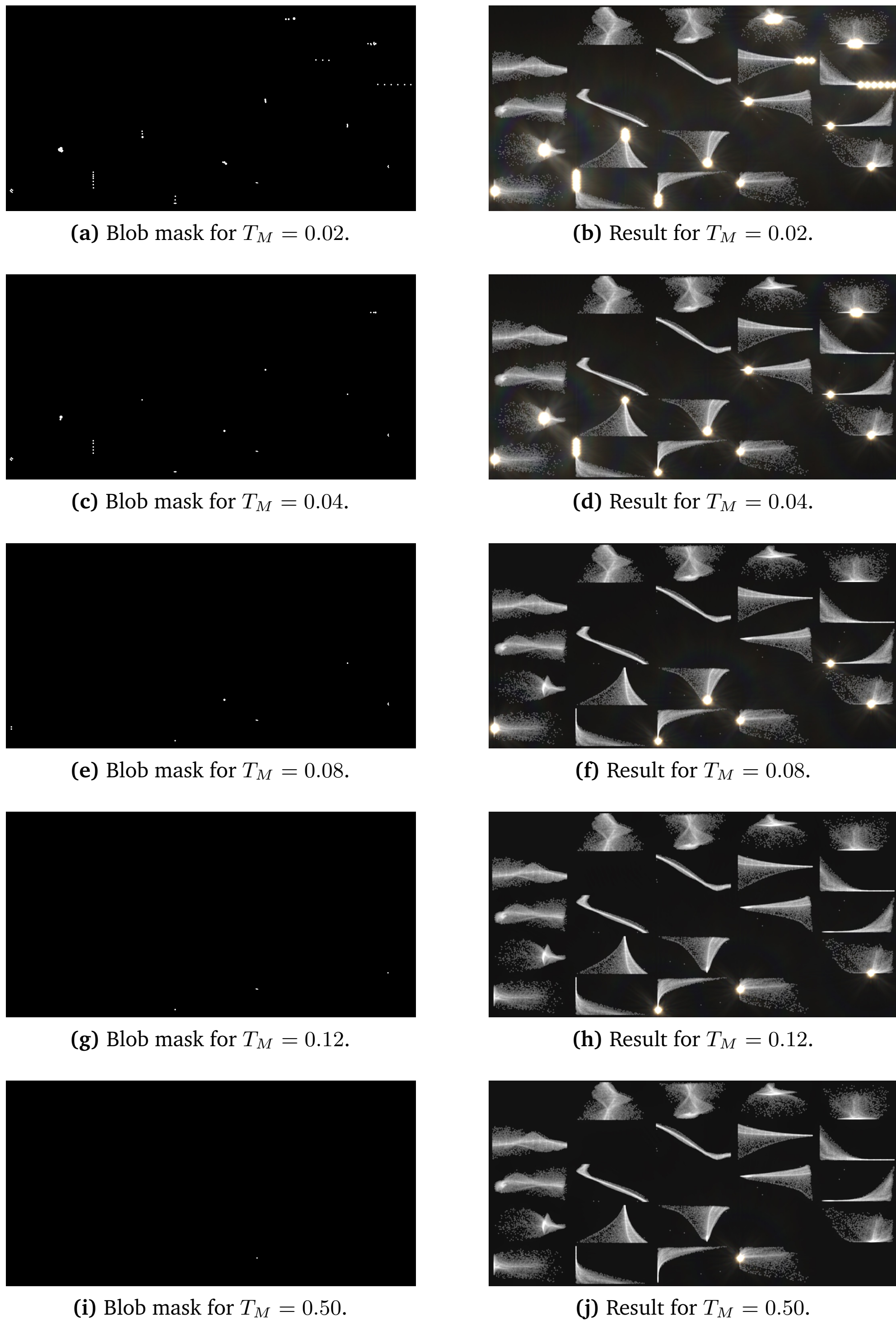
The bright pixel detection has two thresholds that can be changed. The blob threshold T_M causes potential blobs with weak responses to be ignored. Lower values of T_M cause more blobs to be detected, while higher values cause less or no blobs to be detected. Setting T_M to zero causes all pixels of the blob mask to be one. This is equivalent to detecting a blob at every pixel or to performing no blob detection at all, and only using the luminance threshold to detect bright pixels. Examples for varying T_M can be seen in Figure 5.1.

The other threshold is used to classify pixels as bright or dark. The luminance threshold T_L is used in combination with the original image and the blob mask to find bright blobs. Lower values of T_L cause more pixels in the detected blobs to be considered as bright, while higher values cause less or no pixels to be considered as bright. Setting T_L to zero causes all pixels to be considered as bright. This means the whole image gets glared. Examples for varying T_L can be seen in Figure 5.2.

The luminance threshold can be used in a natural way to highlight locations with values above a known threshold. The blob threshold can be set to zero in this case. If no threshold for the values is known, the blob detection can be used to detect potentially interesting regions.

The parameter used in the blending step is used to modify the intensity of the glare. The glare intensity P_B can be used to increase or decrease the glare. This parameter has to be changed only if the glare is too bright, or if the glare can be hardly seen. Setting P_B to zero is equivalent to just performing the tone mapping and no glaring. Examples for varying P_B can be seen in Figure 5.3.

The default values of $T_M = 0.12$ and $T_L = 0.6$ produce good results when applied to the tested scatter plots. For the tested trajectories and parallel coordinates, the parameters have to be changed slightly to produce better results than the default ones. This suggests that a pair of fitting parameters can be found for each class of input data to serve as a good heuristic, so no further adjustments have to be made to get good results.



(a) Blob mask for $T_M = 0.02$.

(b) Result for $T_M = 0.02$.

(c) Blob mask for $T_M = 0.04$.

(d) Result for $T_M = 0.04$.

(e) Blob mask for $T_M = 0.08$.

(f) Result for $T_M = 0.08$.

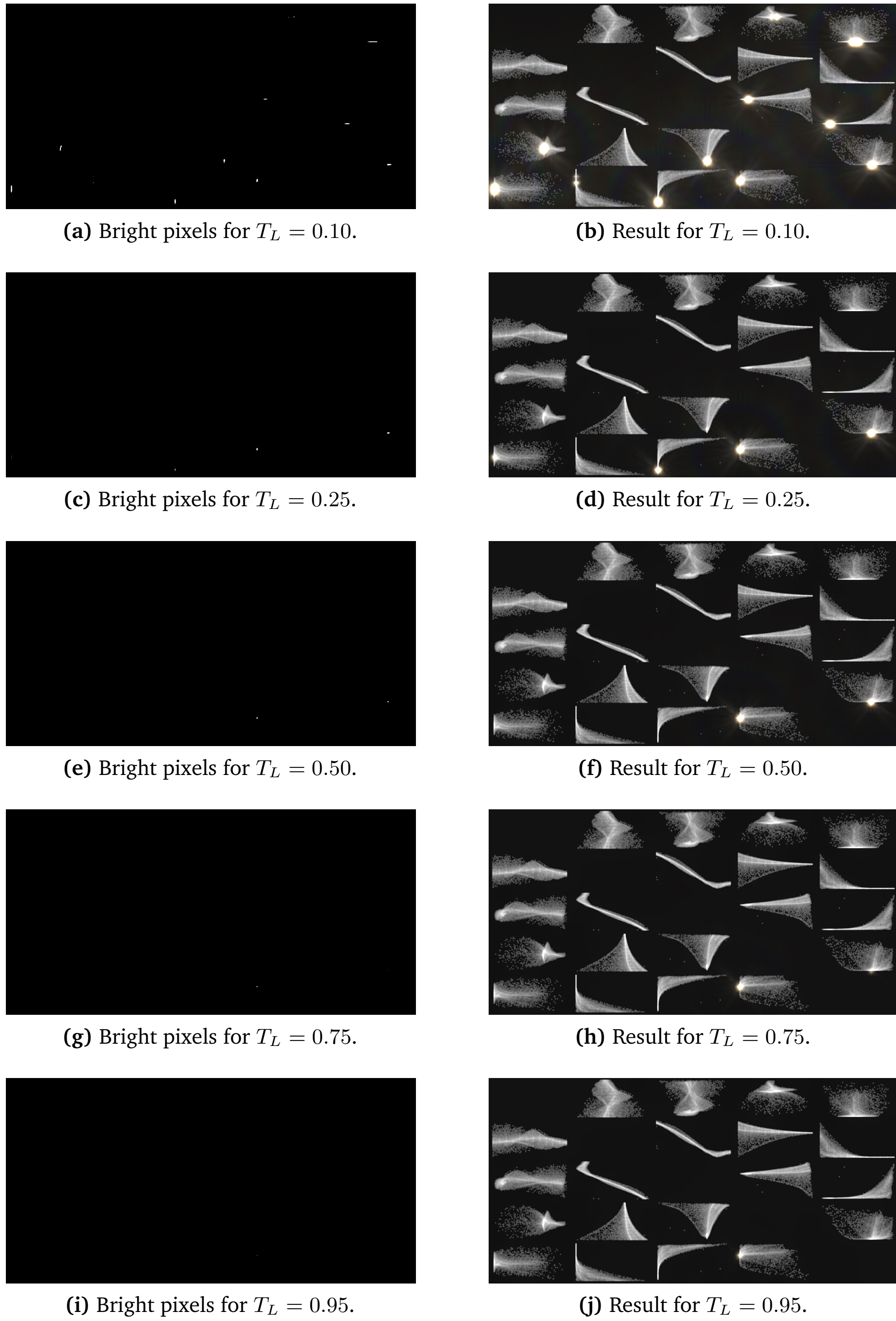
(g) Blob mask for $T_M = 0.12$.

(h) Result for $T_M = 0.12$.

(i) Blob mask for $T_M = 0.50$.

(j) Result for $T_M = 0.50$.

Figure 5.1: Different outputs for varying values of the blob threshold T_M. The luminance threshold T_L is set to 0.01 for all images.



(a) Bright pixels for $T_L = 0.10$. (b) Result for $T_L = 0.10$.

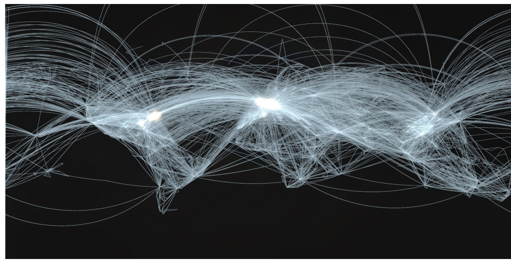
(c) Bright pixels for $T_L = 0.25$. (d) Result for $T_L = 0.25$.

(e) Bright pixels for $T_L = 0.50$. (f) Result for $T_L = 0.50$.

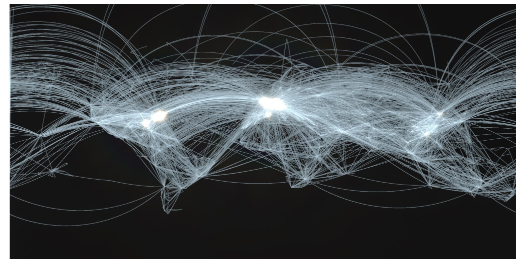
(g) Bright pixels for $T_L = 0.75$. (h) Result for $T_L = 0.75$.

(i) Bright pixels for $T_L = 0.95$. (j) Result for $T_L = 0.95$.

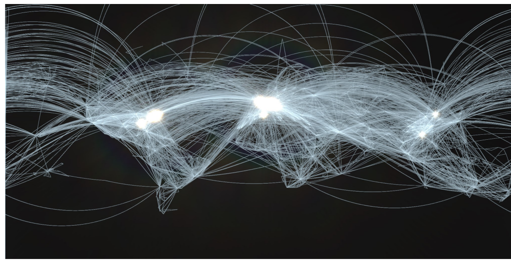
Figure 5.2: Different outputs for varying values of the luminance threshold T_L. Blob detection is turned off by using a blob threshold T_M of zero.



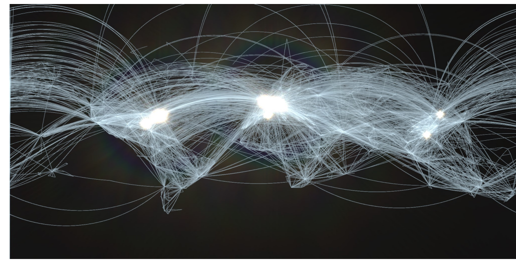
(a) Result for $P_B = 1$.



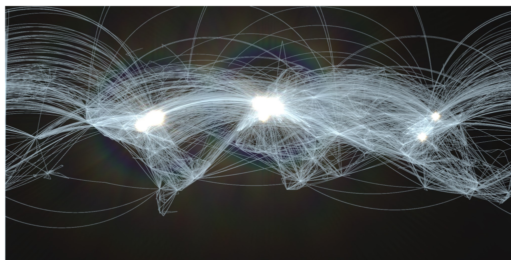
(b) Result for $P_B = 2$.



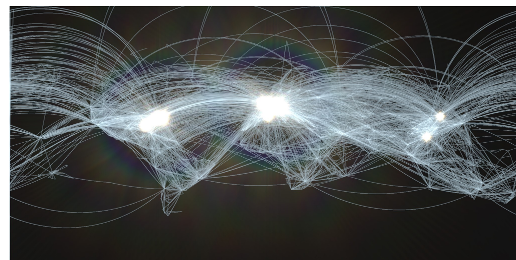
(c) Result for $P_B = 5$.



(d) Result for $P_B = 10$.



(e) Result for $P_B = 15$.



(f) Result for $P_B = 20$.

Figure 5.3: Different outputs for varying values of the glare intensity P_B using “flight” data.

6 GPU-Based Implementation

Many operations can be efficiently implemented using current graphics processing unit (GPU) technology. This is beneficial, as almost all operation necessary to implement the approach outlined in Chapter 4 require point-wise or very local computations on images. These can be realized using compute shaders. General details of the implementation are discussed in Sections 6.1 to 6.3 while the implementation of the parts of the approach is discussed in Sections 6.4 to 6.6.

6.1 Data Structures

The most important data structures for all features are textures. The texture objects of DirectX are used for that purpose. All internal images are stored in textures with $32\,\mathrm{bit}$ floating point values per channel,[1] The used textures have either a single channel or four channels.[2]

To handle these more easily, several wrapper classes for textures are implemented. The classes keep track of the texture format, with, and height. There are different wrappers, depending on whether the texture should be used as a render target, a resource in a shader, or both. The last type is used for almost all algorithms here. It manages pointers to the texture itself, an optimal depth buffer, and different views for the texture. The class holds one pointer to a `RenderTargetView`, a `ShaderResourceView`, and an `UnorderedAccessView` for the texture and a `DepthStencilView` for the depth buffer. The other wrapper classes are similar and hold the pointers to the necessary views.

The helper class `RenderToolkit` is implemented to simplify arithmetic operations on textures as well. It holds pointers to the necessary shaders and buffers to perform the operations. The texture instances that should be used as a target for the operation and the ones that are used as operands are used to call the member function that corresponds to the operation. The `RenderToolkit` sets shaders, resources, and views necessary for the operation; then the operation is executed; and finally everything is un-set.

For some operations (finding the minimum value of a texture, finding the maximum value of a texture, and computing the sum over all pixels of the texture) are implemented using the `RenderToolkit` and pyramids. The texture on the lowest level has a width and height

[1] The only exception is the blob mask M which is saved with a $8\,\mathrm{bit}$ unsigned integer per pixel.

[2] `RenderTargerViews` or `UnorderedAccessView` cannot be created for three-channel $32\,\mathrm{bit}$ floating point textures.

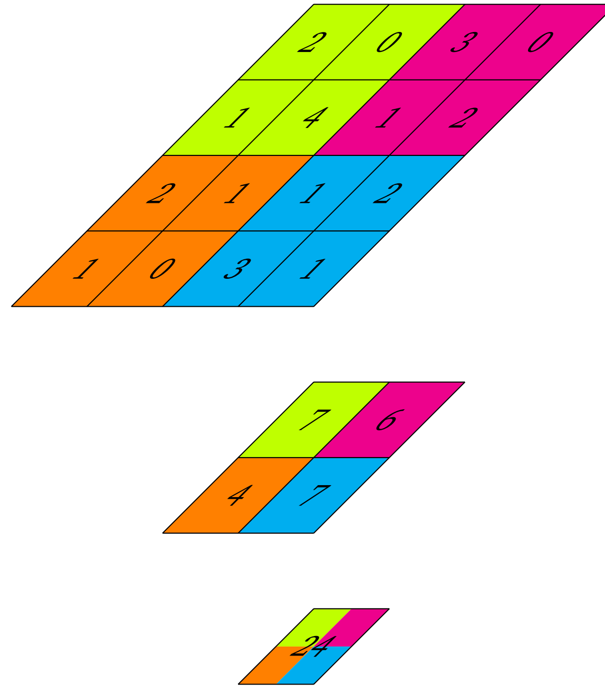


Figure 6.1: Evaluation of operations using pyramids. In this example the sum of all pixels of a scalar 4×4 texture are computed. The top texture is the input texture, the smaller textures are the textures of the pyramid. The bottom texture contains the result of the operation.

of one pixel, and the textures on higher levels double in width and height. The width and height of the texture on the highest level is the smallest power of two which is less than the maximum of width and height of the texture, on which the operation is computed on, because the texture itself is used as well. The pyramids are used to compute the values of these operations by computing the operation on a 2×2 pixel neighborhood and saving the result to the texture with half the width and height until the result is saved at the lowest level. Out-of-bound reads are handled with the respective neutral element of the operation (positive infinity for the minimum, negative infinity for the maximum, and zero for the sum). An example can be seen in Figure 6.1. Using pyramids, these operations can be performed on the GPU and only a single pixel has to be downloaded to the central processing unit (CPU) to get the result. A naive implementation would download the whole texture and loop once over the whole data to perform the operation.

An existing compute shader implementation [S.15] of the fast Fourier transform is used to compute the Fourier transforms. The implementation features two techniques to compute the transform and two ways of computing the exponentials. The first technique uses two pairs of textures (two for real data and two for imaginary data) and writes data in a ping-pong way to the textures to perform logarithmically many passes. The second one uses shared local memory to perform the passes while synchronizing all threads in a thread group after each pass. The second technique requires fewer texture reads and writes. The two ways of computing the exponentials are saving precomputed values to textures and

computing them in the shaders using the `sincos` function. As with the second technique of performing the transform, computing the exponentials in shaders requires fewer reads and writes.

The second technique and the exponentials computed in shaders perform better if memory bandwidth is a bottleneck, but the second technique has some issues as well. The available local memory is limited, so this technique cannot be used for textures of large dimensions. Additionally, the shaders have to be recompiled if the dimensions of the textures change, which takes more time than using precompiled shaders.

The first technique and exponentials computed in shaders are used in this implementation. This technique supports textures of any size and shaders are not compiled at runtime. The given implementation in the `CFFT` class was changed to use precompiled shaders and smart pointers. Support for complex-valued inputs and member functions to expose `RenderTargetViews` and `UnorderedAccessViews` of the internal textures were added as well.

6.2 Image Files

The DirectXTex library [Mic17] is used to read image files and to save textures to image files. The Radiance RGBE format is supported for HDR images,[3] but the OpenEXR format can be added as well.[4] Other file formats are also supported and can be used as input formats or to save LDR images to files.

6.3 Implementation of the Pipeline

The pipeline of Section 4.1 is implemented by performing the following steps in order: 1. preparation, 2. tone mapping, 3. blob detection, 4. masking, 5. aperture generation, 6. diffraction approximation, 7. convolution, and 8. blending. The intermediate images (tone mapped image, blob mask, bright pixels, aperture, glare filter, and glare overlay) of the pipeline are saved in textures to speed-up computations when parameters are changed. As a result, all images of the pipeline (input image, intermediate images, and output image) can be displayed or saved. The implementation does not include a typical rendering loop but updates the necessary textures when an update with a set of parameters is requested.

The preparation step is done once for an input image. This step includes computing the minimum and maximum luminance and the creation of the intermediate textures of the right sizes. The implementation details of the other steps are described in the following sections.

[3] see Mic17, https://github.com/Microsoft/DirectXTex/wiki/HDR-I-O-Functions.
[4] see Mic17, https://github.com/Microsoft/DirectXTex/wiki/Adding-OpenEXR.

6.4 Tone Mapping

An existing implementation of the tone mapping operator of Mantiuk, Myszkowski, and Seidel [MMS06] from pfstools [Man+16] is used as a baseline for the tone mapping. The existing C++ code is mostly transformed to DirectX compute shaders.

The Gaussian pyramids are implemented as `std::vectors` of one or two textures. Contrast and response values are saved in two textures per level of the pyramid as they are computed for x- and y-direction. As all calculations from Step 2 to Step 5 are done component-wise, only one Gaussian pyramid can be used for the contrast, the response and the modified versions. The values are read from a texture, calculations are done on them, and the result is written back to the texture. The luminance to compute the contrast, as well as a helper variable used to solve the optimization problem, are stored as Gaussian pyramids with one texture on each level. These are implemented as `std::vectors` of textures.

The functions of pfstools operate on arrays of floating point values. These can be easily transferred to compute shaders that operate on textures. The functions operate component-wise on the arrays or inspect only a few neighboring values.

First, the luminance is computed and the logarithm of it. A Gaussian pyramid is calculated for this logarithmic luminance. After that, contrast is calculated for the x- and y-direction on all levels of the pyramid. This covers Step 1 and Step 2 of the tone mapping.

In Step 3 to Step 5, the response of the human visual system is calculated for both contrasts on all levels using a lookup table for the transducer function. Then the parameter P_R is multiplied with each response value to get the modified response. The modified contrast is calculated for each modified response using a lookup table for the inverse transducer function.

The optimization problem of Step 6 is implemented using mostly arithmetic functions and an efficient way of performing a matrix-vector multiplication [see MMS06, Section 7]. Only a few scalar values have to be moved from the GPU to the CPU while solving this problem. These are the result of the scalar product of two vectors, which is computed by multiplying two textures and computing the sum of all pixel of the result.

The final color reconstruction on Step 7 requires percentiles to be computed. The tone mapped luminance has to be sorted to achieve this. This is done on the CPU after downloading the texture using the `concurrency::parallel_buffered_sort` function.

6.5 Bright Pixel Detection

The implementation of the bright pixel detection discussed in Section 4.3 is covered in the following sections. The scale-space blob detection can be efficiently implemented using compute shaders and a few textures. The masking can be easily implemented in a single shader as well.

6.5.1 Blob Detection

The scale-space for the luminance L of an image I is computed to find bright blobs in the original image. The normalized Laplacian response $\hat{\Delta}S$ of Equation (2.17) is implemented with a discrete Gaussian filter followed by a discrete Laplacian operator.

As a two-dimensional Gaussian filter can be separated, the filter is implemented as a one-dimensional Gaussian filter in x-direction followed by another one-dimensional Gaussian filter in y-direction. A one-dimensional Gaussian filter is implemented in a compute shader as a weighted sum of the neighborhood of a pixel. The weights are exponential of the Gaussian function (see Equation (2.16)) of the distance vector to the current pixel. The sum is then normalized by the sum of the weights. This is equivalent to computing a weighted sum with weights that sum up to one. The considered neighborhood is given by $2 \cdot \lceil 3 \cdot \sigma \rceil + 1$ pixels centered at the current pixel. For pixels that are outside the texture, no weight is calculated.

The Laplacian operator

$$\Delta f(x, y) = \partial_{xx} f(x, y) + \partial_{yy} f(x, y) \tag{6.1}$$

can be discretized by discretizing and approximating the second order partial derivatives

$$\partial_{xx} f(x, y) \approx f(x-1, y) - 2 \cdot f(x, y) + f(x+1, y), \tag{6.2}$$

$$\partial_{yy} f(x, y) \approx f(x, y-1) - 2 \cdot f(x, y) + f(x, y+1). \tag{6.3}$$

Both are approximations with consistency order 2. A partial derivative is set to zero if any pixel that has to be sampled is not inside the texture.

To get the normalized Laplacian response, the normalization is done after the Laplacian operator by multiplying the factor σ^2 with the result. After that, the absolute values are computed, as extrema are detected by inspecting only absolute values.

Extrema are detected for each considered scale. The absolute normalized Laplacian response is saved to textures for the current, previous, and next scale. If there is no previous scale, the texture for the previous scale is the same as the one for the next scale. If there is no next scale, the texture for the next scale is the same as the one for the previous scale. Only at most three textures for the $\hat{\Delta}S$ have to be kept in memory and only one has to be calculated per iteration.[5] The other textures can be used from previous iterations.

A compute shader inspects the neighborhood with respect to x, y, and σ for each pixel on the current scale. It outputs the value of the current pixel at the current scale if it is strictly greater than the values for all other pixels in the neighborhood. Pixels that are outside of one texture require no special treatment, as an out-of-bounds read returns zero and only absolute values are compared.

[5] In the first iteration, two textures have to be calculated, while none has to be calculated in the last iteration.

This output is used to draw circles on the blob mask M. If the value of a current pixel is greater or equal to the blob threshold T_M, a circle is drawn around the current pixel with a radius of $\sqrt{2}$ times the current scale. This is done in a compute shader by looping over all pixels that are inside this circular neighborhood and setting them to one.

6.5.2 Masking

The blob mask M is then used in combination with a brightness threshold T_L to get the bright pixels, for which the glare is generated. A compute shader sets a pixel as a bright pixel in B to the value of the tone mapped image $\hat{I}$ if the luminance of the masked original image I at the same position is greater or equal to T_L. Otherwise, the pixel stays black. Glare is only generated for the masked image.

6.6 Glare Simulation

The glare simulation builds upon the implementation by Frisvad [Fri09] of the paper of Ritschel et al. [Rit+09]. The existing OpenGL code was mostly converted to DirectX compute shaders to better work with the rest of the new implementation.

The final glare filter should be twice the size of the image [Spe+95] to fully cover the image. A width and height of

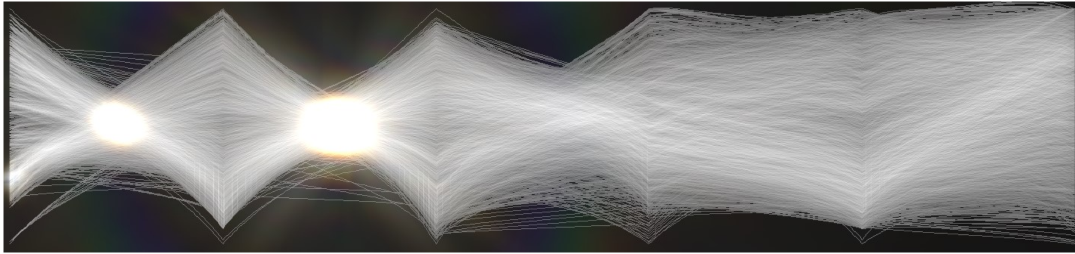
$$N_F = 2 \cdot \min\{N_x, N_y\} \tag{6.4}$$

is chosen for the glare. The maximum of width and height would cover the whole image, no matter what aspect ratio the input has, but the *lenticular halo* could be mostly outside the image and the glare would lose the wanted locality to emphasize high-intensity regions. An example for this can be seen in Figure 6.2.

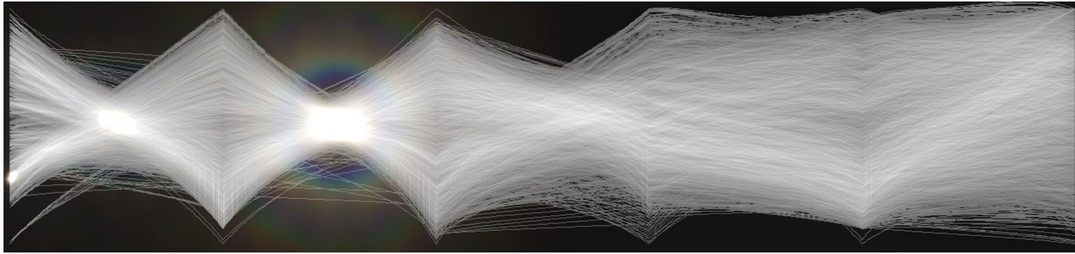
As the glare filter is computed using a fast Fourier transform, it has to be computed on a texture with width and height that are a power of two, leading to an aperture of width and height of

$$N_A = 2^{\lceil \log_2 N_F \rceil}. \tag{6.5}$$

Primitives are drawn on the aperture using supersampling, leading to a size of $2 \cdot N_A$ for drawing them. The aperture is downsampled from $2 \cdot N_A$ to N_A after drawing the primitives and the glare filter is computed using this aperture size. The glare filter is then downsampled to a size of N_F before the final convolution.



(a) Using twice the maximum of width and height as size for the glare filter.



(b) Using twice the minimum of width and height as size for the glare filter.

Figure 6.2: Comparison of the results of the approach using different glare sizes using "power plant" data.

6.6.1 Aperture Generation

The aperture of the human visual system is generated by drawing parts of the human eye on a texture. First black fibers and particles are drawn on a white texture using supersampling. Both types of primitives are drawn using one vertex per primitive containing the position. The same vertex shader is used for both types and forwards the position to a geometry shader. The geometry shader for fibers generates one quad per vertex and multiples each vertex position of the quad with the model-view-projection matrix while the pixel shader just outputs black for each pixel. The geometry shader for particles also generates one quad per vertex and performs the matrix-vector multiplications. Each vertex computed for a particle contains a texture coordinate in addition to the position. This texture coordinate is used in the pixel shader to discard pixels lying outside of a circle around the quad center and to output black for the others. Both geometry shaders use a constant defining the line width and the circle radius, respectively. These shaders are used to draw fibers in equidistant angles around the center of the aperture, large particles randomly placed on the texture, and small particles randomly distributed in the center of the aperture. After drawing the fibers and particles the aperture is downscaled by a factor of two to reduce pixelated edges.

Then the eyelashes are added. The eyelashes were hand-drawn in an image manipulation program and saved to an image file. This is then loaded to a texture and scaled to match the size of the aperture texture. The two textures are multiplied to add the black eyelashes on the aperture.

The final part of the human eye that is considered in this model is the pupil. The pupil is drawn as a black circle with the pupil diameter. This is combined in one compute shader with the complex exponential from Equation (4.15) that has to be multiplied with the aperture. To fit the other chosen parameters, the exponential is modified (see Equation (6.6)). The real part and the imaginary part of the exponential can be computed using the `sincos` function in the shader if the pixel is within the pupil. Otherwise, both parts are set to zero. Both parts are then multiplied with the input-aperture to get the real and imaginary parts of the final aperture. The aperture shown in Figure 4.2 is the aperture before this compute shader because it is the last aperture that is only real-valued. Therefore, the pupil is missing in that output image.

6.6.2 Diffraction Approximation

As the Fresnel diffraction is chosen to approximate the glare, the glare filter is computed using the generated complex aperture ($A(\mathbf{x}) \cdot E(\mathbf{x})$ in Equation (4.15)). Like the existing implementation, Equations (4.15) and (4.17) are combined to get the glare filter

$$\bar{F}(\mathbf{x}) = \sum_{j=0}^{n} \frac{\mathcal{S}(\lambda_j)}{(\lambda_j c_1 d)^2} \left| \mathcal{F}[A(\mathbf{u}) \cdot \exp(c_2 \pi i \cdot \langle \mathbf{u}, \mathbf{u} \rangle)] \left(\frac{\mathbf{x}}{\lambda_j c_1 c_3 d} \right) \right|^2 \tag{6.6}$$

with three constants c_1, c_2 and c_3. This follows the implementation of Frisvad [Fri09], but the squared terms of Equation (4.15) are used here, opposed to non-squared terms used by him. The constants are used to control the sampling positions, and to get a nice-looking glare filter.[6] A distance d between pupil and retina of $20.32\,\mathrm{mm}$ is used. The coordinates here are in the range $[-1, 1] \times [-1, 1]$.

As suggested by Ritschel et al. [Rit+09], $n = 32$ wavelengths between $\lambda_{\text{min}} = 380\,\mathrm{nm}$ and $\lambda_{\text{max}} = 770\,\mathrm{nm}$ are used for the sampling. A fast Fourier transform is computed once, and the `SampleLevel` function is used for bilinear sampling of the real and imaginary textures of the result. The colors are computed on the CPU using the code provided by Wyman et al. [see Wym+13, Listing 1], and the current wavelength and wavelength-dependent color are uploaded to the GPU. The sum is computed in a ping-pong way by adding the summand for each wavelength. This ensures that the FFT result gets sampled at similar positions for neighboring pixels, which improves the access read speed, compared to looping over all wavelengths in one shader and sampling in the loop. To ensure that the glare filter has a circular shape, the result is only written to a circular region of the texture.

After computing the sum, the glare filter is normalized by the sum of the luminance of all pixels like a point spread function. As the normalized glare is very weak, it gets multiplied by a constant factor[7] while the glare intensity can be further manipulated in the blending step. Finally, a Gaussian filter is applied to the glare filter before it is downsampled to the final size.

[6] $c_1 = 10^{-4}$, $c_2 = 2.5$ and $c_3 = 0.75 \cdot N_A/512$ are used to mimic the look of the glare of Frisvad, which is designed to look good with a glare size of 512 pixels.

[7] The glare is multiplied by a factor of 100 in this implementation.

6.6.3 Convolution

The glare filter $\bar{F}$ is convolved with the bright pixels B to generate a glare overlay $\hat{B}$, which is added to the tone mapped image $\hat{I}$. The convolution is done using FFTs as suggested by Equation (4.20). To eliminate wrap-around artifacts, the convolution has to be done on a texture of width

$$N_{x,\mathrm{FFT}} = 2^{\lceil \log_2(N_x + N_F) \rceil} \tag{6.7}$$

and height

$$N_{y,\mathrm{FFT}} = 2^{\lceil \log_2(N_y + N_F) \rceil}.$$

As the center of the glare filter is in the center of the texture, the result of the convolution has to be shifted by $-N_F/2$. The glare overlay is then a texture of width N_x and height N_y that is cut out of the shifted IFFT result.

6.6.4 Blending

As final composition step, the tone mapped image and the glare overlay are combined. A single compute shader is used to combine the tone mapped image and the glare overlay, as in Equation (4.21). Because the intermediate results are saved, the blending step can be performed very efficiently if the glare intensity changes.

The blending step is the final step of the implementation of the pipeline outlined in Section 4.1. After this step, all intermediate textures (each one corresponds to an image in Figure 4.2) are computed. A selected texture can then be displayed or saved. The software system that uses this implementation is described in Chapter 7.

7 The HDR-Toolkit System

The GPU-based implementation is used to create an easy-to-use software to visualize HDR image data. The software is built with the Windows 8.1 SDK to use the C++ and DirectX implementation outlined in Chapter 6 together with a user interface (UI) built with Microsoft's Extensible Application Markup Language (XAML). Screenshots of the application can be seen in Figure 7.1.

7.1 User Interface

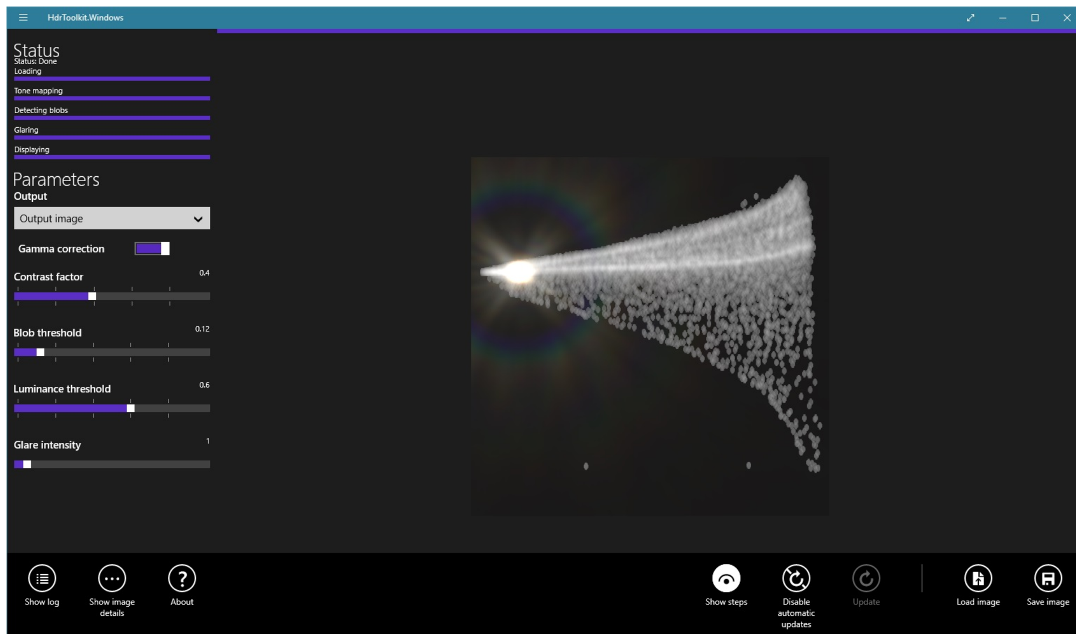
The `SwapChainPanel` class can be used to render to a DirectX `SwapChain`, while the `SwapChainPanel` is used as a UI element in XAML. The GPU-based implementation of the approach is implemented in a custom `SwapChainPanel` object based on Microsoft's `DirectXBasePanel` [Mic13]. The UI code can then request an update with a set of parameters. To fit all input image sizes, the `SwapChainPanel` is simply placed in a `ScrollViewer` to let the user zoom and scroll the image.

The parameters of the approach are mostly manipulated using `Sliders`. The contrast factor, blob threshold, and luminance threshold are `Sliders` ranging from zero to one, and the glare intensity `Slider` ranges from 0 to 20. All `Sliders` use steps of 0.01.

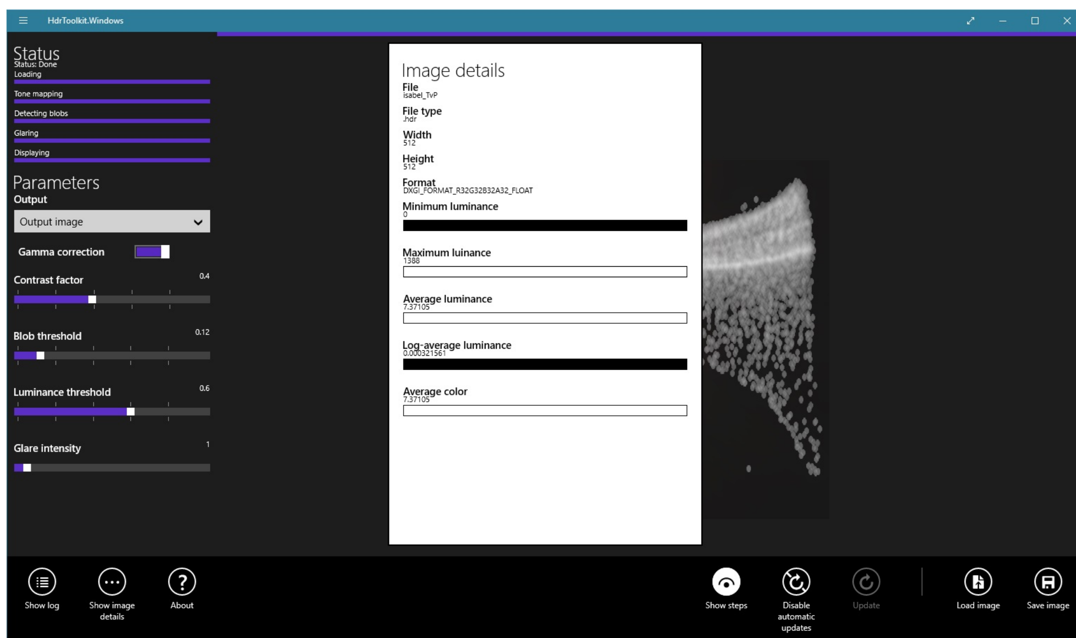
The output can be modified with a `ToggleButton` for the *gamma correction* and a `ComboBox` for the displayed texture. All intermediate images (visualized by images shown in Figure 4.2) can be displayed. A *gamma correction* with $\gamma = 2.2$ is applied on the chosen texture before rendering it on the `SwapChainPanel` if the option is chosen.

Further actions are controlled via a `CommandBar`. `AppBarButtons` for loading an image and saving the currently selected texture are located there. Other options related to the way the application updates the textures and shows textures are implemented as `AppBarToggleButtons`. One controls whether the textures should be updated as soon as a parameter changes, or only when an update is requested by the user. If updates should not happen immediately, a button gets enabled in the `CommandBar` to start an update. Otherwise, this button is disabled.

The other toggle button turns displaying of intermediate results on the `SwapChainPanel` on and off. When this option is off, only the selected texture is displayed on the panel, after the textures got updated. When this option is turned on, intermediate results are shown on the panel while other computations are done. These intermediate results include the



(a) Application with open `CommandBar`. A collapsed `CommandBar` is not visible for Windows 8 applications.



(b) Application displaying image details.

Figure 7.1: Screenshots of the HDR-Toolkit application using the Windows 8.1 SDK.

modified luminance while the optimization problem of the tone mapping is solved, or the blob mask, on which new circles are drawn for each scale of the scale-space.

Extra information is displayed using `Flyouts` and `ProgressBars`. One region above the parameters displays the progress of the different steps using multiple progress bars and one bar above the `SwapChainPanel` indicates the total progress. Details about the image can be displayed using an `AppBarButton` and a `Flyout`, while another pair of a button and a flyout can be used to display a log of the application. Both buttons are located in the `SecondaryCommands` section of the `CommandBar`, and the `Flyouts` are placed in the center of the application.

7.2 Image Files

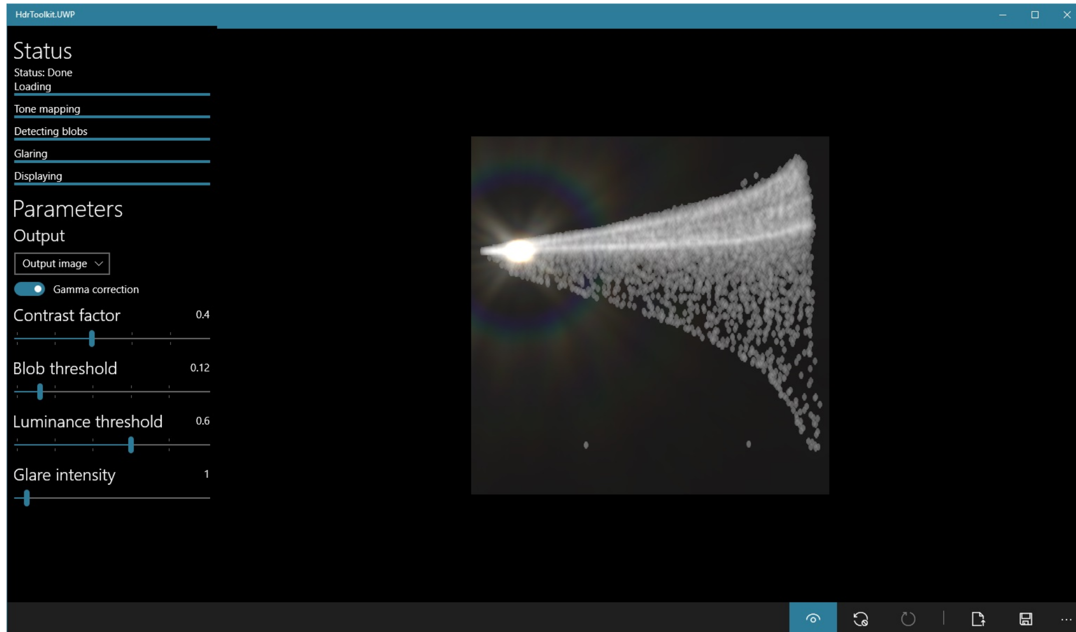
The application can be used to open HDR files using file-type associations. Files in the Radiance RGBE format with the extensions `.hdr` and `.rgbe` can be opened with the application using these associations. If the application was closed, it opens and processes the image as if it was opened using the button to open an image. If the application was not closed when opening a file using the file type association, the parameters get reset to the default values and the image gets processed.

Images with these file-types and images with other file-types can be loaded using the `AppBarButton` to load images. This can be used, for example, to load `.tiff` or `.dds` images. Images can be saved as `.jpg`, `.png`, or other LDR file-types, but also as HDR image files.

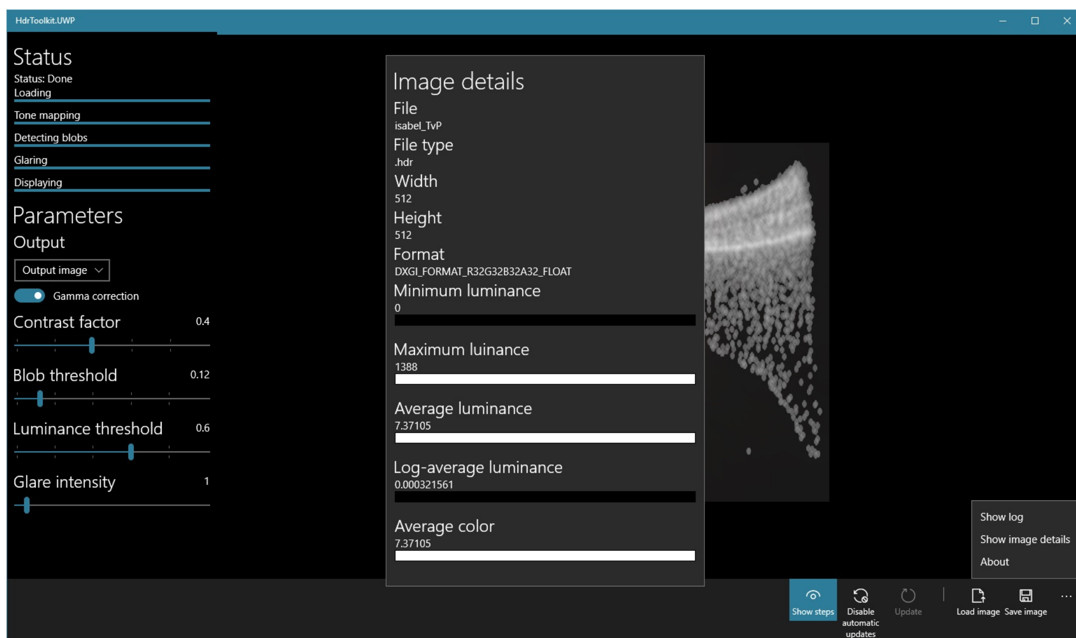
7.3 Other Features

All parts of the user interface can be localized using localized resources. Currently, the application is localized for en-US and de-DE language-region combinations. Furthermore, the application is compatible with newer SDKs and can be built with the Windows 10 Anniversary Update SDK without any modifications of the source code.[1] Screenshots of the Windows 10 application can be seen in Figure 7.2.

[1] A new manifest file and visual assets (logos and splash screen) have to be provided.



(a) Application with collapsed `CommandBar`.



(b) Application displaying image details.

Figure 7.2: Screenshots of the HDR-Toolkit application using the Windows 10 Anniversary Update SDK.

8 Evaluation and Examples

Further examples for tone mapped and glared images can be seen in Figures 8.1 to 8.6. Bright spots can be easily noticed due to the applied glare. The visualization of flight trajectories in Figures 8.2 and 8.3 can be used to spot the busiest airports at glance. For Figure 8.3 the default parameters are used, and high-intensity regions can be easily spotted in Europe and North America. For Figure 8.2 the blob detection is turned off, and the luminance threshold is set to 0.5. This corresponds to finding all regions, where the intensity is at least $50\,\%$ of the maximum intensity, which could be a typical task. Compared to the common visualization techniques shown in Figure 8.7, high-intensity regions can be seen as well as structural details. In Figure 8.4 high-intensity regions could not easily be spotted without the glare because of the tone mapping. The glared region in the lower right of the image helps to identify it as a high-intensity region. The glare also helps seeing high-intensity regions in Figure 1.1e. The relative intensity is lost using any tested TMO as seen in Figure 3.1, but the pixels with the highest intensity can be seen at a glance using our method.

Timings for images of different sizes can be seen in Table 8.1. The Windows 10 version of the software was used on a machine with a $2.20\,\mathrm{GHz}$ Intel Core i5-5200U processor and a $300\,\mathrm{MHz}$ Intel HD Graphics 5500 graphics card. The computation time was determined by using time points of the `std::chrono::system_clock` before the computation started and after it finished. The timings show that the software can be used on mobile computers to visualize images in reasonable time, but the performance is far from real-time.

The proposed approach has also some downsides. Because the glare is applied to pixels with tone mapped colors, the intensity in the output image does not represent the relative intensity of the input data. A glared region appears bright, if many pixels are in the region, not if the intensity of the pixels in the input is high. Additionally, for the pixels in the center of a glared region, only the brightness of the glare can be seen, while details are overlaid with the brightest pixels of the glare. A dynamic glare model could solve this by

Image	N_x	N_y	min	max	avg
Figure 8.5	512	512	4123.83 ms	4554.37 ms	4350.07 ms
Figure 8.6	1024	1024	14 377.91 ms	18 099.75 ms	15 677.77 ms
Figure 8.2	2048	1024	20 066.22 ms	20 402.08 ms	20 226.13 ms

Table 8.1: Timings for different input images. The width N_x and height N_y are given in pixels. The minimum (min), maximum (max), and average computation time (avg) are given for ten runs of the HDR-Toolkit software.

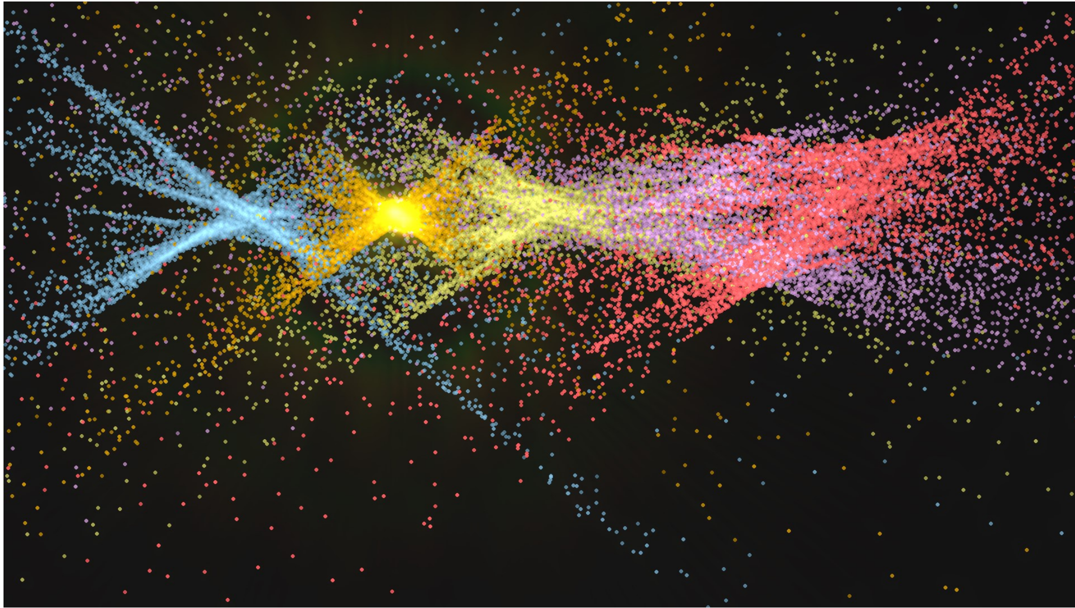


Figure 8.1: Visualization of 1-flat indexed points parallel coordinates [ZW17] with color indicating the subspace using the "power plant" data with $P_R = 0.4$, $T_M = 0.12$, $T_L = 0.6$, and $P_R = 7.5$.

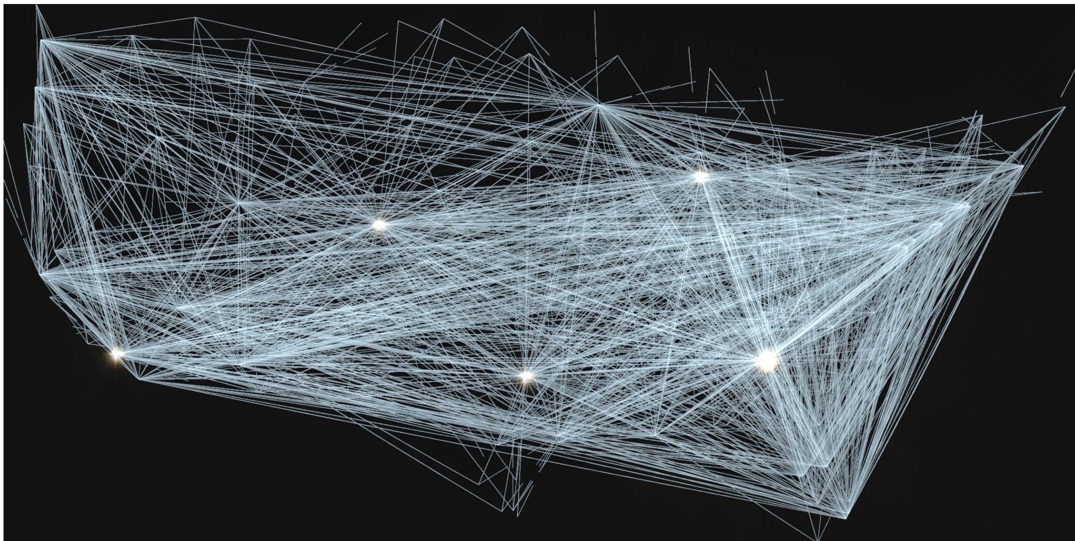


Figure 8.2: Visualization of trajectories using "flight" data for the United States of America with $P_R = 0.4$, $T_M = 0$, $T_L = 0.5$, and $P_R = 5$.

Figure 8.3: Visualization of trajectories using global "flight" data with $P_R = 0.4$, $T_M = 0.12$, $T_L = 0.6$, and $P_R = 2.5$.

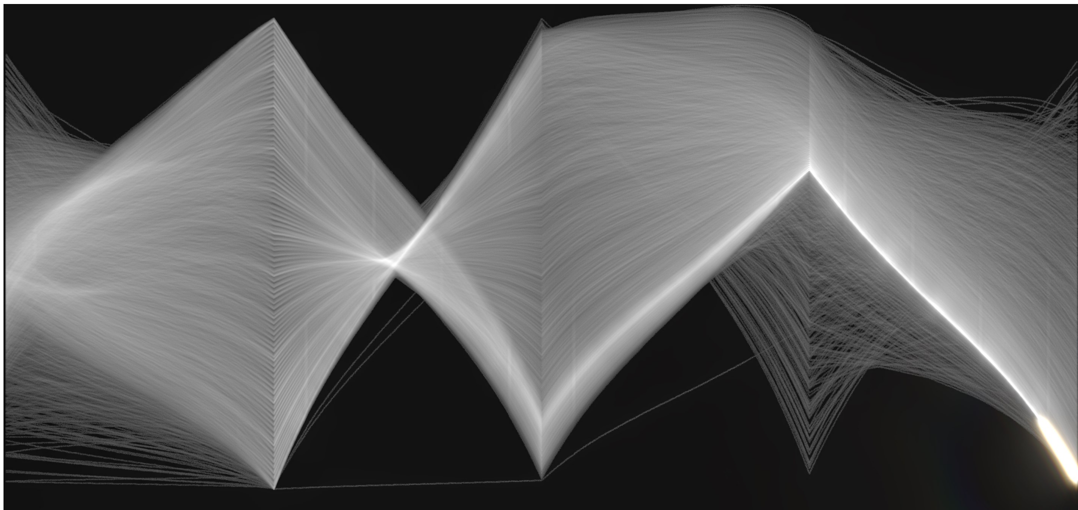


Figure 8.4: Visualization of parallel coordinates using the "Isabel" data with $P_R = 0.4$, $T_M = 0.12$, $T_L = 0.6$, and $P_R = 1$.

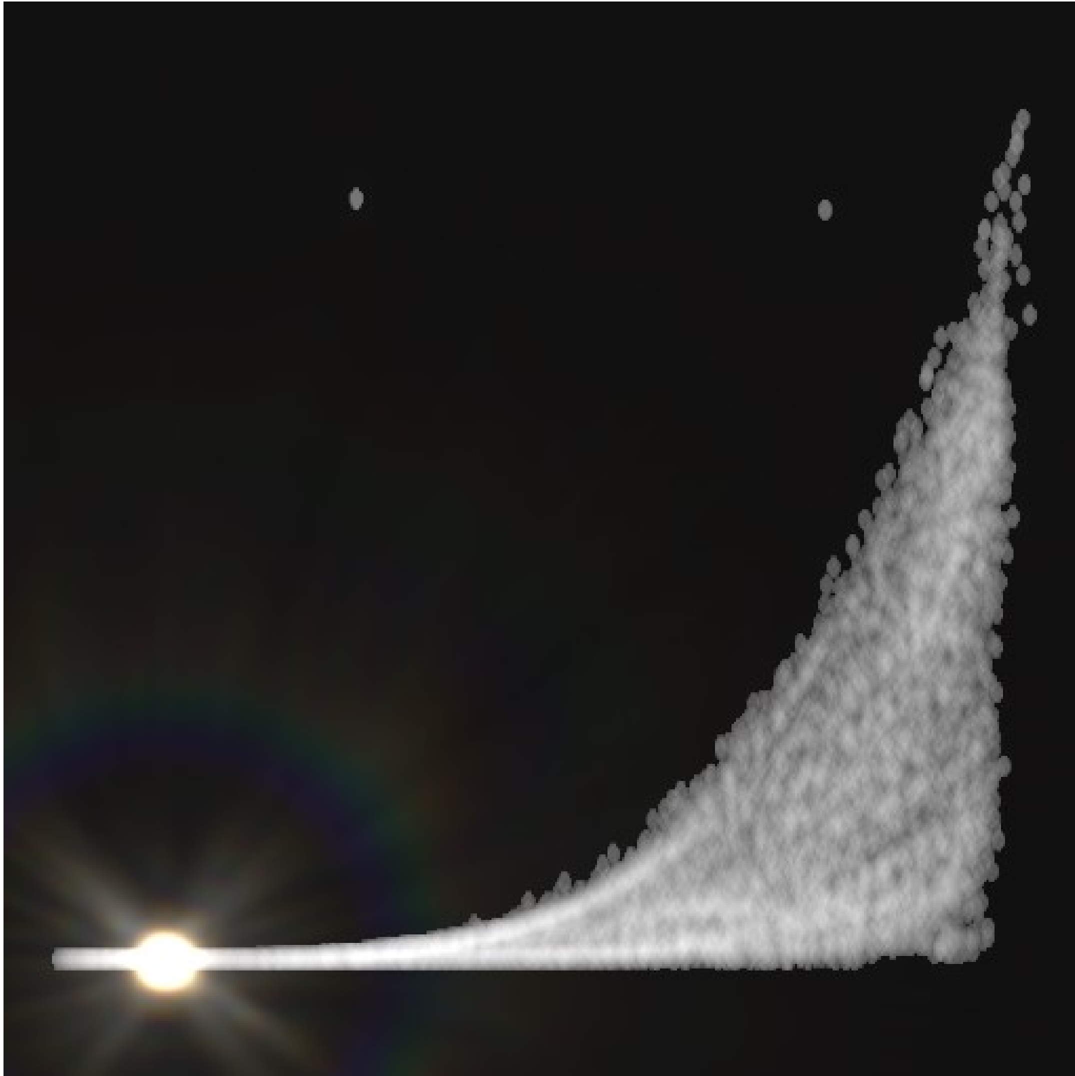


Figure 8.5: Visualization of a scatter plot using the “Isabel” data with $P_R = 0.4$, $T_M = 0.12$, $T_L = 0.6$, and $P_R = 1$.

allowing a reduction of glare intensity while the glared spots stay visible through slight fluctuations in time. The glare can also appear to be behind some structures (for example in Figure 1.1e). The bright pixel detection can be improved as well. The blob detection detects mostly circular shapes. Other methods to detect elliptical shapes or other features could be used instead.

The implementation also has some disadvantages. The application requires a lot of memory for large images. For example, an image of width and height of 2048 pixels requires textures of width and height of 8192 pixels to compute the FFT (see Equation (6.7)). Each of these texture uses $1\,\mathrm{GiB}$ of memory, and at least six textures are needed for the convolution. Some parts of the implementation can be implemented more efficiently as

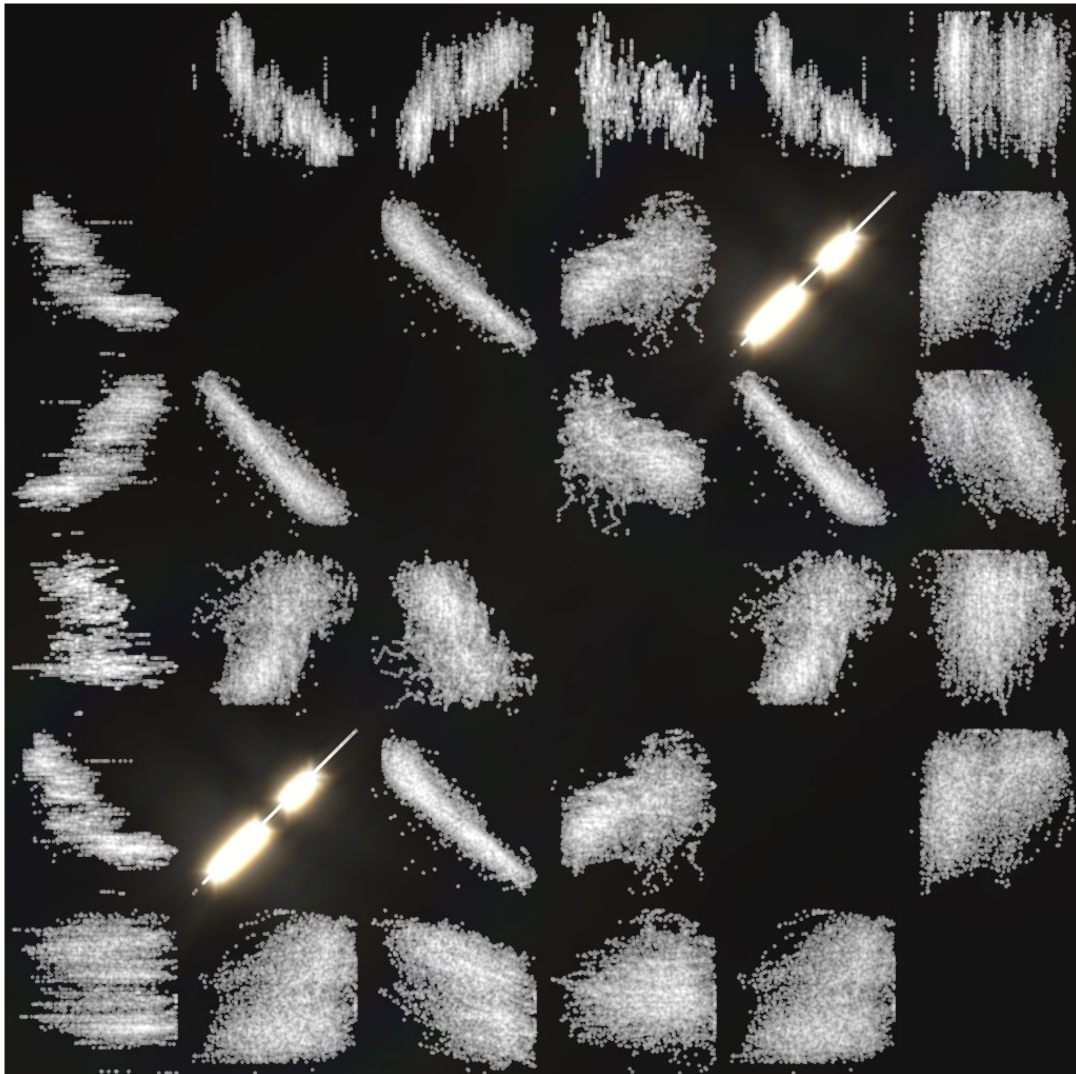
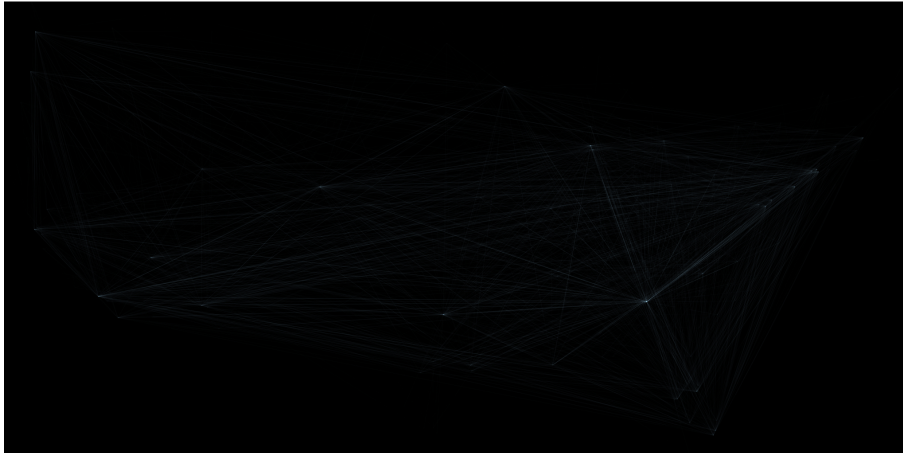
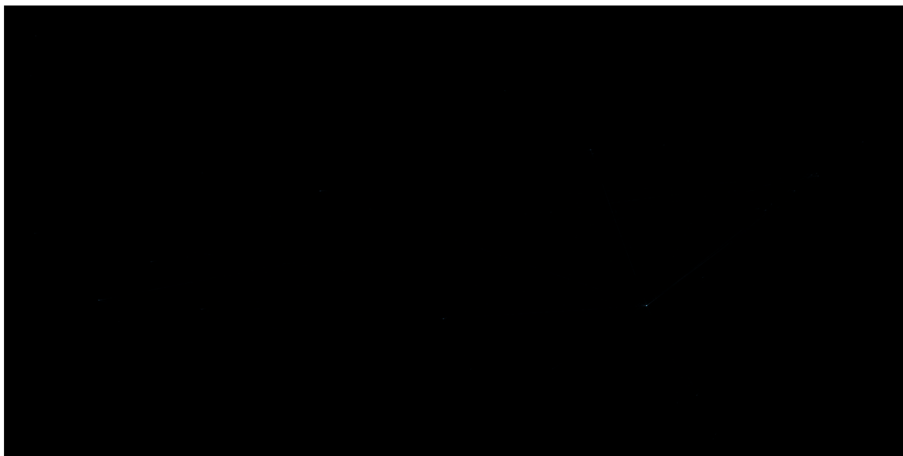


Figure 8.6: Visualization of a scatter plot matrix using the “power plant” data with $P_R = 0.4$, $T_M = 0.12$, $T_L = 0.6$, and $P_R = 1$.

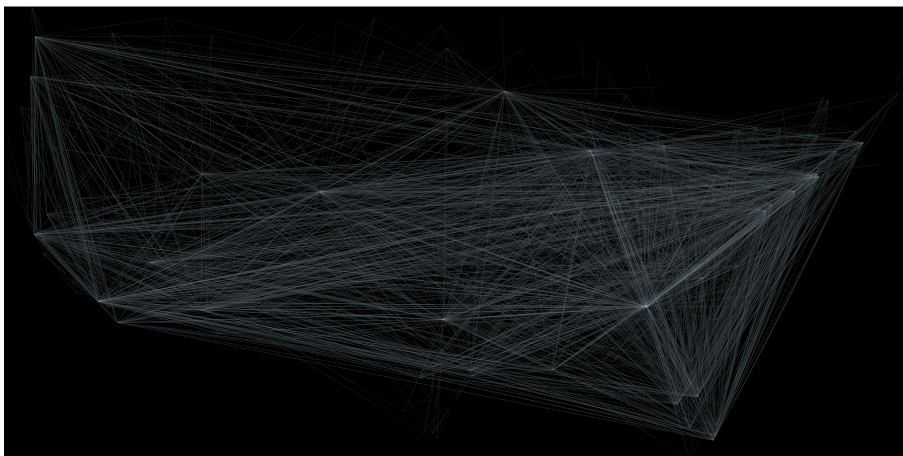
well. For example, the Gaussian blur of the scale-space could be computed with FFTs for large values of σ.



(a) Using linear mapping (see Equation (2.11)).



(b) Using logarithmic mapping (see Equation (2.12)).



(c) Using gamma mapping (see Equation (2.14) with $\gamma = 2.2$).

Figure 8.7: Visualization of trajectories using "flight" data for the United States of America with common visualization techniques.

9 Conclusion and Future Work

In this thesis, we have proposed a perceptual-based approach to better visualize density maps. The novel visualization model includes tone mapping with an appropriate TMO, bright pixel detection with blob detection, and a wave-optics-based glare simulation. The approach makes details in the input image visible and highlights pixels of high density with glares. In addition, a short evaluation of tone mapping operators on density maps was done. The evaluation showed that many operators are not suitable for density maps. An efficient GPU-based implementation and an easy-to-use application for this approach were provided as well. With various typical density maps, we have shown the effectiveness of the approach.

9.1 Conclusion

The evaluation of TMOs showed that many operators are not suitable for visualization of density maps. The TMO based on a perceptual framework [MMS06] appears to be a good TMO for HDR density maps. The glare generation is also applicable to density maps and highlights high-density regions in a noticeable manner. Blob detection can be combined with glare simulation to find interesting regions, but if high-density regions are of interest, a thresholding has to be applied as well. The proposed approach to combine tone mapping, glare simulation, and blob detection seems to be able to produce satisfying results with only a few required interactions but is also flexible enough to be modified to fit different types of input images.

9.2 Future Work

User studies will be carried out to evaluate the approach introduced in this work. This could also include more research in finding good TMOs and parameters to better visualize different types of density maps. The HDR-Toolkit software will be updated as well with efficient implementations (like using a GPU-sorting algorithm and FFTs for Gaussian blur). More features like offering a choice of different TMOs, different apertures, or dynamic glare could be added in the future.

Dedication

This thesis is dedicated to my parents, Ursula and Martin Rivinius.

Acknowledgments

Thanks to everybody who made helpful suggestions, gave valuable feedback, and for proofreading. Hurricane Isabel data was produced by the Weather Research and Forecast model, courtesy of the U.S. National Center for Atmospheric Research and the U.S. National Science Foundation. M. Lichman provided the “power plant” data in the UCI Machine Learning Repository. The HDR images and density maps were created by Liang Zhou from the respective data sets.

A Zusammenfassung

Density-Maps sind ein wichtiges Mittel der Datenvisualisierung. Dazu gehören Scatter-, Parallelkoordinaten-, oder Trajektorie-Plots. Density-Maps haben typischerweise einen hohen Dynamikumfang, der außerhalb des Dynamikumfangs von Anzeigegeräten liegt. Übliche Operatoren (wie lineare, logarithmische, oder Gamma-Mappings), die Daten in anzeigbare Werte umwandeln, liefern unzufriedenstellende Resultate, wobei Features verloren gehen oder irreführende Visualisierungen erzeugt werden.

In dieser Arbeit wird ein neuer Ansatz vorgestellt, um HDR-Density-Maps zu visualisieren. Wir werten die Effektivität verschiedener Tone-Mapping-Operatoren auf Density-Maps aus, um einen passenden TMO zu finden, mit denen der Dynamikumfang von Density-Maps reduziert werden kann. Mit einem guten TMO sind alle nicht-Null-Werte und viele Details erkennbar. Da nicht mehr gewährleistet ist, dass relative Intensitätsunterschiede erkennbar sind, werden Pixel mit hoher Intensität besonders hervorgehoben. Dafür wird eine Glare-Simulation verwendet. Glare tritt auf, wenn Lichtquellen mit sehr hoher Helligkeit betrachtet werden. Somit scheint Glaring ein intuitiver Weg zu sein, um hohe Intensitäten hervorzuheben. Um die Pixel mit hoher Intensität zu finden wird eine automatische Blob-Erkennung verwendet. Weder die Verwendung von Tone-Mapping-Operatoren, noch die von Glare-Simulation für Density-Maps ist bereits erforscht worden.

Die Evaluierung verschiedener Tone-Mapping-Operatoren hat ergeben, dass kaum ein Operator zufriedenstellende Ergebnisse liefern kann. Meist sind einige Regionen nicht erkennbar, oder es sind kaum Unterschiede zwischen einzelnen Pixeln zu erkennen. Ein Operator, der auf der menschlichen Wahrnehmung von Kontrast basiert [MMS06], lieferte gute Ergebnisse. Dieser wird in unserem Modell zur Visualisierung von Density-Maps verwendet.

Das Modell zur Visualisierung von HDR-Density-Maps besteht aus Tone-Mapping, Erkennung von Pixeln mit hoher Intensität und Glare-Simulation. Pixel mit hoher Intensität werden mit einer Scale-Space-Blob-Erkennung [Lin98] im Eingabebild erkannt. Die Position dieser Pixel wird verwendet, um Farben nach dem Tone-Mapping auszuwählen. Nur für diese wird Glare simuliert. Ein Glare-Filter wird mit einem Ansatz berechnet, der auf Wellenoptik basiert [Rit+09]. Der Teil des Tone-Mapping-Bildes, der als hell eingestuft wurde, wird mit dem Glare-Filter gefaltet. Das Ergebnis der Faltung wird schließlich mit dem Tone-Mapping-Bildes kombiniert.

Das Modell verfügt über vier Parameter, die verändert werden können, um das Ergebnis zu beeinflussen. Ein Kontrastfaktor des Tone-Mappings bestimmt, wie sehr der Kontrast komprimiert wird. Über einen Blob-Schwellwert und einen Luminanz-Schwellwert wird

gesteuert, welche Pixel als hell eingestuft werden. Ein letzter Parameter wird benutzt, um die Intensität des Glare zu verringern oder zu verstärken.

Das Modell wurde größtenteils mit Compute-Shadern implementiert, die auf der GPU ausgeführt werden. Für alle Bestandteile des Modells werden nur Komponenten-weise Berechnungen, oder Berechnungen, die nur Informationen einer kleinen Pixel-Nachbarschaft benötigen, durchgeführt. Somit ist die effiziente Implementierung der einzelnen Schritte möglich. Für die Implementierung des Modells ist auch eine einfach zu benutzende Anwendung entwickelt worden. Diese ermöglicht eine einfache Veränderung der Parameter über Schieberegler.

Die Ergebnisse, die mit unserem Modell erzeugt wurden, ermöglichen es, viele Details zu sehen, während Pixel mit hoher Intensität auf den ersten Blick erkannt werden können. Die Details sind mit den üblichen Operatoren größtenteils nicht sichtbar und Pixel mit hoher Intensität können mit den meisten Tone-Mapping-Operatoren nicht erkannt werden. Das Modell hat jedoch auch einige Nachteile. Beispielweise hat die Anwendung einen hohen Speicherverbrauch für große Bilder und die Berechnungszeit ist deutlich höher als bei den üblichen Operatoren. Außerdem ist die Stärke des Glares von der Größe einer hellen Region und nicht von der Intensität im Eingabebild abhängig.

Unser Ansatz stellt eine deutliche Verbesserung im Vergleich zu den üblichen Methoden der Density-Map-Visualisierung dar. Der Ansatz kann effizient implementiert werden und ist einfach zu benutzen. Andere Arten des Tone-Mappings, der Glare-Simulation und der Blob-Erkennung können untersucht werden, um das vorgestellte Modell zu verbessern, oder für andere Anwendungsgebiete anzupassen.

Bibliography

[AA05] N. Andrienko, G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer, 2005 (cit. on p. 14).

[ACK+17] D. Anastasia, F. Comida, D. Kaneider, et al. *Luminance HDR*. 2017. URL: http://qtpfsgui.sourceforge.net/ (visited on 10/10/2017) (cit. on p. 19).

[Ash02] M. Ashikhmin. "A Tone Mapping Algorithm for High Contrast Images." In: *Proceedings of the 13th Eurographics Workshop on Rendering*. 2002, pp. 145–156 (cit. on pp. 19–22).

[BW08] S. Bachthaler, D. Weiskopf. "Continuous Scatterplots." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1428–1435 (cit. on pp. 14, 15).

[Cha83] J. M. Chambers. *Graphical methods for data analysis*. Wadsworth International Group, 1983 (cit. on p. 14).

[DD02] F. Durand, J. Dorsey. "Fast Bilateral Filtering for the Display of High-dynamic-range Images." In: *ACM Transactions on Graphics* 21.3 (2002), pp. 257–266 (cit. on pp. 19–22).

[Dra+03] F. Drago, K. Myszkowski, T. Annen, N. Chiba. "Adaptive Logarithmic Mapping For Displaying High Contrast Scene." In: *Computer Graphics Forum* 22.3 (2003), pp. 419–426 (cit. on pp. 19–22).

[EDF08] N. Elmqvist, P. Dragicevic, J. D. Fekete. "Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation." In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1539–1148 (cit. on p. 14).

[FLW02] R. Fattal, D. Lischinski, M. Werman. "Gradient Domain High Dynamic Range Compression." In: *ACM Transactions on Graphics* 21.3 (2002), pp. 249–256 (cit. on pp. 19–22).

[Fer+11] S. Ferradans, M. Bertalmio, E. Provenzi, V. Caselles. "An Analysis of Visual Adaptation and Contrast Perception for Tone Mapping." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.10 (2011), pp. 2002–2012 (cit. on pp. 19–22).

[Fri09] J. R. Frisvad. *Glare demo*. 2009. URL: http://people.compute.dtu.dk/jerf/code/glare_demo.zip (visited on 08/12/2017) (cit. on pp. 9, 46, 48).

[HW09] J. Heinrich, D. Weiskopf. "Continuous Parallel Coordinates." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1531–1538 (cit. on pp. 14, 15).

[Kak+04] M. Kakimoto, K. Matsuoka, T. Nishita, T. Naemura, H. Harashima. "Glare Generation Based on Wave Optics." In: *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*. 2004, pp. 133–140 (cit. on pp. 17, 34).

[Kon+13] H. Kong, H. C. Akakin, S. E. Sarma. "A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications." In: *IEEE Transactions on Cybernetics* 43.6 (2013), pp. 1719–1733 (cit. on p. 31).

[Laz11] S. Lazebnik. *Blob detection*. 2011. URL: http://www.cs.unc.edu/~lazebnik/spring11/lec08_blob.pdf (visited on 10/20/2017) (cit. on p. 31).

[Lin98] T. Lindeberg. "Feature Detection with Automatic Scale Selection." In: *International Journal of Computer Vision* 30.2 (1998), pp. 79–116 (cit. on pp. 17, 30, 31, 65).

[MDK08] R. Mantiuk, S. Daly, L. Kerofsky. "Display Adaptive Tone Mapping." In: *ACM Transactions on Graphics* 27.3 (2008), p. 68 (cit. on pp. 19–22).

[MMS06] R. Mantiuk, K. Myszkowski, H.-P. Seidel. "A Perceptual Framework for Contrast Processing of High Dynamic Range Images." In: *ACM Transactions on Applied Perception* 3.3 (2006), pp. 286–308 (cit. on pp. 11, 16, 19–22, 26, 28, 29, 44, 61, 65).

[Mai+11] Z. Mai, H. Mansour, R. Mantiuk, P. Nasiopoulos, R. Ward, W. Heidrich. "Optimizing a Tone Curve for Backward-Compatible High Dynamic Range Image and Video Compression." In: *IEEE Transactions on Image Processing* 20.6 (2011), pp. 1558–1571 (cit. on pp. 19–22).

[Man+16] R. Mantiuk et al. *pfstools*. 2016. URL: http://pfstools.sourceforge.net/index.html (visited on 08/12/2017) (cit. on pp. 30, 44).

[Mic13] Microsoft. *XAML SwapChainPanel DirectX interop sample*. 2013. URL: https://code.msdn.microsoft.com/windowsapps/XAML-SwapChainPanel-00cb688b (visited on 09/19/2017) (cit. on p. 51).

[Mic17] Microsoft. *DirectXTex*. 2017. URL: https://github.com/Microsoft/DirectXTex (visited on 08/12/2017) (cit. on p. 43).

[NH06] M. Novotny, H. Hauser. "Outlier-Preserving Focus+Context Visualization in Parallel Coordinates." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 893–900 (cit. on p. 14).

[Pat+00] S. N. Pattanaik, J. Tumblin, H. Yee, D. P. Greenberg. "Time-dependent Visual Adaptation for Fast Realistic Image Display." In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. 2000, pp. 47–54 (cit. on pp. 19–22).

[RD05] E. Reinhard, K. Devlin. "Dynamic Range Reduction Inspired by Photoreceptor Physiology." In: *IEEE Transactions on Visualization and Computer Graphics* 11.1 (2005), pp. 13–24 (cit. on pp. 19–22).

[Rei+02] E. Reinhard, M. Stark, P. Shirley, J. Ferwerda. “Photographic Tone Reproduction for Digital Images.” In: *ACM Transactions on Graphics* 21.3 (2002), pp. 267–276 (cit. on pp. 16, 19–22, 26, 28).

[Rit+09] T. Ritschel, M. Ihrke, J. R. Frisvad, J. Coppens, K. Myszkowski, H.-P. Seidel. “Temporal Glare: Real-Time Dynamic Simulation of the Scattering in the Human Eye.” In: *Computer Graphics Forum* 28.2 (2009), pp. 183–192 (cit. on pp. 16, 17, 33, 34, 46, 48, 65).

[S.15] J. S. *Fast Fourier Transform for Image Processing in DirectX 11*. 2015. URL: https://software.intel.com/en-us/articles/fast-fourier-transform-for-image-processing-in-directx-11 (visited on 08/18/2017) (cit. on p. 42).

[SW89] L. Spillmann, J. Werner. *Visual Perception: The Neurophysiological Foundations*. Academic Press, 1989 (cit. on p. 9).

[Spe+95] G. Spencer, P. Shirley, K. Zimmerman, D. P. Greenberg. “Physically-based Glare Effects for Digital Images.” In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. 1995, pp. 325–334 (cit. on pp. 9, 10, 16, 17, 33, 46).

[Wym+13] C. Wyman, P.-P. Sloan, P. Shirley. “Simple Analytic Approximations to the CIE XYZ Color Matching Functions.” In: *Journal of Computer Graphics Techniques* 2.2 (2013), pp. 1–11 (cit. on pp. 34, 48).

[ZW17] L. Zhou, D. Weiskopf. “Indexed-Points Parallel Coordinates Visualization of Multivariate Correlations.” In: *IEEE Transactions on Visualization and Computer Graphics* in press (2017) (cit. on p. 56).

[Int15a] International Color Consortium. *How to interpret the sRGB color space (specified in IEC 61966-2-1) for ICC profiles*. 2015. URL: http://www.color.org/chardata/rgb/sRGB.pdf (visited on 09/19/2017) (cit. on pp. 15, 34).

[Int15b] International Telecommunication Union. *Recommendation ITU-R BT.709-6: Parameter values for the HDTV standards for production and international programme exchange*. 2015. URL: http://www.itu.int/rec/R-REC-BT.709 (visited on 09/19/2017) (cit. on p. 14).

[Ča+08] M. Čadík, M. Wimmer, L. Neumann, A. Artusi. “Evaluation of HDR Tone Mapping Methods Using Essential Perceptual Attributes.” In: *Computers & Graphics* 32.3 (2008), pp. 330–349 (cit. on pp. 9, 16).

All other links were last visited on October 24, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature