# Hardware-Accelerated Algorithms in Visual Computing

Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von
Pascal Gwosdek

Saarbrücken
2011

| | |
|---|---|
| Tag des Kolloquiums: | 11.07.2012 |
| Dekan: | Prof. Dr. Mark Groves |
| Prüfungsausschuss: | Prof. Dr. Thorsten Herfet<br>Universität des Saarlandes (Vorsitz) |
| | Prof. Dr. Joachim Weickert<br>Universität des Saarlandes (1. Gutachter) |
| | Prof. Dr. Horst Bischof<br>Technische Universität Graz (2. Gutachter) |
| | Dr. Christian Schmaltz<br>Universität des Saarlandes |

## Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, 11.07.2012

## Veröffentlichungs- und Übereinstimmungserklärung

Ich übertrage der Saarländischen Universitäts- und Landesbibliothek (SULB) das Recht, diese Arbeit in elektronischer und gedruckter Form zugänglich zu machen. Ich bestätige, dass die hierzu übermittelte elektronische Version der Arbeit mit der genehmigten Originalfassung in Form und Inhalt übereinstimmt.

Saarbrücken, 11.07.2012

# Contents

## Short Abstract

This thesis presents new parallel algorithms which accelerate computer vision methods by the use of graphics processors (GPUs) and evaluates them with respect to their speed, scalability, and the quality of their results. It covers the fields of homogeneous and anisotropic diffusion processes, diffusion image inpainting, optic flow, and halftoning.

In this turn, it compares different solvers for homogeneous diffusion and presents a novel 'extended' box filter. Moreover, it suggests to use the fast explicit diffusion scheme (FED) as an efficient and flexible solver for nonlinear and in particular for anisotropic parabolic diffusion problems on graphics hardware. For elliptic diffusion-like processes, it recommends to use cascadic FED or Fast Jacobi schemes. The presented optic flow algorithm represents one of the fastest yet very accurate techniques. Finally, it presents a novel halftoning scheme which yields state-of-the-art results for many applications in image processing and computer graphics.

## Kurzzusammenfassung

Diese Arbeit präsentiert neue parallele Algorithmen zur Beschleunigung von Methoden in der Bildinformatik mittels Grafikprozessoren (GPUs), und evaluiert diese im Hinblick auf Geschwindigkeit, Skalierungsverhalten, und Qualität der Resultate. Sie behandelt dabei die Gebiete der homogenen und anisotropen Diffusionsprozesse, Inpainting (Bildvervollständigung) mittels Diffusion, die Bestimmung des optischen Flusses, sowie Halbtonverfahren.

Dabei werden verschiedene Löser für homogene Diffusion verglichen und ein neuer 'erweiterter' Mittelwertfilter präsentiert. Ferner wird vorgeschlagen, das schnelle explizite Diffusionsschema (FED) als effizienten und flexiblen Löser für parabolische nichtlineare und speziell anisotrope Diffusionsprozesse auf Grafikprozessoren einzusetzen. Für elliptische diffusionsartige Prozesse wird hingegen empfohlen, kaskadierte FED- oder schnelle Jacobi-Verfahren einzusetzen. Der vorgestellte Algorithmus zur Berechnung des optischen Flusses stellt eines der schnellsten und dennoch äußerst genauen Verfahren dar. Schließlich wird ein neues Halbtonverfahren präsentiert, das in vielen Bereichen der Bildverarbeitung und Computergrafik Ergebnisse produziert, die den Stand der Technik repräsentieren.

# Abstract

This thesis describes the acceleration of computer vision methods by the use of graphics processors (GPUs). Besides the design and the optimisation of efficient parallel algorithms, it particularly covers the evaluation with respect to their speed, scalability, and the quality of their results.

On the one hand, this thesis deals with techniques which are based on partial differential equations (PDEs). This includes homogeneous and non-linear, potentially anisotropic diffusion processes, their elliptic extension to diffusion-driven image inpainting methods, as well as the determination of motion information in image sequences, the so-called optic flow. On the other hand, this thesis also covers the transformation of images into binary halftones, i.e. exclusively black and white images. For all of these methods, it presents GPU-based algorithms which solve the respective task in a fraction of the time required by a conventional desktop processor (CPU).

Moreover, this thesis also focusses on the central question about the magnitudes by which techniques from the various fields of visual computing can be accelerated by GPUs. To this end, it describes the implementation of different numerical approaches on this architecture and compares them to each other.

Homogeneous diffusion is implemented by explicit and implicit solvers for discrete diffusion problems, as well as by methods that are based on frequency filters. This includes the convolution with a discrete Gaussian kernel, multiplications in the frequency domain, recursive filters, and iterated box filters. As a part of the latter group, this thesis also presents a novel 'extended' box filter. This new filter performs well on CPUs, but it cannot reach the performance of the recursive filter on GPUs.

In the field of non-linear, potentially anisotropic diffusion processes, this work follows the recently published 'fast explicit diffusion scheme' (FED) [GWB10], and introduces it as an efficient parallel solver for GPUs and other massively parallel platforms. Its superior performance compared to traditional explicit and implicit schemes carries over to GPUs. This is shown for the example of edge and coherence enhancing diffusion [Wei11a].

A similar numerical scheme based on FED is well suited to fill in missing

image information by means of anisotropic or homogeneous diffusion image inpainting schemes. For further increasing the convergence behaviour, a uni-directional multigrid method is employed [GWB10].

The concepts presented in the context of the previously treated PDEs are extended to a modern model for motion estimation. Although this technique belongs to the most complex visual computing methods in terms of its algorithmic structure, the presented FED-based algorithm enjoys enormous speedups over the CPU. At the time of publication, the international Middlebury benchmark [BSL$^+$11] listed this algorithm on the 6th rank with respect to accuracy, where it represents the fastest method among the top 10. By the application of a 'Fast Jacobi' scheme (FJ) [Wei11b], its performance is increased even further.

Finally, this thesis also presents a novel halftoning method based on electrostatic particle models, which has many applications also in the field of importance sampling. The algorithmic solution of this problem requires too much time on conventional platforms, such that its applicability would normally be limited. However, the presented efficient GPU-based algorithms for this purpose result in a tremendous speedup. Among those is the first GPU-based 'fast summation' technique that sets up on non-equispaced fast Fourier transforms (NFFTs). It causes a runtime which opens doors for extensive research in this field: In order to demonstrate the versatility of the method, this thesis presents many adaptations and application examples. This includes a dithering method, an improvement of saturated regions in point-based halftones, multichannel halftoning, a support of points with different sizes, and an importance sampling technique that allows different primitives in different sizes. In many of these application areas, these methods yield state-of-the-art results.

# Zusammenfassung

Diese Arbeit befasst sich mit der Beschleunigung von Methoden der Bildinformatik unter der Verwendung von Grafikprozessoren (GPUs). Neben dem Entwurf und der Optimierung von effizienten parallelen Algorithmen steht vor allem auch deren Evaluation in Hinblick auf ihre Geschwindigkeit, Skalierbarkeit und die Qualiät der Ergebnisse im Vordergrund.

Auf der einen Seite behandelt diese Arbeit dabei Verfahren, die auf partiellen Differenzialgleichungen (PDEs) beruhen. Darunter fallen homogene sowie nichtlineare und potenziell anisotrope Diffusionsgleichungen, deren elliptische Erweiterung auf diffusionsgetriebene *Inpainting*-Modelle, sowie die Berechnung von Bewegungsinformationen in Bildsequenzen, dem sogenannten optischen Fluss. Daneben befasst sich diese Arbeit auch mit der Transformation von Bildern in binäre, d.h. ausschließlich schwarz-weiße, Halbtongrafiken. Für all diese Aufgaben werden GPU-basierende Algorithmen vorgestellt, die das gegebene Problem in einem Bruchteil der Zeit lösen, die ein gewöhnlicher Prozessor (CPU) dafür benötigt.

Im Fokus dieser Arbeit steht aber auch die zentrale Fragestellung, in welchem Umfang Methoden aus verschiedenen Teilbereichen der Bildinformatik mittels GPUs überhaupt beschleunigt werden können. Dazu werden verschiedene numerische Verfahren auf dieser Plattform umgesetzt und gegeneinander verglichen.

Für die homogene Diffusion kommen dabei neben expliziten und impliziten Lösern für diskrete Diffusionsprobleme auch Verfahren zum Einsatz, die auf Frequenzfiltern beruhen. Neben Faltungen mit einem diskretisierten Gaußkern werden Multiplikationen im Frequenzraum, rekursive Filter, sowie iterierte Mittelwertfilter angewendet. Im Rahmen der letzteren Gruppe stellt die Arbeit auch einen neuen 'erweiterten' Mittelwertfilter vor, der sich zwar auf traditionellen Architekturen als vorteilhaft erweist, jedoch auf GPUs nicht an die Leistung eines rekursiven Filters heranreicht.

Im Bereich der nichtlinearen, potenziell anisotropen Diffusionsprozesse orientiert sich die Arbeit am kürzlich vorgestellten 'schnellen expliziten Diffusionsverfahren' (FED) [GWB10], und führt es als effizienten parallelen Löser auf Grafikkarten und anderen massivparallelen Plattfor-

men ein. Die vorteilhafte Geschwindigkeit gegenüber herkömmlichen expliziten und impliziten Schemata lässt sich dadurch mit nur geringem Aufwand auch auf GPUs übertragen. Dies wird am Beispiel der kanten- und kohärenzverstärkenden Diffusion [Wei11a] vorgeführt.

Eine ähnliche auf FED basierende Numerik eignet sich auch hervorragend für die Vervollständigung fehlender Bildinformationen mittels anisotroper oder homogener Diffusion. Hier wird zur zusätzlichen Steigerung der Konvergenz ein unidirektionales Mehrgitterverfahren eingesetzt [GWB10].

Die im Rahmen der bisher behandelten PDEs vorgestellten Konzepte werden schließlich auf ein modernes Modell zur Bewegungsschätzung erweitert. Obwohl dieses hinsichtlich der algorithmischen Struktur zu den komplexeren Verfahren der Bildinformatik gehört, erreicht der auf FED basierte Algorithmus enorme Beschleunigungen gegenüber der CPU. Zum Zeitpunkt der Veröffentlichung listete der Middlebury-Benchmark [BSL+11] das Verfahren auf Rang 6 im Bezug auf die Genauigkeit, wo es die schnellste Methode unter den Top 10 stellte. Diese Leistung wird durch den Einsatz eines 'schnellen Jacobi-Verfahrens' (FJ) [Wei11b] sogar noch erhöht.

Schließlich präsentiert diese Arbeit auch ein neues, auf elektrostatischen Partikelmodellen beruhendes Halbtonverfahren vor, das auch vielfältige Anwendung im Bereich der adaptiven Abtastung hat. Die algorithmische Lösung dieses Problems benötigt zu viel Zeit auf gewöhnlichen Plattformen, sodass die Anwendung des Verfahrens normalerweise stark eingeschränkt wäre. In dieser Arbeit werden jedoch schnelle Algorithmen vorgestellt, die die Laufzeit drastisch senken. Darunter ist auch das erste GPU-basierte Verfahren zur 'schnellen Summierung', welches auf schnelle Fouriertransformationen für ungleichmäßig verteilte Datenpunkte (NFFT) zurückgreift. Die neu gewonnene Geschwindigkeit erlaubt eine umfangreiche Forschung auf dem Gebiet: Als Beispiele für die Vielfältigkeit des Verfahrens stellt die Arbeit eine gitterbasierte Halbtonmethode, eine Verbesserung stark gesättigter Bildbereiche in punktbasierten Halbtonbildern, Mehrkanal-Halbtonverfahren, eine Unterstützung verschiedener Punktgrößen, sowie ein adaptives Abtastverfahren mit unterschiedlichen Geometrieelementen unterschiedlicher Größe vor. Viele dieser Auskopplungen repräsentieren den aktuellen Stand der Technik.

## Acknowledgements

During my work on this project, I enjoyed the support of many people without whom this thesis would never have become reality. I would like to take this chance to express my thanks to everybody who contributed to this success.

My thanks go to Prof. Dr. Joachim Weickert for supplying the idea for this exciting project, for supervising my thesis, and for giving me the opportunity to work in his group. I also wish to express my gratitude to the cluster of excellence 'Multimodal Computing and Interaction' who enabled me to pursue this project by funding my work. Furthermore, I like to thank Prof. Dr. Horst Bischof for agreeing to become an external reviewer.

I owe my special thanks to my friends Sven Grewenig, Markus Mainberger, Christian Schmaltz, and Henning Zimmer, who spent many days and nights on the proofreading of this thesis. Without their appreciated help, their day-and-night support, and their instant replies on e-mails, I could not even hope to have reached my submission deadline.

Moreover, I want to thank all current, former, associated, and visiting members of the MIA group. The great atmosphere and the many activities we enjoyed together were a beautiful experience. In particular, I would like to express my thanks to Marcus Hargarter and Ellen Wintringer who solved many of my problems, often without me even noticing it. Their constant effort was a relaxing relief. I also thank all my colleagues who were always there to discuss interesting ideas, some of which finally led to exciting publications. In particular, I would like to mention Christian Schmaltz, Sven Grewenig, Markus Mainberger, Andrés Bruhn, and Henning Zimmer. It is amazing to see how team spirit turns into great things.

Last but not least, I wish to thank my family and friends who supported me in all situations, and in particular within the recent stormy days. Many people stood beside me and helped out whenever they could. While it is impossible to name them all, I would like to say a big thank you to my parents Jörg and Silvia Gwosdek, to Lena Gwosdek and Alex Maida, Michael Bauer, Marcus Hargarter, Markus Mainberger, Martin Grochulla, and Sven Grewenig.

# Chapter 1

# Introduction

*The world is changing very fast.*
*Big will not beat small anymore.*
*It will be the fast beating the slow.*
Rupert Murdoch

## 1.1 Motivation

Visual computing comprises some of the most exciting topics in computer science. All subjects in this field are in one or the other way concerned with the human visual system, and by this they create and promote an intuitive interface between computers and our real world: *Image processing* targets at the automatic improvement, modification, or simplification of images. *Computer vision* enables machines to understand a depicted scene, and to extract and categorise advanced features such as motion, structure, or material properties. *Computer graphics* reverts this process by creating realistic images out of abstract descriptions of a scene.

Techniques from all of these fields are omnipresent in our everyday lives, and with the development of highly accurate models their influence is still rising. Machines take over parts of our work, and accelerate this process by being less flexible, but much faster and reliable. We trust in computer vision systems to detect defects in materials, to help us steer our cars, or to warn us from sudden situations of danger. We entrust our own health to image processing, since it can detect diseases or anomalies, or help the surgeon during complicated operations.

The tremendous improvements in visual computing are one of the rea-

sons for the high standards of safety, medical care, economical productivity, and consumer technology we enjoy nowadays. However, the rising needs in many of these fields also lead to a severe bottleneck. More accurate methods require in general a higher computational workload. This challenge can be mastered by supplying better numerical schemes, more compute resources, or by investing more time. For many applications, the latter option is inapplicable because certain time constraints hold. As a consequence, we observe a vivid research in the development of new numerical schemes and efficient algorithms.

If this performance is still not sufficient to solve all arising problems in the desired time, we have two choices. The most defensive yet not satisfying solution is the reduction of the problem size or algorithmic complexity. This yields less accurate results, which are nevertheless often accepted as a necessary compromise. However, there is also a second option which promises to provide the original quality in a higher runtime: special hardware.

In the last decade, graphics devices advanced to a popular coprocessor for applications in visual computing. On the one hand, this is motivated by their high availability and low prices. Today, it can be assumed that every modern desktop computer is equipped with a powerful programmable graphics card which is often underutilised. On the other hand, graphics cards are designed for a high data throughput and massively parallel operations — which exactly matches the profile of many modern visual computing algorithms. Moreover, the development of graphics processors (GPUs) enjoyed a rapid progress in the last few years. Modern GPUs are almost one order of magnitude faster than standard CPUs[NVi11b], which promises a performance that normally only applied to expensive high-performance compute clusters or custom-built hardware.

As a consequence, there is a variety of works in the literature that use GPUs to accelerate visual computing applications. The first approaches in general purpose computing on GPUs (GPGPU) were still closely related to the original orientation of GPUs on graphics processing [CN93, WE98]. Although the purpose of these works still was the visualisation of complex data on a graphics device, they already exploited the high performance of dedicated texturing circuits and programmable shaders to accelerate the processing of 3-D datasets.

Soon, this tight binding between computations and visualisation was released in favour of using the GPU as a real numerical co-processor for matrix-valued computations [RS01, Göd05]. This *traditional* GPGPU idea was based on a few general concepts to encode a mathematical problem in image buffers and blending operations. After being processed in the OpenGL or DirectX pipelines of the graphics device, the resulting im-

age could be re-interpreted to the sought solution. This gave rise to the development of libraries which provide an abstraction layer to the programmer, and offer visual computing primitives as high-level functions [FMA05, BS08, AHAS08]. By doing so, they hide the complicated and time-consuming re-coding and execution stages from the programmer.

In the following years, graphics card manufacturers officially started to support GPGPU, and developed high-level languages for an easier and faster access to the CPU [Hen07, NVi11b, Khr10, Adv11]. The accompanying simplification to the design process caused a massive number of new GPU-based parallel computer vision algorithms and a rising competition in this field. Besides many standalone algorithms for specific applications, there are also high-level visual computing libraries that use the high-level CUDA API by NVidia [NVi11b] to accelerate their computations [Cor11, Ope11], as well as loose collections of algorithms and libraries for complex vision applications [PBH+11].

The work presented in this thesis pursues similar objectives, but its ambitions are much higher: Many other works are focussed on obtaining real-time performance with algorithms that already possess near-realtime performance on traditional architectures. This is driven by their direct applicability in large-scale processes, and is often endorsed and co-financed by industry. Such efforts lead very often to extremely fast algorithms which may sometimes even be implemented in dedicated hardware. However, since these algorithms are tailored towards a good runtime performance, they do often not yield state-of-the-art results.

Our motivation in this thesis is slightly different. While we are also interested in an optimal runtime, we intentionally place this demand behind the requirement of a high quality of the results. We base our efficient GPU-based algorithms on some of the most accurate models in their fields. Hence, our work also explores the limits that arise for the fast solution of complex problems in visual computing. To this end, we also obtain an overview map of expectations we can put into the hardware-accelerated solution of other, potentially more complex tasks.

In the first part of this thesis, we review those image processing and computer vision techniques that are based on partial differential equations (PDEs). They have a good reputation for their high quality and flexibility, but nevertheless people frequently abstain from using them for real-world applications. Often, PDE-based algorithms are brought into disrepute wholesale for being too slow to solve a particular task, or people simply do not feel experienced enough to design a PDE-based algorithm for a new problem. In the following chapters, we are going to see that PDEs can lead to fast and accurate solutions for many problems. Moreover, we learn

that the additional challenge of designing a GPU-based algorithm can be extremely simplified if we adhere to a few very simple strategies.

The second part of this thesis is then concerned with a new physical model for the binarisation of image, the so-called *halftoning*. While the algorithmic complexity for this method is even higher than for many PDEs, we are also going to see that GPUs can yield significant speedups over traditional architectures. This advantage promotes and opens the novel method to a high number of exciting applications, where it yields state-of-the-art results.

## 1.2 Goals

The goals of this thesis are the development of efficient GPU-based algorithms for the most important problems in visual computing, and the investigation of their scaling behaviour on massively parallel architectures. Our new GPU-based algorithms shall not only provide a similar quality as established methods on the CPU in a much shorter time. They shall also explore the general limitations and pitfalls which occur in the development of such algorithms.

Our algorithms are supposed to present efficient approaches to general concepts rather than solutions for specific problems. By this, they allow universal insights on the parallelisation gain of typical visual computing problems, and allow an easy adaptation to new, potentially more complex, tasks. In particular, we are interested in efficient numerical and algorithmic schemes which are suited to solve these problems efficiently on GPUs. Even if hardware changes, it is likely that these general strategies do not lose their relevance.

## 1.3 Outline

This thesis is organised as follows. We continue with a short introduction into GPU programming in Chapter 2. Following this technical review of the underlying architecture, we focus on the area of PDE-based image processing and computer vision. In Chapter 3, we are concerned with *homogeneous diffusion* which is represented by one of the simplest PDEs. We compare many different numerical solutions of this problem, and show that they possess fundamentally different runtimes on the GPU. Chapter 4 then examines parabolic *anisotropic diffusion* processes. We employ the recently proposed FED scheme by Grewenig *et al.* [GWB10] as an efficient

and well-parallelisable solver for arbitrary diffusion-like processes on GPUs. In Chapter 5, we see that similar ideas carry over to the elliptic setting, and propose a parallel algorithm for *PDE-based image inpainting*. This concept is further generalised in Chapter 6, where we present an efficient GPU-based solver for *variational optic flow*. In this context, we also develop an algorithm based on the FJ scheme by Weickert [Wei11b] which seems to be even better suited for elliptic processes than FED. After this chapter, we leave the area of PDE-based visual computing models. Chapter 7 presents a novel technique for *halftoning* and *sampling* that has many applications in image processing and computer graphics. This technique yields state-of-the-art results for many applications, but the CPU-based method is inherently slow. Our new efficient algorithms yield results in a much smaller runtime. We use this advantage to extend the method to a broad range of applications. The core of this thesis then closes with a summary in Chapter 8. It is supplemented by a collection of proofs in Appendix A, by an academic résumeé, and by a list of bibliographic references.

# Chapter 2

# CUDA

*The utopist sees the paradise.*
*The realist sees the paradise with the snake.*
Friedrich Hebbel

## 2.1 Graphics Processing Units

Graphics Processing Units (GPUs) are made for the efficient processing of large vector-based data sets. Thanks to an optimised hardware layout, modern GPUs are able to process thousands of elements of such vectors in parallel. However, this specialised hardware design also implies that the use of GPUs requires programming concepts which differ fundamentally to those that are commonly applied on traditional hardware. Hence, all algorithms that should be accelerated by GPUs must be re-designed on the whole. This design process calls for a profound knowledge of the underlying architecture and possible race conditions that can occur in the massively parallel setting. These requirements, and the lack of a comfortable development environment, make the design and optimisation of GPU-aided algorithms a highly challenging task.

In this chapter, we briefly review the characteristics of modern GPUs, and compare them to the corresponding properties of classical CPU architectures. We discuss the consequences on the layout of algorithms, and point out differences and similarities to sequential concepts, as well as to traditional parallelisation concepts. To this end, we follow closely the official documentation for GPUs manufactured by NVidia [NVi11b], but many of the described ideas directly carry over to other brands and models. This

holds in particular because we are primarily interested in the impact of architectural characteristics on algorithms in the field of visual computing.

### 2.1.1 Performance

In the last decade, graphics cards enjoyed a boost in performance. When the first programmable graphics cards were published in 2003, their performance was lower than $5.0 \cdot 10^{10}$ single precision floating point operations per second (Flops/s) [NVi11b]. In contrast, a modern NVidia GeForce GTX 580 published in 2011 provides more than $1.5 \cdot 10^{12}$ Flops/s. This corresponds to a speedup of more than 30 over a period of less than 10 years. However, this rapid development also causes 30 times more data to be processed in a certain time interval. As a consequence, also the memory bandwidth was increased tremendously within this period. While old devices from 2003 have a memory throughput of less than 20 GB/s, modern graphics cards almost reach 200 GB/s and possess an elaborate caching hierarchy.

Although the values for both measures are less than one order of magnitude above those for modern desktop architectures [NVi11b], it is a common belief in the community that GPUs are generally capable of speedups by 2–3 orders of magnitude [SSG+08, LKC+10]. While this rumour was frequently used as a marketing argument by GPU manufacturers, there are also counter-statements from CPU producing companies. They expect average speedups by a *factor* 2.5 [LKC+10].

The truth is that the parallelisation speedup depends on the structure and size of the problem, as well as on the optimisation state of the CPU-based and GPU-based algorithms to be compared. In the following chapters, we see several GPU algorithms for visual computing tasks. While some of them only yield accelerations by less than a factor 10, others improve the performance on the CPU by more than 100. We are also going to see that different numerical solutions of a problem scale differently on a GPU than on a CPU. Hence, different ideas can lead to an optimal algorithm on a particular architecture. This reduced comparability of approaches complicates the determination of a valid speedup even further.

### 2.1.2 The GPU as a Coprocessor

GPUs are designed as numerical coprocessors in CPU-based desktop computers. Although this concept stems from the traditional role of the graphics card as an auxiliary device, it is today also defined by the layout of this architecture. The high number of parallel resources does not allow each

unit to maintain its individual control flow. Both the die area and the communication overhead would significantly exceed the admissible limits.

Instead, it is required that each element of the vector-based input data can be processed by the same operation simultaneously. Moreover, the control flow must be sufficiently simple. Due to the lack of a call stack, all GPU-programs, called *kernels*, must consist of only one function. This means that function calls including recursion are impossible on a GPU.

However, this limitation is less restrictive as it seems at first glance, because complex control flow patterns can much more efficiently be executed by the sequential CPU. This leads to a programming model in which the coarse-scale program flow is decoupled from the data flow:

- Programs are executed on the **CPU**. This allows to be as flexible as in the fully sequential case. Besides potentially recursive, hierarchic, and highly conditional control flow patterns, this flexibility also involves a free access to all resources such as the command line, the hard disk, and special devices.

- The **GPU** is only employed for specific data-parallel tasks, which are performed very efficiently on this architecture. The CPU explicitly calls each of these GPU kernels, and waits for them to finish. Once this happens, the GPU goes idle until the next kernel is called.

In visual computing, this splitting leads to an intuitive design principle. Given a working CPU-based program, all *image-based* operations are exchanged by efficient GPU kernels. Figuratively spoken, this affects all locations in the program source code in which a 2-D pixel index is gradually incremented, such that some operations are performed on either an image or an intermediate solution. Examples for such cases are the application of a solver to a linear system of equations, point-wise operations on an image, the scaling or translation of images, or the convolution of two images.

Due to this close cooperation of CPU and GPU, we avoid misleading terms such as *GPU algorithm* which are frequently found in the literature. Instead, we refer to *GPU-based* or *GPU-aided* algorithms to underline the heterogeneous nature of this setup. Speedups occurring in such cases are not only related to the high theoretical compute power of GPUs, but also to a smart splitting of the algorithm into two partitions, each of which can efficiently be processed on exactly one of the two architectures.

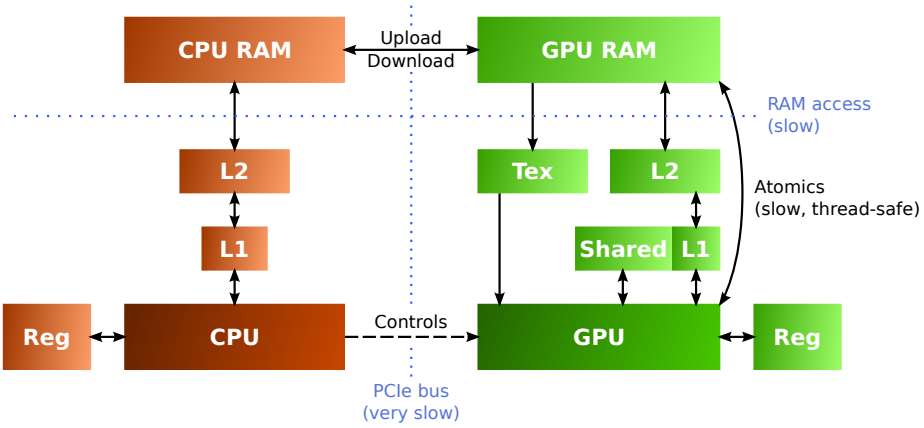



**Figure 2.1**: Simplified model of the CUDA memory hierarchy and host interaction on a recent NVidia graphics card.

### 2.1.3 Memory Model

Because of this separation, the designer of an algorithm must choose appropriate memory locations to store data. Both the CPU and the GPU are equipped with individual physical memories (RAM) and with caches. Figure 2.1 shows an abstract model of the different memories and their connections, assuming a recent graphics device by NVidia. It does not exactly reflect the actual hardware layout, but it serves as a good picture to have in mind when it comes to the different types of memory and their interconnection.

Note that two types of memory access are particularly expensive. Reading data from the RAM always involves significant memory latencies in the order of several hundreds of clock cycles. However, the transfer of data between the RAMs of the CPU and the GPU even takes much longer, usually up to several milliseconds per megabyte. This is because data must be transferred via the PCIe bus which does not provide the necessary bandwidth.

To reduce this transfer time, it is the challenge for the programmer to keep data in one memory as long as possible. Unless the overall memory requirements of a program exceed the physical limits, this paradigm is easy to fulfil. In context of visual computing, we are interested in the processing of large images. Once they are *uploaded* to the graphics RAM, there is usually no need to *download* large data blocks back to CPU memory before the complete program (with all kernels) finishes:

- The **CPU-sided algorithm** is only interested in *control information*

which affects the program flow of the program. Such data includes the dimensions of the image, numerical parameters such as the iteration number or recursion depth, and casual scalar feedback such as a convergence state of the program.

- In contrast, the **GPU** requires all *vector-based* inputs and intermediate solutions. In addition, it casually requires scalar parameters such as individual time step widths, thresholds, the current size of the problem to solve, or the number of iterations to perform.

Scalar parameters can conveniently be transferred *to* the GPU by passing them as arguments to the kernel function. The inverse direction, downloading scalar data from the GPU, works similar to the download of vector-valued data. Note that such a step usually not only involves to *retrieve* this information, but also to *create* a scalar out of large vector-based data. This *reduction* operation is discussed in more detail in Section 2.2.4.

Threads can directly address the GPU RAM, also called *global memory*, and exchange data with it. Since such accesses are not thread-safe, all threads must ensure that their operations do not interfere. Often, this constraint leads to laminar data access patterns where threads in a row read and write subsequent memory cells. As a side effect, such 'coherent' accesses are also more efficient than single reads and writes in a buffer.

On recent graphics cards, this 'standard' type of global memory interaction is buffered by a hierarchy of L1 and L2 caches. Similar to their counterparts on traditional architectures, these caches are mostly hidden to the programmer, but help to reduce read and write latencies. The L1 caching policy can partially by configured from CPU side.

These fully managed caches are complemented by a user-controlled scratchpad, the so-called *shared memory*. Its name is related to the fact that a group of threads share this joint partition of fast on-chip memory. Within such a group of threads that we further detail on in the next section, threads can use shared memory for a quick exchange of information.

In addition to these read-and-write caches, GPUs also offer a hardware managed read-only cache which directly communicates with global memory. This *texture cache* is optimised towards 2-D access, and will be discussed in more detail in Section 2.2.2. In case of a cache hit, requests are similarly fast as if they are issued to shared memory.

Finally, there is also a possibility to enforce thread-safe writes. These *atomic operations* are not very efficient, but sometimes inevitable. We discuss them in more detail in Section 2.2.3.

## 2.2 CUDA Programming Model

As a convenient way to write general-purpose applications for graphics cards, NVidia introduced a high-level programming model and API, called *Compute-Unified Device Architecture (CUDA)* [NVi11b]. While traditional GPGPU programs were required to rewrite a problem as a sequence of graphics operations [Göd05], this new programming model allows to formulate vector- and matrix-based operations in a high-level programming language which extends the C/C++ standards. Similar concepts are also implemented in the *APP* interface by AMD [Adv11], which sets up on the platform-independent programming standard *OpenCL* [Khr10].

Since the graphics card we use in our experiments is produced by NVidia, we formulate all algorithms with respect to the CUDA API. This design promises the best occupancy of the available hardware resources, while it preserves its generality with respect to devices from other vendors: All parallel processing APIs for graphics cards use similar concepts both in software and in hardware, such that the algorithms we write can be ported to other platforms without difficulties.

Maybe even more important than the chosen API is the native instruction set of the used hardware. NVidia distinguishes their different hardware versions by an abstract release number, the *compute capability*, which consists of a combination of a major and a minor index. For our applications, we can often ignore the minor number and focus on the differences in the major index. Besides the aforementioned caches, the 2.x *Fermi* series of NVidia graphics cards also introduces additional features. This allows for runtime improvements, but also for an application to problems that cannot be solved with cards that only support compute capability 1.x. We come back to this detail later in this chapter. For our experiments, we use a modern NVidia GeForce GTX 480 which supports compute capability 2.0.

### 2.2.1 General Concepts

In CUDA, a parallel GPU kernel consists of a grid of independent threads. Although this grid can have either 1, 2, or 3 dimensions, computer vision applications often naturally impose a 2-D thread grid. In such setup, each thread computes the result for exactly one pixel. This *massively parallel* programming concept is visualised in Figure 2.2.

It has a direct influence on the way how CUDA kernels are written. Instead of a domain decomposition into larger chunks such as it is common in CPU multi-threading, a CUDA kernel is conceptually centred around the needs for one arbitrary pixel. This is similar to the single instruction

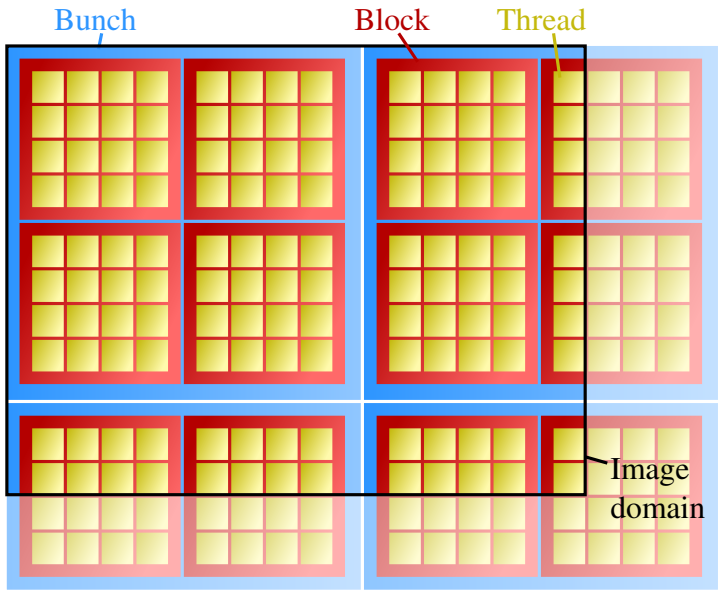



**Figure 2.2**: CUDA thread hierarchy. Threads are organised in blocks, which are again organised in bunches. Each thread handles exactly one pixel from the image domain.

multiple data (SIMD) mechanism of modern CPU architectures. Whenever special cases must be handled, only those threads for which a condition holds are active. All other threads are idle until the special case handling is finished. As a consequence, branching should be avoided wherever this is possible.

A user-specified number of threads is organised in one CUDA *block*. Such blocks are an algorithmic unit which corresponds to a group of threads that are simultaneously hold in memory, and that are executed on one physical core of the device. In general, it is safe to assume that threads of one block are executed in parallel, although they are factually processed in *warps* of 16 threads per clock cycle. The latter detail is important for special optimisations, such as the one described in Section 2.2.4.

Threads within one block share common resources such as the shared memory, and can use this scratchpad to communicate with each other. Beyond the boundary of blocks, communication can only happen via slow global memory operations. This is because different blocks are not necessarily executed at the same time, but can also be scheduled sequentially onto the same core. The decision about which GPU core handles which block is up to the run-time scheduler and cannot be influenced by the programmer. The size of each block can be chosen and optimised manually, but must be

constant per kernel call. It is limited by the physical resources a CUDA core provides. Besides the 48 kB of shared memory on recent devices, this also affects the number of registers.

The number of blocks that are necessary to process a task depends on their size. In general, we must make sure that the discrete domain of the underlying input image is densely tessellated by blocks, where potential overlaps at the image boundaries are tacitly accepted. Regarding these surplus pixels, note that a different treatment of 'inner' and 'outer' threads would introduce significant overheads due to branching. This can be avoided if all pixels are first processed as if they were inside the image domain. Erroneously written values can then efficiently be reset or invalidated in a second step, or during write-back to global memory.

For technical reasons, the CUDA scheduler can only handle a certain device-dependent number of blocks at the same time. This value is always much higher than the number of physical resources, such that a good occupancy can be guaranteed. However, it sometimes does not suffice to tessellate a full input image. In these cases, the same kernel must be started several times with different sets of blocks. We refer to each of this sets as one *bunch*.

This hierarchic structure of CUDA primitives has consequences on the layout of CUDA kernels. The position of a thread in the grid from Figure 2.2 determines the memory offset at which it is supposed to read and write data in global memory. This address must be computed manually, typically as the first step in a kernel. While the thread and block offsets are directly accessible as built-in variables (`threadIdx` and `blockIdx`), the bunch offset must be given as a parameter from CPU side. With these variables, the x coordinate of a thread can be computed as

```
int x = (bunch_x * number_blocks_per_bunch  + blockIdx.x)
                 * number_threads_per_block + threadIdx.x;
```

where the variables `bunch_x`, `number_blocks_per_bunch`, and `number_threads_per_block` are specified by the user. The offset in y direction follows straightforwardly, and the overall offset in linear memory is (in row-first ordering) given by

```
int offset = y * nx + x;
```

where `nx` determines the width of the 2-D buffer in pixels.

### 2.2.2 Textures

A very powerful concept in GPU programming covers so-called *textures*. It stems from the original aim of graphics cards to support hardware-

accelerated rasterisation of 3-D scenes. In such pipelines, textures refer to 2-D bitmaps which are placed onto 3-D objects. To accelerate the retrieval and visualisation of such information, graphics cards offer special circuits and caches. They store 2-D information in a coherent way. On a cache miss, a whole 2-D neighbourhood of the pixel in charge is loaded from global memory. Note that this behaviour is fundamentally different to classical caches, which only support 1-D cache lines.

This performance can also be exploited in CUDA. From a programmer's point of view, a CUDA texture is a special read-only reference to global memory which is automatically cached by the 2-D aware texture caches. In this context, it is not even necessary to access data in the same laminar structure as imposed by Figure 2.2, but it is instead possible to access random values from the whole image domain. These properties make textures a convenient interface to be used in stencil-based operations, as well as in applications which cannot guarantee a laminar memory access.

Moreover, many other features of the original graphics processing pipeline are also supported in CUDA. This includes:

- An optional **interpolation** of texture data. Texture values can be requested between grid points. Depending on the chosen interpolation scheme, either the next neighbour, or a bi-linear interpolation between the surrounding grid points is returned. Since these operations take place in a dedicated circuit, they are as fast as a plain memory read.

- Automatic **boundary handling**. Textures can be requested outside their defined domain. Depending on the configuration, such load can return the closest boundary value, or the value from a periodic continuation of the image. Note that the first type is equivalent to assuming reflecting boundary conditions on a 1-pixel wide stripe. On graphics cards with compute capability 2.x, this set of options is extended by the emulation of a true mirroring with periodic continuation, as well as by the continuation with zeroes. These options correspond exactly to the concepts of Neumann and Dirichlet boundary conditions as they appear in visual computing applications.

- An optional **normalisation** of coordinates. Instead of addressing textures with their true number of grid points, both the width and the height are normalised to 1. This can be interesting for applications in which the underlying sampling rate is not necessarily bound to a predefined grid, such as it is for a convolution kernel. Combined with the bi-linear interpolation from above, the kernel can be dynamically

discretised in an arbitrary precision without the requirement to change the corresponding convolution function.

In the following chapter, we discuss many examples where the 2-D aware caches and the additional features help to speed up an algorithm significantly. This holds in particular if we perform stencil-based operations on the image, or if we process spare elements such as it is shown in Figure 2.2. In both cases, there is no need to pad or mirror the input, as the texturing unit handles these boundary situations automatically.

### 2.2.3 Atomic Operations

As it turns out, the laminar global memory access and the random read by textures does not suffice to efficiently formulate arbitrary algorithms. As a simple counter-example, assume a kernel that is given one address per pixel, and that is supposed to manipulate the addressed pixel in a buffer. The resulting memory pattern describes a random, potentially conflicting write access to single pixels in the image domain. Because such operations usually do not adhere to regular memory patterns, it is very hard to predict and handle conflicts. Nevertheless, such tasks are wide-spread in visual computing, such as for forward warping in context of optic flow [RFSB10], or as one step during the computation of non-equispaced Fourier transforms (see Section 7.6.2 for details).

For such applications, CUDA supports *atomic functions*. They are guaranteed to be thread-safe, and allow the manipulation of randomly addressed memory cells in global memory. While the first atomics were introduced with the compute capability 1.1, the instruction set supported in compute capability 2.0 also includes an atomic accumulation of single precision floating point values. The latter function is the one which we most frequently need in context of our algorithms.

One of the most frequently used operations in visual computing is the *atomic addition*, i.e. to increment the value in a certain memory cell by a user-specified value. Other functions describe the *atomic exchange* of a value, or (in case of integer arguments) the update with the minimum or maximum of the residing value and the given one.

To this end, an programmer often sees atomics as an 'inverse' of a texture. While the latter supports *read* access to random pixels, atomic functions grant *write* access to those locations. However, they are much more expensive and can lead to sequentialisation if a resource is unavailable due to mutual access.

### 2.2.4 Algorithmic Optimisation Techniques

Besides these hardware concepts, there are also a number of algorithmic tricks which are frequently appearing in context of GPU programming. Because these ideas belong to the standard design principles, we do in general not discuss them in detail when they appear in context of our algorithms, but refer to official sources such as the programming manual [NVi11b]. However, because two of these runtime-critical techniques are fundamental to many of our algorithms, we briefly sketch their basic ideas.

#### Optimisation of Shared Memory Access

Shared memory is very fast. If it is addressed in the right way, reads and writes to shared memory are as efficient as the corresponding operations on registers [NVi11b]. In order to understand how we can achieve this performance, we must briefly review the way how this storage is accessed.

For a high bandwidth, shared memory is organised into 16 groups, so-called *banks*. Elements of a vector are always addressed by a round-robin assignment of banks: The first element is addressed by the first bank, the second element by the second, until the 17th element is then again residing in the first bank. If we recall that there is never more than one warp with 16 threads active at a time, 16 banks suffice to handle all shared memory operations simultaneously. However, this holds only if all banks serve only one memory cell at a time. If several threads access different memory cells that all fall into the same bank, these can only be served sequentially and the program slows down. This situation is called a *bank conflict*. Note that it is possible that several threads access the *same* item within one bank. This *broadcast* operation is bank-conflict-free.

#### Reduction

One important concept that suffers particularly from bank conflicts is the *reduction* of data. Because this programming pattern occurs frequently in our algorithms, let us briefly review its basic idea [Har11]. Reduction is necessary when matrix-valued values residing on a grid shall be used to compute a single scalar which must be given back to the CPU. Prominent examples for such values are the convergence state of a problem, the minimum or maximum of a data set, or the sum over all elements. It is clear that a sequential processing of the whole buffer is no option, because this process would in general exceed the runtime requirements. On the other hand, the value can also not be obtained by a fully parallel process, because a joint writing to the same memory cell causes another sequentialisation.

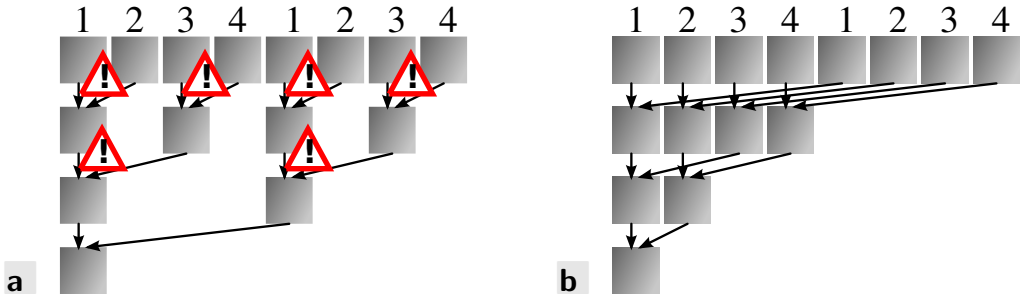



**Figure 2.3**: Reduction, **a.** with bank conflicts, and **b.** without bank conflicts.

Instead, we first perform a parallel reduction of all values within one block and store one value per block back to global memory. This process can be iterated until it ends up with a scalar. For the essential operation, the reduction within the block, we must pay attention to bank conflicts.

Figure 2.3 shows this issue on a simplified example with four banks which are indicated by numbers. Let us assume that each thread first loads the left operand, and in a second step the right one. For the counter-example in part a. where each element is merged with its right neighbour, this leads to bank conflicts: Both on bank 1 and on bank 3, we request two different values at a time, such that this request takes double the time as necessary. The same issue repeats in case of the right operand. Note that the very last merge is conflict-free, because there is only one thread left which is sequential by design.

The planar strategy shown in part b. is bank conflict free, because all threads read from different banks at a given time. As it turns out, this performance can further be boosted by standard concepts such as loop unrolling. These techniques are also applied in our algorithms where possible. A good overview about them is given in [Har11].

## 2.3 Runtime Measurement

When we discuss novel algorithms in the following chapters, we are frequently in the situation that we require reliable runtime benchmarks of our algorithms. Besides principle stumbling blocks that arise from the architecture itself, there are also pitfalls which are mainly related to the special structure of problems as they arise in the field of visual computing.

On traditional CPU hardware, runtime benchmarks are always affected by noise arising from background processes, memory latencies, or cache misses. As it turns out, the same holds for GPU-accelerated algorithms,

even if we make sure that only one program claims the GPU at a time. These effects are even more strikingly visible in benchmarks of GPU-based algorithms than they are on CPUs. This is a result of the higher data throughput and the lower absolute runtimes, where the same absolute deviation in runtime has a higher relative impact. Moreover, there are more sources for runtime deviations than on a CPU:

- The execution order of data-parallel operations on massively parallel hardware is not predetermined. In contrast, it is even the case that each streaming multiprocessor holds a high number of warps whose execution order is dynamically changed whenever one warp idles for an access to global memory [NVi11b].
- Because the overall program flow of GPU-based programs still resides on the CPU, background processes frequently delay threads from being dispatched on the GPU. See Section 2.2 for details.
- The higher speed of GPUs causes a much higher data throughput than on CPUs. Hence, there are more cache misses and potentially higher bursts in access rates to the memory flow controller. Depending on which of the concurrent processes accesses which data at which time, severe memory latencies can be the consequence.
- The graphics driver handles two compute-intensive problems at the same time. While our GPU-based visual computing task is executed on the graphics card, the driver must concurrently schedule graphics operations. Accelerated OpenGL applications compete with our algorithm for the available resources. While we can disable accelerated graphics output for non-interactive programs, this issue becomes important if we consider interactive real-time applications such as the ones described in Sections 5.6 and 6.6.

As a consequence of these considerations, we must pay particular attention to the validity of time measurements. In experiments, a truncated mean turned out to be a very suitable measure [Kre05]: Unless stated differently, we perform 5 independent measurements for every data point, discard the lowest and the highest value, and average over the remaining 3 entries. This system reliably smoothes out obvious outliers. At the same time, it relieves us from the need to identify outliers at hand of special measures, which could lead to a situation in which we wrongly discard meaningful benchmarks and falsify the experiment.

Since all algorithms presented in this thesis cover topics in visual computing, we must also take care of some common anomalies that arise from

the special requirements of such applications. One fundamental property of such algorithms is the multi-dimensionality of data. Memory is always organised in a linear manner, while images were optimally stored in 2-D data structures. As a consequence, neighbouring pixels are only stored in adjacent memory cells if they touch each other in one dimension. Neighbours with respect to the other dimension reside in detached memory cells. This difference is noticeable in the runtime, if an operator can be separated into two 1-D operations. In this case, the operation along the *coherent* memory direction usually requires less time than the operation across, since the first setup can make efficient use of caches while the other frequently struggles with cache misses.

As a consequence, we should be aware that many algorithms behave sensitive to the aspect ratio of the used image. In most cases, we optimise algorithms with respect to squarish images and consequently use such examples for benchmarking purposes. Intuitively, this measure yields 'typical' runtimes for our algorithms, which can increase or decrease for actual problems depending on the aspect ratio. The only exception to this rule is our optic flow algorithm presented in Chapter 6 which is optimised to images in the ratio 4:3. This accounts for the fact that we evaluate it in the established *Middlebury* benchmark which contains images in this ratio.

Moreover, we should pay attention to the fact that some algorithms have favourable runtimes if they are executed on images that possess a particular side length, such as a power of two. Often, this special behaviour is a result of a hierarchic structure of the program, or of the fact that the problem can be better decomposed into CUDA primitives such as blocks and bunches. This leads to two consequences:

1. We should not only test the runtime on power-of-two images, but also perform experiments for other sizes to check whether this affects the runtime of the process.

2. If we find that an algorithm performs badly on non-power-of-two images, we should consider to execute it on the next larger optimal size instead. Often, it is possible to pad the problem with bogus values, compute the solution on the larger image domain, and to discard the solution at surplus positions.

This finishes our brief overview on CUDA. Details about all sketched ideas can be found in the comprehensive programming manual from [NVi11b].

# Chapter 3

# Homogeneous Diffusion

*The universal view melts things into a blur.*
Émile Michel Cioran

## 3.1 Motivation

Many modern methods in the field of visual computing are based on partial differential equations (PDEs). With their help, even challenging problems can easily be modelled and solved in a clean and structured mathematical framework. In this chapter, we focus on the most elementary PDE in visual computing: *Homogeneous Diffusion* [Iij59, Wit83].

Despite – or maybe because – of its simplicity, homogeneous diffusion is omnipresent in image processing and computer vision. Its good low-pass filtering properties make it a standard operation for the smoothing or denoising of images [KZ96]. This is particularly important as a preprocessing step for operations which require a certain smoothness or differentiability of the input data. Popular applications are the computation of the structure tensor [FG87], or anti-aliasing for the generation of mipmaps [BA83].

Homogeneous diffusion constitutes the basis of the Gaussian scale-space theory [Iij59, Wit83, Lin94, SNFJ97, Flo97]. Such scale spaces led to the development of scale-invariant filters [Wit83], and are today indispensable for the detection of edges [MH80, Can86] or image features [FG87, Low04]. Related to those scale-selection approaches are also filters which steer their evolution by scale spaces, such as diffusion-thresholding methods for the computation of mean curvature motion [MBO92].

Often, the operand smoothed by homogeneous diffusion is not even an image in the classical sense. In context of optic flow computation, it is used to robustify the results by smoothing away outliers in the estimation [LK81]. Even more importantly, homogeneous diffusion is frequently used to regularise variational methods in visual computing [HS81, WB02].

The applicability of homogeneous diffusion goes far beyond visual computing. Some approaches, such as unsupervised learning [Par62], are used in many fields in computer science including computer vision [RBK98, FPZ03]. Other techniques are among the standard tools in closely related fields such as audio processing. Prominent examples are the chirp spread spectrum technique [Pra98] or Gaussian minimum shift keying [EVB01].

All these approaches have in common that they require homogeneous diffusion filtering on potentially large datasets. This task may become very time consuming, such that the need for efficient algorithms for this purpose is obvious. At this point, we can try to use graphics cards for an additional boost in performance. Such acceleration offers to process even larger or higher resolved images under the same given time constraints.

The search for fast GPU-based homogeneous diffusion algorithms is the subject of this chapter. We review many different numerical approaches from the literature, and complement them by novel promising techniques. For all schemes, we discuss ways to implement them efficiently on GPUs, and evaluate the outcome with respect to their quality-speed tradeoffs.

## 3.2 Introduction to Homogeneous Diffusion

Let us begin with a short review of the classical idea of homogeneous diffusion in image processing [Iij59, Wit83]. It is borrowed from physics, where *diffusion* describes a mass-preserving equilibration procedure of concentrations over time. Fick's law leads to the general *diffusion equation* [Wei98]

$$\partial_t u = \operatorname{div}(\boldsymbol{D}\ \boldsymbol{\nabla} u)\,, \tag{3.1}$$

where $u$ denotes the concentration, $t$ represents time, $\boldsymbol{\nabla} u$ is the gradient of the concentration, and 'div' denotes the divergence operator. The *diffusion tensor* $\boldsymbol{D}$, a positive definite symmetric matrix, steers the type of diffusion. In this chapter, we focus on the simplest case in which mass is distributed homogeneously into all directions. This corresponds to $\boldsymbol{D}$ being chosen as the identity matrix. Later in Chapters 4 and 5, we also see more advanced choices for $\boldsymbol{D}$.

In visual computing, we associate the concentration $u$ with a continuous image $u : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$ denotes a rectangular image domain. Grey

values play the role of mass, and the mass-preserving property of diffusion carries over to a preservation of brightness in the image.

On images, homogeneous diffusion behaves as a low-pass filter. With increasing time $t$, the image is gradually blurred until, for $t \to \infty$, the process yields a uniform image with the same average grey value as the original. For most applications, we are thus interested to interrupt the process at a certain *stopping time* $T$. The corresponding result $u(\boldsymbol{x}, T) : \Omega \times \mathbb{R}^+ \to \mathbb{R}$ can be obtained as a solution of the *linear diffusion equation* (or *heat equation*) at time $T$:

$$\partial_t u = \Delta u \qquad \text{on } \Omega \times \mathbb{R}^+ , \tag{3.2}$$

with the initial state

$$u(\boldsymbol{x}, 0) = f(\boldsymbol{x}) \qquad \text{on } \Omega , \tag{3.3}$$

and reflecting (Neumann) boundary conditions

$$\langle \boldsymbol{\nabla} u, \boldsymbol{n} \rangle = 0 \qquad \text{on } \partial\Omega \times \mathbb{R}^+ . \tag{3.4}$$

In this context, $\boldsymbol{n}$ represents the normal vector across the boundary of the image domain $\Omega$. On this rectangular image domain, $\boldsymbol{n} = (\boldsymbol{n}_x, \boldsymbol{n}_y)^\top$ simply corresponds to the unit vectors $\boldsymbol{n}_x = (1, 0)^\top$, $\boldsymbol{n}_y = (0, 1)^\top$. Moreover, the set $\mathbb{R}^+ = (0, \infty)$ contains the real positive numbers, and $\langle \cdot \rangle$ denotes the Euklidean inner product on $\mathbb{R}^2$ [Wei98]. On the following pages, we discuss suitable ways to solve the heat equation (3.2) on a spatially discrete domain.

## 3.3 Classical Numerics

In the literature, the problem of linear diffusion is approached in many different ways which have advantages with respect to various aspects. They are designed to yield a very high accuracy or a low runtime complexity, and are often particularly efficient under special conditions such as a certain range of standard deviations. Let us now see some prominent representatives for these classes of algorithms. In Section 3.5, we then use these insights to develop fast parallel GPU-based algorithms for these approaches.

### 3.3.1 Explicit Linear Diffusion

A classical approach to linear diffusion is an explicit discretisation of the heat equation (3.2) in time:

$$\frac{\boldsymbol{u}^{k+1} - \boldsymbol{u}^k}{\tau} = \boldsymbol{A}\, \boldsymbol{u}^k , \tag{3.5}$$

where $\tau$ denotes an artificial time step size, $\boldsymbol{u}^k$ is the discrete vector-valued representation of $u$ at time $\tau k$ and $\boldsymbol{A}$ is a 'suitable' spatial discretisation of the Laplacian operator. For our convenience, we index $\boldsymbol{u}$ in the following with two indices $i \in [0, \dots, n_x]$ and $j \in [0, \dots, n_y]$ to denote the spatial location of a pixel in the image domain. Nevertheless, $\boldsymbol{u}$ is technically a 1-D vector, whose canonical index $\kappa$ can be computed as $\kappa = j \cdot n_x + i$. In Section 2.3, we have already discussed this layout in context of runtime bottlenecks for efficient GPU programs.

By rewriting (3.5), we obtain the iterative *explicit scheme*

$$\boldsymbol{u}^{k+1} = (\boldsymbol{I} + \tau \boldsymbol{A})\, \boldsymbol{u}^k \,. \tag{3.6}$$

For the 2-D Laplacian, one often takes second-order finite differences [Smi04]:

$$\boldsymbol{A}\boldsymbol{u} = [\Delta u]_{i,j} = \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right]_{i,j} \tag{3.7}$$

$$\approx \frac{\boldsymbol{u}_{i-1,j} - 2\boldsymbol{u}_{i,j} + \boldsymbol{u}_{i+1,j}}{h_x^2} + \frac{\boldsymbol{u}_{i,j-1} - 2\boldsymbol{u}_{i,j} + \boldsymbol{u}_{i,j+1}}{h_y^2} \,, \tag{3.8}$$

where $h_x$ and $h_y$ denote the spatial grid distances, and $i, j$ are spatial indices. By $[\cdot]_{\boldsymbol{p}}$, we indicate a discretisation of the argument at grid point $\boldsymbol{p}$. In the following, we assume a squarish grid, such that $h_x = h_y =: h$. For the explicit scheme from (3.6), this leads to

$$\boldsymbol{u}_{i,j}^{k+1} = \boldsymbol{u}_{i,j}^k + \frac{\tau}{h^2} \sum_{(m,n)\,\in\,\mathcal{N}_4(i,j)} \left(\boldsymbol{u}_{m,n}^k - \boldsymbol{u}_{i,j}^k\right) \,, \tag{3.9}$$

where $\mathcal{N}_4(i,j)$ is the 4-neighbourhood around the pixel $(i,j)$:

$$\mathcal{N}_4(i,j) = \Big\{(i-1,j),\, (i+1,j),\, (i,j-1),\, (i,j+1)\Big\} \,. \tag{3.10}$$

In Section 3.5, we design a parallel GPU-based implementation of this algorithm.

Let us now briefly discuss the selection of the virtual time step size $\tau$. Since we require the process to be stable, we must ensure that all weights of the stencil from (3.9) are non-negative. For the main diagonal of the 2-D operator $(\boldsymbol{I} + \tau \boldsymbol{A})$, this leads to the condition

$$\tau \leqslant \frac{h^2}{4} \,, \tag{3.11}$$

while the off-diagonals are by construction positive. For an optimal runtime, we want $\tau$ to be as large as possible, since the number of updates $k_{\max}$ is computed as

$$k_{\max} = \frac{T}{\tau}\,, \tag{3.12}$$

where $T$ denotes again the stopping time of the process. If we assume that $\tau$ is set constant, we obtain a runtime complexity of $\mathcal{O}(Tn)$, where $n$ denotes the number of pixels in the discrete image domain.

As it turns out, however, the choice $\tau = {}^{h^2}\!/_4$ does in general not yield the desired solution as well. In this case, the main diagonal of the operator $(\boldsymbol{I} + \tau \boldsymbol{A})$ vanishes, and this matrix looses its irreducibility. We obtain two decoupled problems on an alternating 'checkerboard' domain, and high-frequent errors cannot be smoothed out by the process.

If we want an optimal variation-diminishing behaviour of the process, we must ensure that it is sign-stable [GK80]. By Gershgorin's Theorem, $\boldsymbol{A}$ possesses eigenvalues in the interval $[-{}^{8}\!/_{h^2}, 0]$. Hence, we must ensure that $1 - {}^{8\tau}\!/_{h^2} \geqslant 0$, which yields a sign-stability of the process for

$$\tau \leqslant \frac{h^2}{8}\,. \tag{3.13}$$

As a consequence from these considerations, we use two different parametrisations for our experiments in Section 3.6.3. Assuming a unit grid, i.e. $h = 1$, we use $\tau = 0.125$ as the fastest process with a good error-dampening behaviour. Besides this quality-optimised setting, we also evaluate an algorithm with $\tau = 0.24$. The latter setup is optimised for a very low runtime, but preserves the irreducibility of the process.

### 3.3.2 Implicit Linear Diffusion

An alternative to the aforementioned explicit scheme is given by an implicit discretisation of the heat equation (3.2) in time [Gou85]:

$$\frac{\boldsymbol{u}^{k+1} - \boldsymbol{u}^k}{\tau} = \boldsymbol{A}\,\boldsymbol{u}^{k+1}\,. \tag{3.14}$$

This leads to an *implicit scheme* in which the linear operator is applied to the left term:

$$(\boldsymbol{I} - \tau \boldsymbol{A})\,\boldsymbol{u}^{k+1} = \boldsymbol{u}^k\,. \tag{3.15}$$

Although the system is stable under arbitrary time step sizes $\tau$, its solution still requires a significant workload. The reason for this is that an inversion of $(\boldsymbol{I} - \tau \boldsymbol{A})$ in general leads to a dense matrix which requires an immense

effort in storage and compute power. Instead, one usually solves such linear systems of equations numerically. On traditional hardware, 1-D problems of this kind are often solved by a Thomas algorithm which yields the solution of one step in only two sweeps. However, since this algorithm is *per se* sequential and we aim at an efficient formulation for GPUs, we use an iterative solver instead. A prominent class of algorithms for this purpose is given by splitting methods. Due to its data-parallel structure, one member of this class, the *Jacobi scheme*, is particularly interesting for application on massively parallel hardware. Rewriting (3.15) in terms of the Jacobi scheme leads to [Mei05]:

$$\boldsymbol{u}^{k+1,l+1} = \boldsymbol{D}^{-1}(\boldsymbol{D} - (\boldsymbol{I} - \tau\boldsymbol{A}))\,\boldsymbol{u}^{k+1,l} + \boldsymbol{D}^{-1}\boldsymbol{u}^{k}\,, \tag{3.16}$$

where $\boldsymbol{D}$ denotes the diagonal part of $(\boldsymbol{I} - \tau\boldsymbol{A})$. The new index $l \in \{1, \dots, l_{\max}\}$ describes the step within the Jacobi solver. If we again use second order finite differences for $\boldsymbol{A}$, we obtain

$$\boldsymbol{u}_{i,j}^{k+1,l+1} = \frac{h^2}{h^2 + 4\tau} \left( \boldsymbol{u}_{i,j}^{k} + \frac{\tau}{h^2} \sum_{(m,n)\,\in\,\mathcal{N}_4(i,j)} \boldsymbol{u}_{m,n}^{k+1,l} \right). \tag{3.17}$$

The choice of the number of outer and inner iterations $k_{\max}$ and $l_{\max}$ represents a tradeoff between accuracy and runtime. Very few outer iterations cause a higher approximation error, while few inner iterations might not suffice to let the linear solver converge to the desired solution. On the other hand, choosing both values large leads to an undesirably high runtime. Generally, we can assume that $k_{\max} \cdot l_{\max} \sim T$, such that the process possesses a runtime complexity of $\mathcal{O}(Tn)$. In Section 3.6.2, we search a 'good' tradeoff between $k_{\max}$ and $l_{\max}$, and compare the resulting method to the explicit scheme, as well as to other numerical alternatives which are to be presented in the following paragraphs.

### 3.3.3 Spatial Convolution

In a continuous setting, it is well-known that the analytical solution of the heat equation at stopping time $T$ is given by a convolution with a Gaussian kernel [AU10]

$$\boldsymbol{u}(\boldsymbol{x}, T) = K_{\sqrt{2T}} * \boldsymbol{u}(\boldsymbol{x}, 0) \tag{3.18}$$

with standard deviation

$$\sigma = \sqrt{2T}\,. \tag{3.19}$$

This equivalence leads to a number of very efficient approximations for linear diffusion. On the following pages, we discuss representatives for the most prominent classes of such algorithms.

As a general remark to all of these solutions, note that the continuous equivalence does not exactly carry over to the discrete case. Due to discretisation artefacts, the space spanned by the arising discrete operator does often not enjoy scale space properties [Lin90]. However, we see in Section 3.6.1 that such principal numerical errors are often below the accuracy of single precision floating point arithmetics. This holds in particular for large stopping times (or standard deviations of the kernel).

As a first, straightforward approach to realise Gaussian convolution on a discrete domain, let us sample the Gaussian kernel and perform a discrete convolution with the signal. In $n$ dimensions, this Gaussian filter is given as follows.

**Definition 3.1 (Gaussian filter)**

*A* Gaussian filter *is a convolution*

$$(K_\sigma * f)(\boldsymbol{x}) := \int_\Omega K_\sigma(\boldsymbol{y}) f(\boldsymbol{x} - \boldsymbol{y}) \mathrm{d}\boldsymbol{y} \tag{3.20}$$

*of an image $f$ with a* Gaussian kernel

$$K_\sigma(\boldsymbol{x}) := \frac{1}{(\sigma\sqrt{2\pi})^n} \, e^{-\frac{|\boldsymbol{x}|^2}{2\cdot\sigma^2}} \tag{3.21}$$

*of standard deviation $\sigma$ and $\boldsymbol{x} \in \mathbb{R}^n$.*

□

An important property which becomes immediately visible from the definition is the separability of the kernel in space. Equivalent to the convolution with an $n$-D Gaussian, a signal can successively be convolved with a 1-D Gaussian in each direction. This is particularly important when the kernel becomes very large, as it helps significantly to obtain a reasonable runtime.

In the following, we review some algorithms for linear diffusion which employ the equivalence to Gaussian filtering. Unless otherwise stated, they all use the separability as one important step towards a small runtime. In Section 3.5, we then discuss the actual implementational details for fast GPU-based algorithms.

Our first approach to Gaussian convolution is straightforward: We convolve the discrete signal with a discretised 1-D Gaussian kernel which has

been truncated at $c\frac{\sigma}{h}$ samples:

$$\widehat{[K_\sigma]}^c_{hj} = \begin{cases} \frac{h}{\sigma\sqrt{2\pi}}\, e^{-\frac{j^2h^2}{2\cdot\sigma^2}}\,, & |hj| \leqslant c\sigma \\ 0\,, & \text{else} \end{cases} \tag{3.22}$$

Because of truncation errors, $\widehat{[K_\sigma]}^c$ does not share the unit sum property with its continuous counterpart. Hence, we normalise it in order to obtain a mass-preserving convolution kernel:

$$[K_\sigma]^c_{hj} = \frac{\widehat{[K_\sigma]}^c_{hj}}{\sum_i \widehat{[K_\sigma]}^c_{hi}} \tag{3.23}$$

Using this discrete kernel, the previously continuous convolution operation from (3.20) turns into a sum:

$$([K_\sigma]^c * \boldsymbol{f})_{hi} = \sum_j \left( \boldsymbol{f}_{hi-hj} \cdot [K_\sigma]^c_{hj} \right) . \tag{3.24}$$

Finally, let us remark some properties of this operation. The kernel cut-off $c$ is motivated by the observation that the Gaussian kernel drops off rapidly towards $\pm\infty$. This guarantees a high accuracy, even for $c$ being chosen small. By setting $c = 3$, a portion of less than $3 \cdot 10^{-3}$ of the grey values of the kernel are lost. For $c = 4$ and $c = 5$, this bound even drops to $1{\cdot}10^{-4}$ and $1{\cdot}10^{-6}$, respectively. Because the loss in mass is already compensated by the normalisation in (3.23), these deviations only represent an abstract notion of an approximation error of the scheme. Note that since this error can be effectively dampened for large $c$, convolution with a truncated Gaussian is also a good candidate to generate reliable ground truths to compare other methods against.

Moreover, this method is also very efficient in runtime when it comes to small standard deviations. By (3.24) and the separability of the process, as well as assuming $c$ to be constant, the algorithm has a runtime complexity of $\mathcal{O}(\sigma n)$. As before, $n$ denotes the number of pixels in the image. Since $\sigma \sim \sqrt{T}$ (see (3.19)), we can equivalently state that discrete convolution with a Gaussian possesses a complexity of $\mathcal{O}(\sqrt{T}n)$. This corresponds more closely to our understanding of Gaussian filtering is an alternative algorithm for linear diffusion. Note that the actual runtime can be additionally reduced if we take the symmetry of the kernel into account. This allows to compute and store only about half of the kernel samples. Moreover, (3.23) ensures a unit sum over the (finite) kernel, such that the normalisation weight $\frac{h}{\sigma\sqrt{2\pi}}$ from (3.22) can be omitted without loss of generality.

However, the runtime complexity of truncated spatial convolution is also its main counter-argument for large standard deviations, i.e. linear diffusion with reasonably high stopping times. In such cases, the size of the kernel lies in the order of magnitude of the length of the signal, and the runtime class of $\mathcal{O}(n\sigma)$ effectively comes down to $\mathcal{O}(n^2)$.

### 3.3.4 Multiplication in the Frequency Domain

Frequency-based convolution techniques are well suited for Gaussian filtering with large standard deviations, as they possess a kernel-independent runtime complexity of $\mathcal{O}(n \log n)$. The motivation for such methods is given by the convolution theorem. In a continuous sense, it states that a convolution of two functions in the spatial domain is equivalent to a multiplication of the duals of these functions in the frequency domain [Bra99]:

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)\,. \tag{3.25}$$

$\mathcal{F}$ denotes the Fourier transform

$$\mathcal{F}(f)(\boldsymbol{\xi}) = \int_{\mathbb{R}^n} f(\boldsymbol{x})\, e^{-2\,\pi\, i\, \langle \boldsymbol{x}, \boldsymbol{\xi} \rangle} \mathrm{d}\boldsymbol{x}\,, \tag{3.26}$$

with $\langle \cdot \rangle$ denoting the inner product. In the concrete setting of Gaussian filtering, we can thus write [GW08]

$$K_\sigma * u = \mathcal{F}^{-1}\left(\mathcal{F}(K_\sigma) \bullet \mathcal{F}(u)\right)\,, \tag{3.27}$$

where $\bullet$ denotes a point-wise product, and where the inverse Fourier transform is given by

$$\mathcal{F}^{-1}(\hat{f})(\boldsymbol{x}) = \int_{\mathbb{R}^n} \hat{f}(\boldsymbol{\xi})\, e^{2\,\pi\, i\, \langle \boldsymbol{x}, \boldsymbol{\xi} \rangle} \mathrm{d}\boldsymbol{\xi}\,. \tag{3.28}$$

By using (3.27), the time required for Gaussian convolution no longer depends on the standard deviation of the kernel. On discrete signals, $\mathcal{F}$ and $\mathcal{F}^{-1}$ can be computed by efficient fast Fourier transforms (FFTs) with a runtime complexity of $\mathcal{O}(n \log n)$ [Bri88]. This is the reason why this technique belongs to the standard approaches for an efficient implementation of Gaussian filtering [GW08]. Moreover, it is often beneficial to perform $n$-D FFTs instead of using the separability of the Gaussian. The reason for this is that although the number of operations for the transformations are in both cases similar, caches can be used more efficiently if all directions are processed at once. Moreover, the multiplication in the Fourier domain must

be performed once per pair of transformations, such that it is again advantageous to compute all directions at once, given the problem can suitably be represented in physical memory.

As can be seen by simple computations, it is not necessary to transform the Gaussian to obtain its dual. Instead, $\mathcal{F}(K_\sigma)$ can immediately be computed by (3.26) and (3.21):

$$\mathcal{F}(K_\sigma)(\boldsymbol{\xi}) = \mathcal{F}\left(\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{|\boldsymbol{x}|^2}{2\cdot\sigma^2}}\right)(\boldsymbol{\xi}) = e^{-2\pi^2\sigma^2|\boldsymbol{\xi}|^2}\,. \tag{3.29}$$

Depending on the architecture, using separability to set up the dual of the kernel can be beneficial with respect to the required runtime.

As a final remark, note that the discrete Fourier transform assumes signals to be periodic. A naive implementation of Gaussian filtering by the convolution theorem causes wrap-around errors, i.e. grey values are transported over the image boundaries back into the opposite side of the image. Since we assume reflecting boundary conditions, the signal can be made periodic by extending it with its mirrored version [But05]. Compared to other approaches to obtain periodicity, such mirroring does not introduce new artificial frequencies, but doubles the workload. In Section 3.6.3, we are going to see that this modified approach yields results in a comparable quality to spatial convolution, but also that its linear-logarithmic runtime complexity only pays off for large standard deviations. In the literature, the break-even point between these two methods is typically given for kernels truncated to about 32 samples [Bri88].

### 3.3.5 Recursive Filtering

Both aforementioned methods yield very accurate results, but they are also very time demanding. *Recursive filters* try to remedy these drawbacks by combining the advantages of both worlds [Der87a, YvV95, Hal06b]. They design a filter in the frequency domain, but apply it in the spatial domain. By using already computed samples from the result as an input for the computation of unprocessed samples, recursive filters have an infinite impulse response. Hence, they can obtain an accuracy comparable to convolution in the frequency domain, while the computation involves only very few additions and multiplications per sample.

In order to develop an efficient GPU-based filter for Gaussian convolution, let us briefly sketch the basic idea without going into detail. To this end, we summarise the concept of a 'parallel' application of the causal and anti-causal parts of the Young-van-Vliet filter [YvV95] as suggested by

Hale [Hal06b]. This instance of recursive filters is reported to yield the best results for large standard deviations, in particular on finite signals [Hal06b].

Let us start with a short introduction into the classical form of the Young-van-Vliet filter [YvV95]. To this end, we take the Fourier transform of a 1-D Gaussian similar to (3.29):

$$\mathcal{F}(K_\sigma)(\xi) = \mathcal{F}\left(\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{x^2}{2\cdot\sigma^2}}\right)(\xi) \tag{3.30}$$

$$= e^{-\frac{\sigma^2\omega^2}{2}}\,, \tag{3.31}$$

where $\omega = 2\pi\xi$. By a Taylor expansion, $\mathcal{F}(K_\sigma)(\xi)$ can be approximated by an expression of the form

$$\mathcal{F}(K_\sigma)(\xi) \approx G_q(\omega) = \frac{n_0}{c_0 + c_1(q\omega)^2 + c_2(q\omega)^4 + c_3(q\omega)^6}\,, \tag{3.32}$$

where $c_0$, $c_1$, $c_2$, and $c_3$ are constants, $n_0$ is a normalisation parameter that allows to have a unit integral over the kernel, and $q \approx \sigma$. While the values of the constants can be immediately computed by solving a linear system of equations (cf. [YvV95]), $q$ is usually optimised experimentally such that the impulse response of the filter to be constructed corresponds best to the one of the Gaussian filter to approximate. Alternatively, one can obtain a less accurate guess for $q$ from an Levenberg-Marquardt estimate of the relation between $q$ and $\sigma$ that is given in [YvV95].

Apart from scaling, the Fourier transform is a special case of the *Laplace transform* where the real part of the argument set to zero, i.e. $s = i\omega$ with $i$ denoting the imaginary unit:

$$\mathcal{F}(f)(\omega) = \int_{-\infty}^{\infty} f(x)e^{-i\omega x}\mathrm{d}x = \mathcal{L}(f)(s)|_{s=i\omega} \tag{3.33}$$

This idea motivates to regard $G_q(\omega)$ from (3.32) in the Laplace domain:

$$G_q(s) = \frac{n_0}{c_0 - (c_1\, q^2)s^2 + (c_2\, q^4)s^4 - (c_3\, q^6)\, s^6}\,. \tag{3.34}$$

$G_q(s)$ can now be factored into two terms

$$G_q(s) = G_{q,L}(s)G_{q,R}(s), \tag{3.35}$$

where $G_{q,L}$ contains all poles that lie in the left half-plane, and $G_{q,R}$ contains all poles from the right half-plane. Again without going into detail about

the actual values for the constants involved in this process, we obtain two equations of the form:

$$G_{q,L}(s) = \frac{n_1}{(\alpha_1 + qs)(\alpha_2 + \alpha_3\, qs + q^2\, s^2)} \tag{3.36}$$

$$G_{q,R}(s) = \frac{n_1}{(\alpha_1 - qs)(\alpha_2 - \alpha_3\, qs + q^2\, s^2)} \tag{3.37}$$

with $\alpha_1$, $\alpha_2$, and $\alpha_3$ being constants, and $n_1$ serving for normalisation.

The authors of [YvV95] continue with a discretisation of these equations. For a uniform sampling on a grid with spacing $h$, the Laplace domain equals the (discrete) Z domain. They use backward differences $s = {}^{(1-z^{-1})}/_{h}$ for $G_{q,L}$, and forward differences $s = {}^{(z-1)}/_{h}$ for $G_{q,R}$. The fact that they apply different discretisations for both cases accounts for the causal structure of $G_{q,L}$ and the anti-causal structure of $G_{q,R}$. Assuming $h = 1$, this leads to a causal system

$$H_L(z) = \frac{n_2}{\beta_1 + \beta_2\, z^{-1} + \beta_3\, z^{-2} + \beta_4\, z^{-3}} \tag{3.38}$$

and an anti-causal system

$$H_R(z) = \frac{n_2}{\beta_1 + \beta_2\, z^{1} + \beta_3\, z^{2} + \beta_4\, z^{3}}\,, \tag{3.39}$$

where $\beta_1$, $\beta_2$, $\beta_3$, and $\beta 4$ are third-degree polynomials in $q$ [YvV95]. Recall that by (3.35), the product of $G_{q,L}(s)$ and $G_{q,R}(s)$ gives the approximation of the Gaussian kernel $\mathcal{F}(K_\sigma)(\xi)$ we are looking for. Hence, the recursive filter is expressed in the Z domain as

$$H(z) = H_L(z) \cdot H_R(z)\,. \tag{3.40}$$

Since a multiplication in the Z domain corresponds to a convolution in the spatial domain, the filtering result is obtained by a 'forward' filtering with the causal system, followed by a 'backward' sweep with the anti-causal system [YvV95].

As it turns out, this strategy bears a fundamental problem. Both systems $H_L(z)$ and $H_R(z)$ provide an infinite impulse response. If we consider an arbitrary finite signal, this means that the intermediate signal obtained from the forward run is unbounded to one side. Moreover, we are not allowed to crop this intermediate signal to the length of the input, because the anti-causal filter $H_R(z)$ requires an accurate representation of this region in order to yield the correct output. A remarkable example for the severity of this issue is shown in [Hal06b]. If we intend to use the filter in its classical

form on real data, we thus require a large boundary at one side of the signal in which we store sufficiently many intermediate values to reduce the error in the backward sweep to the minimum.

In [TS06], the authors suggest to truncate the intermediate signal to the domain of the input, and to remedy the bad initialisation of the backwards sweep by a sophisticated heuristics. To this end, they analyse the error that is induced by a truncation, and propose to correct the coefficients of the backwards sweep such that it suppresses these errors in the output. Although this approach allows to compute the filtering result without additional overhead in storage, some artefacts can still not be removed. This can also be seen in the original paper (cf. [TS06, Fig. 1]).

An alternative remedy is motivated from the observation that the problem at the right boundary is inherent to the Young-van-Vliet filter, but does not occur for the first recursive implementation of a Gaussian filter by Deriche [Der87a]. This is because the Deriche filter combines a causal and an anti-causal filter by a sum rather than by a product. In this 'parallel' implementation style, both filter components are executed on the same input signal, and their outputs are summed up to obtain the overall filter output. As a consequence, Hale suggests a 'parallel' implementation style for the Young-van-Vliet filter [Hal06b]. This yields a method which fuses the better approximation quality of the Young-van-Vliet filter, in particular for large standard deviations, with the Deriche filter's better behaviour at boundaries. By the partial fraction method, he rewrites (3.40) as a sum

$$H(z) = \tilde{H}_L(z) + \tilde{H}_R(z)\,. \tag{3.41}$$

Using one additional expansion, $\tilde{H}_L$ and $\tilde{H}_R$ are simplified to sums of second-order filters:

$$\tilde{H}_L(z) = \frac{b_{00}^{+} + b_{10}^{+}\,z^{-1}}{a_{10}\,z^{-1} + a_{20}\,z^{-2}} + \frac{b_{01}^{+} + b_{11}^{+}\,z^{-1}}{a_{11}\,z^{-1} + a_{21}\,z^{-2}}\,, \tag{3.42}$$

$$\tilde{H}_R(z) = \frac{b_{10}^{-}\,z^{1} + b_{20}^{-}\,z^{2}}{a_{10}\,z^{1} + a_{20}\,z^{2}} + \frac{b_{11}^{-}\,z^{1} + b_{21}^{-}\,z^{2}}{a_{11}\,z^{1} + a_{21}\,z^{2}}\,, \tag{3.43}$$

where the coefficients can be immediately computed from the (complex) poles from (3.38) and (3.39) [Hal06a]: After the scaling factor $q$ has been determined using any of the methods described above, the polar coordinates of the poles

$$1.12075 + 1.27788\,i \qquad \text{and} \tag{3.44}$$

$$1.76952 + 0.46611\,i \tag{3.45}$$

are adjusted by taking their radius to the power of $2/q$, and by multiplying their angle by $2/q$. The resulting new poles $d_0$ and $d_1$ are the poles of the Young-van-Vliet filter for the selected standard deviation $\sigma$. From them, the values for $a_{10}$, $a_{11}$, $a_{20}$, and $a_{21}$ can directly be computed as

$$a_{10} = -2 \cdot \Re(d_0)\,, \qquad a_{11} = -2 \cdot \Re(d_1)\,, \tag{3.46}$$

$$a_{20} = |d_0|^2\,, \qquad a_{21} = |d_1|^2\,. \tag{3.47}$$

The coefficients $b_{00}^+$, $b_{01}^+$, $b_{10}^+$, and $b_{11}^+$ for the causal filter parts are related to the zeroes of the filter. Using that the gain of the filter is given by

$$g = \Re((1 - d_0) \cdot (1 - \overline{d_0}) \cdot (1 - d_1) \cdot (1 - \overline{d_1}))\,, \tag{3.48}$$

the residues for the first and the second pole can be computed by

$$g_0 = \frac{g^2}{(1 - \frac{d_1}{d_0}) \cdot (1 - \frac{\overline{d_1}}{d_0}) \cdot (1 - d_0^2) \cdot (1 - \overline{d_0}\, d_0) \cdot (1 - d_1\, d_0) \cdot (1 - \overline{d_1}\, d_0)} \tag{3.49}$$

$$g_1 = \frac{g^2}{(1 - \frac{d_0}{d_1}) \cdot (1 - \frac{\overline{d_0}}{d_1}) \cdot (1 - d_0\, d_1) \cdot (1 - \overline{d_0}\, d_1) \cdot (1 - d_1^2) \cdot (1 - \overline{d_1}\, d_1)} \tag{3.50}$$

from which the coefficients follow as

$$b_{00}^+ = \frac{\Im(\frac{\overline{g_0}}{d_0})}{\Im(\frac{1}{d_0})}\,, \qquad b_{01}^+ = \frac{\Im(\frac{\overline{g_1}}{d_1})}{\Im(\frac{1}{d_1})}\,, \tag{3.51}$$

$$b_{10}^+ = \frac{\Im(g_0)}{\Im(\frac{1}{d_0})}\,, \qquad b_{11}^+ = \frac{\Im(g_1)}{\Im(\frac{1}{d_1})}\,. \tag{3.52}$$

In this context, an overline $\bar{\cdot}$ denotes the complex conjugate, and $\Re$ and $\Im$ denote real and imaginary parts, respectively. Finally, because the anti-causal filters must be symmetric to the causal ones, $b_{10}^-$, $b_{11}^-$, $b_{20}^-$, and $b_{21}^-$ can be computed as

$$b_{10}^- = b_{10}^+ - b_{00}^+ \cdot a_{10}\,, \qquad b_{11}^- = b_{11}^+ - b_{01}^+ \cdot a_{11}\,, \tag{3.53}$$

$$b_{20}^- = \quad - b_{00}^+ \cdot a_{20}\,, \qquad b_{21}^- = \quad - b_{01}^+ \cdot a_{21}\,. \tag{3.54}$$

If we apply the filters given by (3.42) and (3.43) to a 1-D signal $\boldsymbol{f}$, we obtain two causal (forwards) filtering steps

$$\boldsymbol{y}_i^{(1)} = b_{00}^+ \cdot \boldsymbol{f}_i \quad + b_{10}^+ \cdot \boldsymbol{f}_{i-1} \; - a_{10} \cdot \boldsymbol{y}_{i-1}^{(1)} \; - a_{20} \cdot \boldsymbol{y}_{i-2}^{(1)}\,, \tag{3.55}$$

$$\boldsymbol{y}_i^{(2)} = b_{01}^+ \cdot \boldsymbol{f}_i \quad + b_{11}^+ \cdot \boldsymbol{f}_{i-1} \; - a_{11} \cdot \boldsymbol{y}_{i-1}^{(2)} \; - a_{21} \cdot \boldsymbol{y}_{i-2}^{(2)}\,, \tag{3.56}$$

and two anti-causal (backwards) filtering steps

$$\boldsymbol{y}_i^{(3)} = b_{10}^- \cdot \boldsymbol{f}_{i+1} + b_{20}^- \cdot \boldsymbol{f}_{i+2} - a_{10} \cdot \boldsymbol{y}_{i+1}^{(3)} - a_{20} \cdot \boldsymbol{y}_{i+2}^{(3)}, \tag{3.57}$$

$$\boldsymbol{y}_i^{(4)} = b_{11}^- \cdot \boldsymbol{f}_{i+1} + b_{21}^- \cdot \boldsymbol{f}_{i+2} - a_{11} \cdot \boldsymbol{y}_{i+1}^{(4)} - a_{21} \cdot \boldsymbol{y}_{i+2}^{(4)}. \tag{3.58}$$

The output signal $\boldsymbol{u}$ is then obtained by a point-wise sum of these results:

$$\boldsymbol{u}_i = \boldsymbol{y}_i^{(1)} + \boldsymbol{y}_i^{(2)} + \boldsymbol{y}_i^{(3)} + \boldsymbol{y}_i^{(4)}. \tag{3.59}$$

As it turns out, this filter is very efficient and accurate for arbitrary standard deviations. As found in [Hal06b], this filter surpasses the quality of other recursive filters such as the one of Deriche significantly for about $\sigma > 32$. Below this value, the Deriche filter is said to yield slightly better results. However, since we are primarily interested in large standard deviations – smaller ones can easily be established with more simple techniques such as spatial convolution with a Gaussian – we refrain to this 'parallel' Young-van-Vliet filter for the time being.

To finish this section, let us briefly recapitulate this filtering algorithm. Given a particular $\sigma$, the process starts with a precomputation phase which is dominated by the search for an appropriate $q$. Once all parameters are computed, this filter requires only 15 additions and 16 multiplications per sample. Moreover, both forward runs and both backwards runs can be performed in parallel, each. This allows for a very efficient usage of caching mechanisms of modern hardware. In Section 3.6, we compare this recursive filter to other implementations for linear diffusion.

### 3.3.6 Iterated Box Filtering

Another very fast approximation of the Gaussian Filter is derived from the *central limit theorem*. This theorem states that, given a sufficiently large number $n$ of random variables $X_1, \ldots, X_n$ with the same distribution with mean $\mu$ and variance $\sigma^2$, their 'average'

$$Y_n = \frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n}\sigma} \tag{3.60}$$

approaches a (0,1) normal distribution for $n \to \infty$ [Kre05]. If this property is carried over to the design of filters, it means that a sufficient number of convolutions with an arbitrary, but similar kernel has a similar effect as one application of a specific Gaussian kernel. This gives rise to use multiple applications of simple so-called *box filters* to approximate a Gaussian

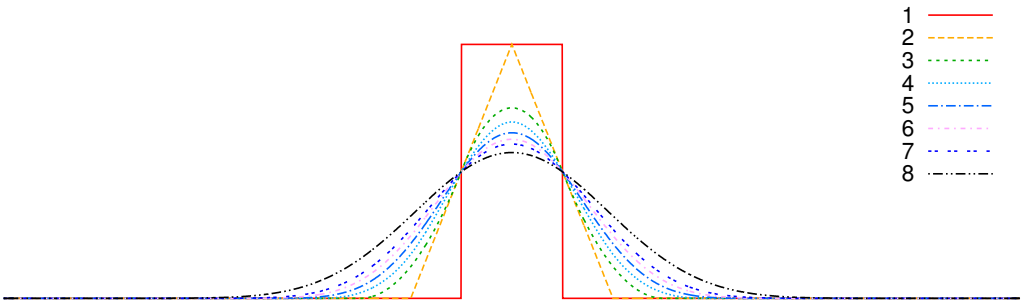



**Figure 3.1**: Visualisation of the central limit theorem. The more box kernels are convolved with each other, the closer approximates the result a Gaussian.

filter [Wel86]. Figure 3.1 visualises this idea. A convolution of already 3–5 kernels approximates a Gaussian very well.

Box filters are famous for their runtime efficiency, but they are seldomly used for theoretical derivations. This is the reason why in the literature, these filters are usually designed in a purely discrete context without reference to their continuous analogue. However, since we are going to review and extend this class of filters in the following section, we approach this problem from the opposite side, and regard the classical form as the discretisation of the continuous variant:

**Definition 3.2 (Continuous Box Filter)**

*A continuous* box filter $B_\Lambda$ *with a real-valued length* $\Lambda \in \mathbb{R}^+$ *is a convolution*

$$(B_\Lambda * f)(x) \;:=\; \int_{-\infty}^{\infty} B_\Lambda(x-y) \cdot f(y)\, dy \tag{3.61}$$

*of a signal* $f$ *with a* box kernel

$$B_\Lambda(x) \;:=\; \begin{cases} \frac{1}{\Lambda}, & x \in (-\lambda, \lambda) \\ 0, & \textit{else} \end{cases} \tag{3.62}$$

*for* $x \in \mathbb{R}$ *and* $\Lambda = 2\lambda$.

□

In the literature, the discretisation is often performed by sampling the problem at a grid with spacing $h$, and by rounding the length $\Lambda$ to the closest grid distance $L := h\lfloor \frac{\Lambda}{h} + \frac{1}{2} \rfloor$. This leads to the following discrete filter.

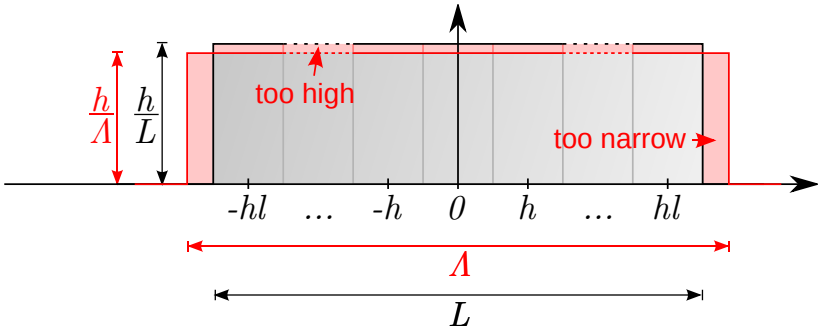



**Figure 3.2**: In general, the classical discrete box kernel does not approximate the continuous box kernel well.

### Definition 3.3 (Discrete Box Filter)

*A* discrete box filter $\boldsymbol{B}_L$ *of length* $L = h(2l+1)$, $l \in \mathbb{N}_0$, *and sampled at an equidistant grid of spacing* $h > 0$ *is a convolution*

$$(\boldsymbol{B}_L * \boldsymbol{f})_{hk} \;:=\; \sum_{m\in\mathbb{Z}} (\boldsymbol{B}_L)_{h(k-m)} \cdot \boldsymbol{f}_{hm} \tag{3.63}$$

*of a signal* $\boldsymbol{f}$ *with a* discrete box kernel

$$(\boldsymbol{B}_L)_{hk} \;:=\; \begin{cases} \frac{h}{L}, & -l \leqslant k \leqslant l \\ 0, & \textit{else} \end{cases} \tag{3.64}$$

*for* $k \in \mathbb{Z}$.

□

Figure 3.2 shows a comparison between the continuous formulation from Definition 3.2 and the discrete version from Definition 3.3 in 1-D. As a consequence from the approximation, $\boldsymbol{B}_L$ does in general not exactly resemble the continuous kernel $B_\Lambda$ that it is supposed to approximate. In most cases, it is either too wide and low or, as depicted in this figure, too narrow and tall. However, the approximation becomes better the finer the grid. To this end, $\boldsymbol{B}_L$ thus approaches $B_\Lambda$ for $h \to 0$. For $h = 1$, we obtain the form that is most frequently found in the literature [Wel86, YvV95, Hal06b, GW08].

1-D box filters can be implemented very efficiently if we exploit the simple structure of their kernels. This leads to a 'sliding window' approach. Each sample of the result is obtained from its predecessor, compensated by the change that occurred between them [Wel86]:

$$(\boldsymbol{B}_L * \boldsymbol{f})_{hi} \;=\; (\boldsymbol{B}_L * \boldsymbol{f})_{h(i-1)} + \frac{h}{L}\left(\boldsymbol{f}_{h(i+l)} - \boldsymbol{f}_{h(i-l-1)}\right). \tag{3.65}$$

Because the first element does not have a predecessor, it must still be computed by a summation over the affected input samples. Hence, for a signal of length $n$ and a kernel with $2l + 1$ samples, box filtering has a runtime complexity of $\mathcal{O}(n + l)$. Moreover, if we additionally assume $l < n$, this leads to an accumulated linear runtime complexity. Due to the separability in $n$-D, this complexity carries over to the multi-dimensional case, such that we obtain an accumulated complexity $\mathcal{O}(n)$ in the number of pixels or voxels $n$.

By the central limit theorem, we can convolve the signal several times with a box filter to approach the result for Gaussian convolution. This removes artefacts that arise from the piecewise linearity of the box kernel, as well as from the rotational invariance property in multiple dimensions. For simplicity of notation, we denote the new kernel by $\boldsymbol{B}_L^d$:

$$\boldsymbol{B}_L^d := \underbrace{\boldsymbol{B}_L * \ldots * \boldsymbol{B}_L}_{d\text{-times}} \tag{3.66}$$

To this end, applying $\boldsymbol{B}_L$ $d$-times to a signal comes down to the convolution with a $C^{d-1}$-continuous kernel $\boldsymbol{B}_L^d$. It possesses the variance $\sigma^2(\boldsymbol{B}_L^d)$ [Wel86]:

$$\sigma^2(\boldsymbol{B}_L^d) \;=\; d\,\frac{L^2 - h^2}{12}. \tag{3.67}$$

As can be seen in this equation, sigma only depends on two quantised variables, the length $L$ and the number of iterations $d$. Moreover, $L$ is required to be (a multiple of) an odd integer, since the kernel must be symmetric around 0. As a consequence, the set of admissible values for $\sigma$ is quantised. For example, assume we use $d = 3$ iterations and set $h = 1$. Then, we can not find an interval of length 1 which contains more than one admissible value for $\sigma$. As we are going to see in Section 3.6.2, this limitation has a significant impact on the quality of the filtering results.

In the literature, people try to remedy this problem by using kernels with different lengths [Wel86]. However, this approach bears new problems:

1. Although the arising filter still possesses low-pass characteristics, it does in general not approximate a Gaussian as well as a series of similar kernels.

2. Because in practice, $d$ is usually chosen from $\{3, 4, 5\}$, the combinatorial possibilities for $\sigma$ are still relatively small.

3. Even if $d$ is large, finding the optimal configuration of different box kernels can be tedious.

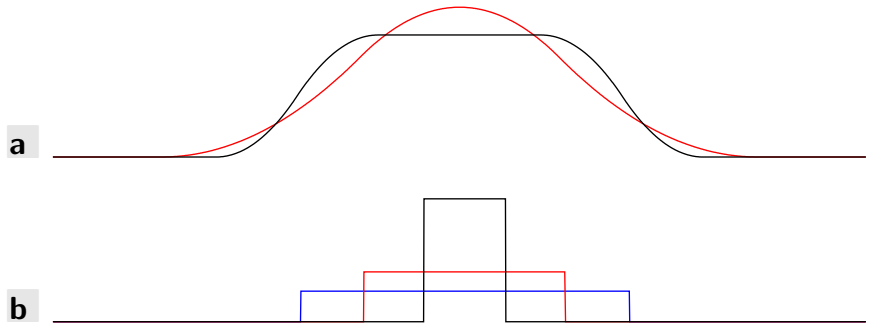



**Figure 3.3**: **a.** Convolution of three box kernels with same length (red) vs. three box kernels with different length (black). **b.** Kernels used for a., scaled by 0.25 in height. The red graph from a. was obtained by a convolution of three red boxes, the black result by a convolution of one blue and two black ones.

While issues 2 and 3 are intuitively clear, the first problem might require further explanation. Hence, let us compare the results of an iterated box filter $\boldsymbol{B}_{11}^3$ with the ones obtained by the combination $\boldsymbol{B}_{13}^2 * \boldsymbol{B}_5$. By Bienaymé's formula, the variance of a sum of uncorrelated random variables equals the sum of individual variances [Kre05]:

$$\sigma^2(X_1 + \ldots + X_n) = \sum_{i=1}^{n} \sigma^2(X_i) \tag{3.68}$$

This leads to $\sigma^2(\boldsymbol{B}_{11}^3) = 3 \cdot 10 = 30$, and $\sigma^2(\boldsymbol{B}_{13}^2 * \boldsymbol{B}_5) = 2 \cdot 14 + 2 = 30$. We compare the impulse responses of these filters on a discrete signal with the one of a convolution with a Gaussian with standard deviation $\sigma = \sqrt{30}$ truncated at $15\sigma$, and compute the $L^2$ error. While $\boldsymbol{B}_{11}^3$ has an error $1.21 \cdot 10^{-7}$, the error for the combination $\boldsymbol{B}_{13}^2 * \boldsymbol{B}_5$ is already twice as high, namely $2.53 \cdot 10^{-7}$. Moreover, the gap between these errors is significantly increased as the variance in the used boxes' lengths grows. In particular for large standard deviations, the approximation quality using this technique can thus grow significantly.

We can convince ourselves about the reasons for this striking difference if we regard the outcomes of different convolutions with box kernels graphically, as shown in Figure 3.3. The resulting kernels from this operation that are depicted in part a. are the 'effective' kernels of the whole cascade of box filters. While the convolution of three similar boxes yields a kernel which already approximates the corresponding Gaussian very well, a convolution of two small boxes with one big box still reveals the overall shape of the larger kernel. In the latter case, a significantly higher number of iterations is required to approach the Gaussian as well as in the uniformly sized case.

In the next section, we see two alternatives to this approach which allow a good approximation of Gaussians with arbitrary standard deviation.

## 3.4 Numerical Improvements

### 3.4.1 Box Filtering with Correction

In the previous sections, we observed some characteristics of iterated box filtering on the one hand, and discrete convolution with a truncated Gaussian on the other:

1. Iterated box filters are very efficient and approximate a Gaussian well even for few iterations, but they only yield certain standard deviations. Their runtime complexity is $\mathcal{O}(dn)$.

2. For a small number of iterations $d$, the distance between two standard deviations that can be expressed by iterated box filters is relatively small.

3. Discrete convolution with a truncated Gaussian kernel has a runtime complexity of $\mathcal{O}(\sigma n)$. If there exists a $c \ll n$ such that $\sigma < c$ for all $\sigma$, this operation is very efficient with a complexity of $\mathcal{O}(n)$.

This gives rise to the design of a new filter that combines the advantages of both worlds. We use Bienaymé's formula to rewrite the Gaussian of the desired filter by a convolution of two Gaussians:

$$K_\sigma = K_{\sigma_B} * K_{\sigma_G}\,, \quad \text{where } \sigma^2 = \sigma_B^2 + \sigma_G^2\,. \tag{3.69}$$

In particular, we can choose $\sigma_B$ such that $K_{\sigma_B}$ can be expressed by an iterated box filter. If $\sigma_B \gg \sigma_G$, the arising filter becomes very efficient:

**Definition 3.4 (Iterated Box Filter With Correction)**

*A* $d$-times iterated box filter with correction *is the convolution*

$$\boldsymbol{C}_\sigma^d * \boldsymbol{f} \tag{3.70}$$

*of an image* $\boldsymbol{f}$ *with a* corrected iterated box kernel

$$\boldsymbol{C}_\sigma^d = \boldsymbol{B}_L^d * [K_{\sigma_G}]\,, \tag{3.71}$$

*where*

$$L = h\left(2\left\lfloor\frac{\sqrt{12\frac{\sigma^2}{dh^2}+1}-1}{2}\right\rfloor+1\right)\,, \tag{3.72}$$

*and*

$$\sigma_G = \sqrt{\sigma^2 - d\frac{L^2 - h^2}{12}}\,. \tag{3.73}$$

□

In (3.72), $\lfloor x \rfloor$ denotes the so-called floor function, which computes the largest integer not greater than $x \in \mathbb{R}$. By construction, $L$ the largest admissible length of a box filter that has a smaller standard deviation than $\sigma$. Its computation in (3.72) follows directly from (3.67). As a consequence we obtain:

**Theorem 3.1** → Proof 1

*Given an iterated box filter with correction $\boldsymbol{C}_\sigma^d$, the standard deviation $\sigma_G$ of the correcting Gaussian kernel $[K_{\sigma_G}]$ is bounded by*

$$\sigma_G < \sqrt{\frac{dh^2}{3}\left(\sqrt{12\frac{\sigma^2}{dh^2} + 1} - 1\right)} \tag{3.74}$$

*if $L > h$, and by $\sigma_G = \sigma$ otherwise.*

□

The special case $L = h$ ($L$ cannot be smaller than $h$, see (3.72)) is not surprising. In this case, the component $\boldsymbol{B}_L^d$ comes down to a discrete unit peak:

$$\boldsymbol{B}_h^d(hk) = \begin{cases} 1, & k = 0 \\ 0, & \text{else} \end{cases}, \tag{3.75}$$

such that $\boldsymbol{C}_\sigma^d = [K_{\sigma_G}]$. Since we do not perform box filtering in this case, it holds that $\sigma_G = \sigma$.

As an example for the more interesting case $L > h$, assume we want to perform Gaussian filtering with a standard deviation $\sigma = 100$ on a grid with $h = 1$ using a box filter with correction in $d = 3$ iterations. Using Definition 3.4, we obtain the length $L = 199$, and as a consequence, $\sigma_G = 10$. This is in accordance with Theorem 3.1, which predicts an upper bound $\sigma_{G,\max} \approx 14.11$.

However, Theorem 3.1 also reveals that $\sigma_G$ grows proportional to $\sqrt{\sigma}$. Thus, the iterated box filter with correction possesses a runtime complexity of $\mathcal{O}(dn + \sqrt{\sigma}n)$ (or equivalently $\mathcal{O}(dn + \sqrt[4]{T}n)$). As it turns out, a correction is more important the smaller the standard deviation of the kernel is. To this end, we can reduce the runtime for real problems without sacrificing the overall quality if we omit the correction step for large $\sigma$. In Section 3.6, we go more into detail about this idea.

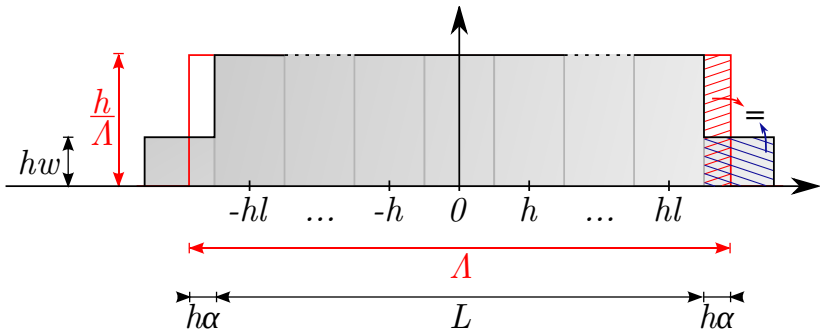



**Figure 3.4**: Construction of an extended box filter (grey) out of a continuous prototype (red) by redistributing weights at both ends.

### 3.4.2 Extended Box Filtering

While iterated box filtering with correction is very well suited to remedy the bad approximation of Gaussians by iterated box filtering, it also introduces a significant dependency of the runtime from the standard deviation. To some extent, this property counteracts the nature of box filters, which typically only depend on the numbers of pixels and iterations. Hence, let us discuss a fast alternative to the aforementioned approach, which does not possess this problem.

To this end, let us come back to the continuous box filter as given in Definition 3.2. Because the continuous box kernel can have arbitrary real-valued lengths, its standard deviation can freely be adjusted. The goal of this section is to find a better discretisation $\boldsymbol{E}_\Lambda$ of the continuous box filter $B_\Lambda$ than is provided by the classical discrete box filter $\boldsymbol{B}_L$. This *extended* box filter shall fulfil the following criteria [GGBW11]:

1. The new filter $\boldsymbol{E}_\Lambda$ must be continuous over $\Lambda$, such that kernels with an arbitrary standard deviation can be constructed. The variance $\sigma^2(\boldsymbol{E}_\Lambda)$ shall be a smooth function in $\Lambda$.

2. $\boldsymbol{E}_\Lambda$ must extend the classical discretisation in a way that it comes down to a standard box filter $\boldsymbol{B}_L$ for $\Lambda \to L$.

3. Moreover, $\boldsymbol{E}_\Lambda$ must approach the continuous box filter $B_\Lambda$ for $h \to 0$, i.e. $\lim\limits_{h\to 0} \sigma^2(\boldsymbol{E}_\Lambda) = \sigma^2(B_\Lambda)$.

In order to achieve these requirements, we write the continuous length $\Lambda$ as a sum of an integer length $L$ that can be obtained by a classical box filter, and two small increments $0 \leqslant \alpha < 1$:

$$\Lambda = h(2l + 1 + 2\alpha) = L + 2h\alpha\,, \tag{3.76}$$

where $l \in \mathbb{N}_0$ as it is for the conventional box filter. Considering that the continuous box filter possesses the height $\frac{h}{\Lambda}$, we redistribute the area for the two extensions, $h\alpha \cdot \frac{h}{\Lambda}$, onto the sampling points $\pm h(l+1)$. This is shown in Figure 3.4. To this end, we obtain two additional weights at both sides, each with height $w = \frac{h\alpha}{\Lambda}$. This leads to the following construction [GGBW11]:

**Definition 3.5 (Extended Box Filter)**

*An* extended box filter $\boldsymbol{E}_\Lambda$ *with a real-valued length* $\Lambda \in \mathbb{R}^+$ *and discretised on a uniform grid of spacing* $h > 0$ *is a convolution*

$$(\boldsymbol{E}_\Lambda * \boldsymbol{f})_{hk} \;:=\; \sum_{m\in\mathbb{Z}} (\boldsymbol{E}_\Lambda)_{h(k-m)} \cdot \boldsymbol{f}_{hm} \tag{3.77}$$

*of a signal* $\boldsymbol{f}$ *with an* extended box kernel

$$(\boldsymbol{E}_\Lambda)_{hk} \;:=\; \begin{cases} \frac{h}{\Lambda}, & -l \leqslant k \leqslant l \\ hw, & k \in \{-(l+1), l+1\} \\ 0, & \textit{else} \end{cases} \tag{3.78}$$

*with*

$$l \;:=\; \left\lfloor \frac{\Lambda}{2h} - \frac{1}{2} \right\rfloor, \qquad w \;:=\; \frac{1}{2}\left(\frac{1}{h} - \frac{2l+1}{\Lambda}\right), \tag{3.79}$$

*and* $k \in \mathbb{Z}$.

□

Compared to Definition 3.3 of the classical box filter, the extended box filter only requires small modifications. The weight $w$ was added, and $l$ was modified to preserve the average grey value during filtering. Similar as in (3.66), we write

$$\boldsymbol{E}_\Lambda^d := \underbrace{\boldsymbol{E}_\Lambda * \ldots * \boldsymbol{E}_\Lambda}_{d\text{-times}} \tag{3.80}$$

As a consequence of this high similarity, the new construction also possesses many advantages of classical box filtering. It is separable in $n$-D, and can still be implemented in a sliding-window manner:

$$\begin{aligned} (\boldsymbol{E}_\Lambda * \boldsymbol{f})_i \;=\; (\boldsymbol{E}_\Lambda * \boldsymbol{f})_{i-1} + \left(\frac{h}{\Lambda} - hw\right)(\boldsymbol{f}_{i+l} \;\; - \boldsymbol{f}_{i-l-1}) \\ + \qquad hw \;\; (\boldsymbol{f}_{i+l+1} - \boldsymbol{f}_{i-l-2})\,. \end{aligned} \tag{3.81}$$

After computing the result for the first pixel in a row, only four additions and two multiplications are required, because $hw$ and ${}^{h}/_{\Lambda} - hw$ are both constants that can be precomputed before the program run. Hence, we obtain the same runtime complexity $\mathcal{O}(dn + l)$ as for conventional box filtering. Moreover, because we can in general assume that $l < n$, the accumulated runtime comes down to $\mathcal{O}(dn)$.

Let us now discuss some mathematical properties that simplify and explain the use of extended box filters for real-world applications. A crucial property is the variance of the arising construction, as it determines which extended box filter must be iterated to approximate a given Gaussian best. As a first step, let us note that $w$ depends linearly on $\alpha$:

$$w \;=\; \frac{1}{2h}\left(1 - \frac{(2l+1)h}{\Lambda}\right) \;=\; \frac{1}{2h}\left(1 - \frac{\Lambda - 2\alpha h}{\Lambda}\right) \;=\; \frac{\alpha}{\Lambda}. \tag{3.82}$$

Since $\alpha$ is a component of $\Lambda$, we can thus express $w$ as a function on $\Lambda$. Moreover, all other parameters of extended box filtering are by definition related to $\Lambda$. Hence, it is feasible to talk about $\Lambda$ as the 'effective' length of the kernel, although its value does in general not appear verbatim in the layout of such filter.

As a consequence, let us note the variance of an extended box filter with effective length $\Lambda$. As done for the conventional box filter, we regard the general case for $d$ iterations:

**Theorem 3.2** → Proof 2

*The variance $\sigma^2(\boldsymbol{E}_\Lambda^d)$ of a $d$-times iterated extended box kernel with effective length $\Lambda$ is given by*

$$\sigma^2(\boldsymbol{E}_\Lambda^d) \;=\; \frac{dh^3}{3\Lambda}\left(2l^3 + 3l^2 + l + 6\alpha(l+1)^2\right) \;. \tag{3.83}$$

□

Although this relation cannot be written in a simple form as in (3.67) for the conventional box filter, this equation falls back to the conventional case if $\alpha = 0$. To this end, the extended box filter generalises the conventional box filter by an an option to select continuous lengths $\Lambda$:

**Theorem 3.3** → Proof 3

*The extended box kernel $\boldsymbol{E}_\Lambda$ constitutes a generalisation of the discrete box kernel $\boldsymbol{B}_L$, i.e. $\boldsymbol{E}_L = \boldsymbol{B}_L$ for $L \in \mathbb{N}_{odd}$,*

$$\forall L \in \mathbb{N}_{odd} : \lim_{\Lambda \to L^+} \sigma^2(\boldsymbol{E}_\Lambda^d) = \sigma^2(\boldsymbol{B}_L^d)\,, \qquad \textit{and} \tag{3.84}$$

$$\lim_{\Lambda \to (L+2h)^-} \sigma^2(\boldsymbol{E}_\Lambda^d) = \sigma^2(\boldsymbol{B}_{L+2h}^d). \tag{3.85}$$

□

Moreover, we also want to guarantee that $\sigma^2(\boldsymbol{E}_\Lambda^d)$ satisfies the intuitive dependency on the length $\Lambda$. To this end, we assume the limiting case $h \to 0$, for which the extended box filter approaches its continuous prototype:

**Theorem 3.4** → Proof 4

*The extended box kernel $\boldsymbol{E}_\Lambda$ is a suitable discretisation of a box kernel $B_\Lambda$ in the continuous domain, i.e. for d-times application,*

1. *its variance approximates the continuous analogue arbitrarily well:*

$$\lim_{h \to 0} \sigma^2(\boldsymbol{E}_\Lambda^d) = \sigma^2(\boldsymbol{B}_\Lambda^d), \qquad \textit{and} \tag{3.86}$$

2. *the order of consistency is $\mathcal{O}(h^2)$.*

□

Note that we obtain the same result for the discrete box filter for $h \to 0$. This can be shown analogously to Proof 4, but with (3.67) instead of (3.83) as the variance. Once again, this insight indicates that the extended box filter is a natural extension of the conventional idea.

Moreover, in the special case of a small extended box kernel with only one central weight, the extended box filter equals one explicit diffusion step with the same stopping time:

**Theorem 3.5** → Proof 5

*One iteration of an extended box filter $\boldsymbol{E}_\Lambda$ with $h < \Lambda < 3h$ equals one explicit diffusion step with a second-order finite difference discretisation in space and a time step width*

$$\tau = \frac{\sigma^2(\boldsymbol{E}_\Lambda)}{2}\,. \tag{3.87}$$

□

Finally, let us remark that the extended box filter is not only closely related to the classical discrete box filtering theory, but also shares a number of common concepts with other filters. By (3.81), extended box filtering may be written in a sliding window form. To this end, one output sample is computed from a linear combination of four input samples and one output sample. This idea follows a similar spirit as recursive filters [Der87a, YvV95].

However, while recursive filters use a small local stencil and typically read balanced numbers of output and input samples, extended box filtering only accesses one output sample that lies close to the processed grid point, plus few input values in a greater distance. As a result, box filters create symmetric results by construction, while recursive filters must always be applied at least once in both directions to obtain a similar result. To this end, extended box filtering is superior to recursive filters when the signal does not possess a specified length. Such situations occur for instance for audio and video streaming, or for data streams gathered by a sensor.

Moreover, extended box filtering can be seen as a special case of stacked integral image filtering [Cro84, BSB10]. This technique assembles a Gaussian kernel from a stack of different box kernels. Each of these box kernels has a unique width and height such that the 'heap' formed by these boxes approximates the Gaussian kernel well. While the heights are typically real-valued, the widths of all individual boxes reside on a grid. Hence, we can write the extended box kernel as a stack of one box that is $h(2l+3)$ wide and $hw$ high, and one box of length $h(2l+1)$ and height $h/\Lambda - hw$. The difference to the original idea is that while stacked integral image filtering approximates a continuous Gaussian, extended box filtering approximates a continuous box filter. In the first case, the result is obtained from a sum over different impulse responses. In order to enhance the quality, more boxes must be added. This increases the approximation quality, but does (in a continuous sense) not increase the smoothness of the arising convolution kernel. In contrast, extended box filtering can be written as a convolution of sums of scaled standard box filters $w_1 \boldsymbol{B}_L$ and $w_2 \boldsymbol{B}_{L+2h}$, where $w_1 = hw\Lambda$, and $w_2 = 1 - hw\Lambda$:

$$\boldsymbol{E}_\Lambda^d = \underbrace{(w_1 \boldsymbol{B}_L + w_2 \boldsymbol{B}_{L+2h}) * \ldots * (w_1 \boldsymbol{B}_L + w_2 \boldsymbol{B}_{L+2h})}_{d\text{-times}} \tag{3.88}$$

By adding more iterations, both the approximation quality and the smoothness of $\boldsymbol{E}_\Lambda^d$ can be increased simultaneously. This fact is the reason why Gaussian convolution can in general much more accurately be performed by extended box filtering than it is by Gaussian convolution.

## 3.5 Efficient GPU-Based Algorithms

### 3.5.1 Explicit Linear Diffusion

As a first GPU-based algorithm, let us design a 2-D explicit scheme for linear diffusion as developed in Section 3.3.1. Because the operations necessary to

compute (3.9) only depend on the intermediate solution at time step $k$, the operation is fully data-parallel. As a consequence, the arising problem can easily be formulated in a parallel manner. In order to preserve the integrity of input and output, we must use two buffers in GPU memory. We can then always compute the update from the first into the second buffer, and exchange their pointers such that the roles of the buffers are flipped.

However, our algorithm's runtime still crucially depends on the strong memory-boundedness of the explicit scheme. By assuming $h = 1$ in (3.9), the update of one pixel requires 3 multiplications and 5 additions, but also 5 read and 1 write operations to global memory. On GPUs, this leads to a misbalance. Arithmetic operations can in general be performed in 4 or 1 cycles, depending on whether the GPU supports CUDA Compute Capability 1.x or 2.x. Memory interactions, in contrast, require about 400-800 cycles [NVi11b]. Hence, a major criterion for the runtime is given by the use of fast on-chip memory for caching.

CUDA textures offer a simple and efficient solution to this problem (see Section 2.2.2). Their 2-D aware cache helps to hide memory latencies behind computations. Moreover, their adjustable boundary behaviour is ideally suited to emulate reflecting boundary conditions.

To this end, our algorithm works as follows. After uploading the image data into the first buffer on the device, the algorithm iterates over the number of steps. In each of these steps, it first binds a previously configured texture reference to the buffer that contains the image. In the parallel execution kernel that is called thereafter, image information is requested by texture fetches and stored back in large chunks from shared memory. After a synchronisation, the device pointers are flipped and the iteration continues until the desired stopping time is reached. Finally, the result is downloaded from the device.

### 3.5.2 Implicit Linear Diffusion

When it comes to the structure of the execution kernel, an algorithm for one Jacobi step of an implicit solver does not differ much from the previously mentioned explicit diffusion kernel. The update rule from (3.17) can be implemented straightforwardly. However, the data flow is more expensive in this case, since both the 'right hand side' from the previous outer iteration $k$, as well as the result from the previous inner iteration must be accessible within the kernel.

For the same reasons as before, the 'offset' stencil calls arising from $u_{m,n}^{k+1,l}$ can best be realised by texture fetches. In contrast to this, the right hand side $u_{i,j}^{k}$ is only queried in the 'central' pixel with respect to the stencil,

hence possessing the same alignment as the output. As a consequence, it is easily fetched by coalesced loads from global memory.

On the CPU side, this new data dependency also requires changes to the buffer juggling mechanism. Instead of performing a double buffering technique as for the explicit scheme, the algorithm must now take care of three arrays. During the 'inner' iterations over $l$, two of them are perpetually exchanged and alternately bound to a texture, while the third buffer is held constant. After each cycle of inner iterations, this 'right hand side' is then exchanged with the most current solution from the inner cycle. This process is iterated until the desired number of 'outer' steps is reached.

### 3.5.3 Spatial Convolution

Implementing a spatial convolution with a Gaussian is straightforward from a logical perspective, but challenging when it comes to the technical implementation. The reason for this is given in the strong memory-boundedness of the algorithm. For every pixel of the result, a large neighbourhood in the size of the support of the truncated kernel must be read. Moreover, the arithmetics involved in this case are inexpensive, such that the whole efficiency of the algorithm comes down to a fast throughput of data.

Unfortunately, this throughput can in general not be reduced. Although a Gaussian can easily be computed on-the-fly, its truncation requires a linear scaling of the remaining kernel samples to preserve the mass of the filtered image. As a consequence, we must precompute the convolution kernel prior to filtering, and apply it to the image. However, by using the separability of the process and the symmetry of the kernel, we can still save many loads and stores. Hence, we regard both directions independent from each other and load the kernel in both cases only once for left-offset and right-offset input samples.

#### Texture-Based Approach

Considering row-first ordering, convolution along the y axis is a very memory-intensive task, since it requires several memory lines to be cached. Here, an implementation in which both the signal and the kernel are fetched as 2-D and 1-D textures, respectively, turns out to be the fastest solution. Such an algorithm is even faster than one which works completely on shared memory. A possible interpretation to this observation is that large implicit memory fetches into 2-D caches of textures are faster than single line loads into shared memory.

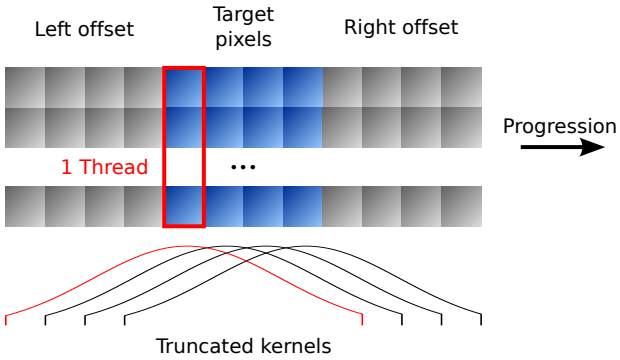



**Figure 3.5**: Fast Gaussian convolution in shared memory, following the linear direction of memory. Shared memory contains the input (grey and blue), as well as solution vectors (only blue range).

Convolutions along the coherent direction of the memory are even more interesting. If the support of the truncated kernel is too large to fit into shared memory, there is no option other than performing a similar strategy as above. Textures work well on coherent memory, and mask a large portion of random access to memory by their large caches. This approach works regardless of the chosen standard deviation.

### Cache-Based Approach for Small Standard Deviations

If all samples for the kernel and the associated input samples both fit into the shared memory, there is a second possibility to speed up the algorithm significantly. In a linear sense, we assign each pixel to one thread. Offset memory operations are then executed in a laminar fashion, such that this process is free of bank conflicts. Moreover, each thread handles eight rows in terms of a loop unrolling scheme. This allows to mask occurring memory latencies to a minimum. Figure 3.5 visualises this algorithm. For each block of pixels for which a solution is computed, a three times larger array with input data is additionally kept in shared memory. To this end, each pixel can be convolved with a (discrete) kernel which has a support of at most $2n + 1$ elements, where $n$ is the length of one block. After all target pixels are processed, the range denoted by 'right offset' becomes the range of new 'target pixels', the old central inputs become the new 'left offset', and a new 'right offset' is loaded from global memory. This ring buffering technique causes each input pixel only to be read once from global memory. If we design the length of one (output) block as 64 pixels and assume 8 rows for loop unrolling, this buffer can still be represented in shared memory. It allows for a convolution with a Gaussian of standard deviation $\sigma = 12$ with

a cutoff at $5\sigma$. To this end, this second algorithm is very well suited for regularisation purposes in the context of more complex frameworks, where the standard deviation is typically very small.

### 3.5.4 Multiplication in the Frequency Domain

Gaussian filtering in the frequency domain is algorithmically challenging, in particular because our main focus lies on efficiency. Our algorithm is dominated by the fast Fourier transform, such that its overall efficiency is directly linked to the performance of the FFT. While the basic idea behind the FFT is simple and easy to implement, several extensions and optimisations are proposed in the literature to optimise the absolute runtime. To this end, the classical algorithm [CT65] is often complemented by other approaches which perform better on special (e.g. prime) problem sizes [Rad68]. Besides these, a number of technically motivated changes, such as optimisations on caching patterns are frequently applied, in particular when it comes to parallel hardware [Sto66, GLD+08]. A good overview of state-of the-art techniques in this field is given in [FJ05].

However, since fast Fourier transforms belong to the standard tools in signal processing, there are many GPU-based algorithms in the literature [MA03, GLD+08, NVi10a]. Some of these algorithms are even freely available for download. Today, the `cuFFT` library provided by NVidia is one of the best maintained software suites for this purpose. Its performance improved significantly in the recent years, and many concepts known from sequential algorithms are now also found in this library [NVi10a]. Moreover, `cuFFT` is shipped together with the CUDA API, such that it is available on all CUDA-enabled platforms. This also makes it very convenient to use in our algorithm. By calling `cuFFT` for both the forward and the backward transform, the arising high-level algorithm consists only of a handful of well-structured operations which are easy to design and optimise.

Note that the CUDA SDK provided by NVidia already provides a very simple implementation of an FFT-based convolution routine [NVi11a]. As it turns out, this method is very specialised, as it assumes kernels of a certain size only, and does not provide reflecting boundary conditions. Nevertheless, the approach we develop in the following pursues a similar strategy, and complements these ideas by additional operations where necessary. Since these modifications are pretty straightforward, we can see our algorithm as a generalisation of NVidia's approach.

Since we are only considering real-valued input and output data during the computation, our algorithm can be accelerated by using real-to-complex forward transforms and complex-to-real backwards transforms. While these

transforms possess the same runtime as their complex-to-complex valued variants, they use the symmetry of the Fourier plane, and read and write only a half-plane. This allows the intermediate scaling operation to be performed much more efficiently.

The essential operation, a point-wise multiplication with the dual of the Gaussian as in (3.25), is arithmetically inexpensive but memory-intensive. For each multiplication, we must load two values from global memory (the signal and the kernel), as well as store one resulting value. Since these memory operations are about two orders of magnitude slower than arithmetic operations, this setup immediately leads to a severe bottleneck. A remedy to this problem is found in the immediate computability of the dual of the Gaussian in frequency space as given in (3.29). It allows to save one load per pixel if we compute the value of the Gaussian inline. This requires 7 multiplications, 1 addition, and 1 call to a hardware-accelerated intrinsic function for the exponential function. On a Fermi architecture (Compute Capability 2.0), these computations require about 20 clock cycles in average [NVi11b]. Not only is this strategy more than one order of magnitude faster than accepting a 400–800 cycles latency for loading the data, but it can also easily be hidden behind load and store latencies for input and output. To this end, the computation of the Gaussian does not measurably increase the runtime.

As an additional means of acceleration, we formulate the 2-D point-wise multiplication with a simple 1-D CUDA kernel. This is numerically equivalent to a real 2-D operation, since the underlying 2-D coordinates can directly be computed from the 1-D thread ID and be used as an argument to the exponential function. However, the advantage of such multiplication is a better tessellation of the Fourier plane with CUDA primitives. The lower padding requirement arising from this setup additionally reduces the memory throughput of this operation.

Finally, let us briefly remark on the mirroring of the input image to fulfil its periodicity, as explained in Section 3.3.4. It is immediately clear that the effective size of the image to be transformed and multiplied grows by a factor 4 as a consequence of this procedure, which is then also reflected in the overall runtime. As we are going to see in Section 3.6.4, this modification also significantly limits the maximal image size possible to be computed by the GPU-based algorithm. However, the mirroring procedure itself can be efficiently realised if we consider that the process consists of two independent mirrorings along different symmetry axes. Assuming row-first ordering, the mirroring along the vertical axis is expensive, since it involves a reversal of coherent memory structures. However, this operation can still be performed efficiently if data is read via a texture. By setting the pitch of the input

to the total width, it can be bound to the 'top left' quadrant of the buffer, while the 'top right' quadrant is written. Note that this proceeding is valid in terms of the CUDA API, since we can ensure that no field of the buffer that is read will also be written. The subsequent mirroring step along the horizontal axis is even more inexpensive, since the coherency of memory lines is preserved during this step. It can thus be efficiently realised by `cudaMemcpy` operations.

### 3.5.5 Recursive Filtering

The GPU-based implementation of the 'parallel' Young-van-Vliet filter presented in Section 3.3.5 extends the sequential algorithm from [Hal06a] by a parallelisation strategy for GPUs, which follows similar design concepts as our algorithm for spatial convolution (see Section 3.5.3). To the beginning of the program run, the coefficients from (3.46)–(3.47) and (3.51)–(3.54) are computed on the CPU. A parallel algorithm for this step does not pay off, since these 12 coefficients are the same for all pixels in the image domain. Moreover, the scaling factor $q$ from (3.32) is optimised by a manual optimisation step which is also an inherently sequential operation which cannot be parallelised.

If we again assume row-first ordering, the filtering process starts with a convolution in y direction. It consists of one forward run and one backward run, since the individual filtering steps (3.55)–(3.56) and (3.57)–(3.58) are pairwise grouped to reduce calls to global memory. Because vertical stripes of pixels are data-parallel with respect to each other, each thread handles one column. This is represented by a linear CUDA ThreadGrid of size 512. Hence, a common load of input data comes down to a large coherent access to global memory. The 'offset' input and feedback values $\boldsymbol{f}_{\cdot\pm1}$ and $\boldsymbol{y}^{(\cdot)}_{\cdot\pm1}$ from (3.55)–(3.58) can conveniently be stored in registers.

For the other direction along the direction of linear memory, loads and stores from and to global memory represent a performance bottleneck. This is because vectors that are jointly loaded or stored by all active threads now reside in memory cells which are full image widths apart from each other. For loading new input data, textures provide a good runtime, as they mask memory latencies well by use of their 2-D aware caches. However, writing back global memory still results in an incoherent write operation, which makes this part of the filter more than a factor 3 slower than the one for the perpendicular direction.

In [Ril09], the author suggests to overcome this problem by transposing the problem twice per filter application, such that the kernels for both dimensions run into the same directions. However, such transposition step

is very expensive. As a consequence, the overall framerate for filters with similar data patterns as in our case is stated as 31–100 FPS for images of size 640×480. In Section 3.6.4, we see that this value is about a factor 3–10 below the performance of the aforementioned approach.

Finally, let us discuss the handling of boundaries. As it is described in Section 3.3.5, technical issues such as the preservation of mass are already compensated by the 'parallel' implementation as suggested in [Hal06b]. These problems used to be a fundamental problem of recursive filtering in the literature. However, it remains to implement a suitable strategy to obtain the desired 'reflecting' behaviour around boundaries. Here, we can apply a similar idea as for spatial convolution and assume the kernel to be truncated at $c\frac{\sigma}{h}$ samples. Since a physical mirroring of the image in memory is expensive, we simulate this scenario with a 'warm-up phase' before every filter run. In this phase, we initialise the filter in distance $c\frac{\sigma}{h}$ from the image boundary, and let it first approach this boundary in 'reverse' direction without writing back the result. The intermediate values accumulated in the filter stencil are then used as an initialisation for the actual run. This has the same effect as if the image was mirrored by a boundary of $c\frac{\sigma}{h}$, and cropped to its original size after the filter run is finished. As it was for spatial convolution, a choice of $c \geqslant 3$ yields good results. Unless stated otherwise, we use $c = 5$ in our experiments in order to obtain a best comparability to the other methods. Note that a 'full' mirroring as performed in context of frequency-based convolutions does not yield a benefit in this case. The filter run is by concept finite, such that the arising periodicity cannot be exploited in this case.

### 3.5.6 Iterated (Extended) Box Filtering

Let us now detail on the implementation of traditional and extended box filters, which enjoy a similar algorithmic structure. The box filter with correction then follows straightforwardly as a concatenation of this box filter implementation with a spatial convolution as in Section 3.5.3.

In this implementation, we assume a grid spacing $h = 1$ in both directions, and exploit the separability of the kernel. To the beginning of the program run, we first compute the coefficients to be used during the filtering process:

- For **traditional box filtering**, the only parameter to be computed is the half-length $l$. Since $l$ must be an integer, the supported range of standard deviations $\sigma$ is limited to a sparse set of values (see (3.67)). This is a problem for an implementation, since $\sigma$ is usually given as

an input parameter and does not fulfil the sparsity constraint. As a consequence, we round $\sigma$ to the closest value that can be represented by a box filter and accept the occurring rounding error as a discretisation artefact. By these considerations, $l$ follows from (3.67) by setting $h = 1$ and exploiting that $L = 2l + 1$:

$$l = \text{round}\left(\sqrt{12\frac{\sigma^2}{d} + 1} - 1\right) , \tag{3.89}$$

where round $(\cdot)$ denotes a rounding to the closest integer. If the argument of round $(\cdot)$ already yields an integer, the created box filter with length $2l + 1$ possesses exactly the desired standard deviation.

- In case of **extended box filtering**, we need the 'inner' half-length $l$, as well as the height of the external weight $w$ at position $\pm(l + 1)$. Since the deficiency in the length is absorbed by the free choice of $w$, we do not need a rounding towards the nearest integer. Instead, we now search the largest $l$ which yields a smaller standard deviation than $\sigma$:

$$l = \left\lfloor \sqrt{12\frac{\sigma^2}{d} + 1} - 1 \right\rfloor , \tag{3.90}$$

where the floor function $\lfloor\cdot\rfloor$ denotes a truncation of the decimal part. Based on this $l$, we apply (3.76) and (3.83) to obtain the size of the 'missing' length increment $\alpha$:

$$\alpha = (2l + 1)\frac{l(l + 1) - \frac{3\sigma^2}{d}}{6(\frac{\sigma^2}{d} - (l + 1)^2)} . \tag{3.91}$$

By (3.82), we then obtain $w$ as

$$w = \frac{\alpha}{2l + 1 + 2\alpha} . \tag{3.92}$$

Moreover, we also precompute $\frac{1}{\Lambda} - w =: \hat{w}$ as a second weight, which simplifies the sliding window algorithm:

$$\hat{w} = \frac{1 - \alpha}{2l + 1 + 2\alpha} . \tag{3.93}$$

In both directions, the respective filtering process consist of two parts. To the beginning, the first element in a row is computed by a spatial convolution with the underlying kernel. This step is necessary because the sliding

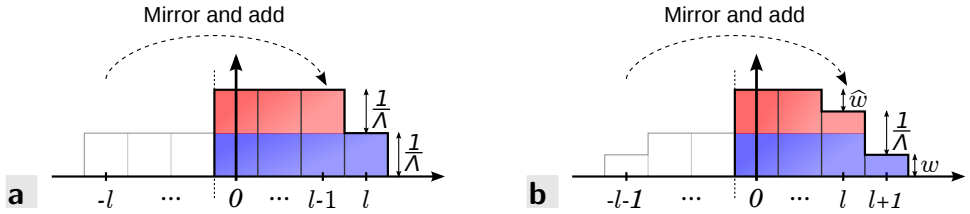



**Figure 3.6**: Initialisation kernel of **a.** the traditional and **b.** the extended box filter. Blue denotes the original contribution right of the central pixel, red visualises the mirrored partition from the left.

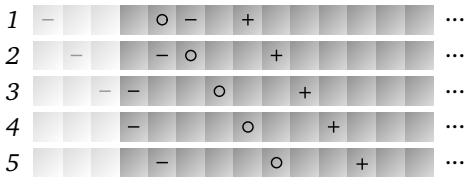



**Figure 3.7**: Implicit boundary handling during box filtering. To update '$\circ$', the solution left of '$\circ$' is reduced by the contribution at '$-$', and increased by the one at '$+$'. The pointer '$-$' is chosen such that it simulates mirroring (light grey).

window approach that is to be started later requires the previously written output as a feedback. Note that since we assume reflecting boundary conditions, there is no need to mirror the signal if we assume an implicitly mirrored kernel. This is shown in Figure 3.6. For a traditional box filter, this amounts to $l-1$ sampling points with weight $2/\lambda$, as well as one sampling point with weight $1/\lambda$. In the case of an extended box filter, the algorithm requires $l-1$ sampling points with weight $2/\lambda$, one with $1/\lambda + w$, and one with $w$.

The actual filtering works by applying the sliding window algorithms from (3.65) and (3.81). These operations are data-parallel in 2-D, where each thread is assigned to one row of data. During these operations, we assume mirroring boundary conditions. However, since an actual mirroring in memory is very expensive, we apply a similar trick as for the 'warm-up phase' during recursive filtering (see above). To this end, we maintain pointers to the locations which are to be read. These pointers are then manipulated where necessary, such that they allow for a boundary handling on-the-fly. For the case of the left boundary, this is shown in Figure 3.7. Here, light grey cells correspond to virtual pixels beyond the image boundary. Note that in this example, the '$-$' pointer mimics a mirroring behaviour of the process during the first three steps.

This idea is simple to realise algorithmically if in addition to the three

pointers, we also store two binary variables which denote the current advance direction of the '−' and '+' pointers. In each step, both pointers are advanced by one pixel into either positive or negative direction, depending on the state of their direction variable. When a pointer runs beyond the image boundary, it is reset and its direction variable is inverted.

Surprisingly, a boundary-aware pointer juggling for *extended* box filtering can be established with only two more value assignments. Assuming the processing of pixel $i$, the two pointers from above are attached to the rightmost sample $(i + l + 1)$ and to the second-but-left sample $(i - l - 1)$. These two pointers again simulate an implicit mirroring and are advanced as above. As an exciting side effect, both values can be used twice over the course of the filter run. The sample at $i + l + 1$ takes the role of one at $i + l$ if $i$ is advanced by 1, and so does the sample at $i - l - 1$ with respect to the one at $i - l - 2$. As a consequence, two out of the four samples can be read out of a register rather than from global memory. This change speeds up the process significantly.

Note that for the sake of notation, we only discussed a 1-D problem in the previous paragraphs. However, keep in mind that due to the separability of the 2-D process and the inherent data-parallelism per dimension, this setting immediately carries over to the 2-D case. In this moment, all previously described scalar loads and stores map to fast loads and stores of full vectors, which makes the process efficient. Similar to the case of recursive filters, we choose textures to load data during the filter run along the coherent memory direction, and direct accesses to global memory otherwise.

This finishes our excursion on the details of efficient GPU-based algorithms for linear diffusion. In the next part, we evaluate the properties of these algorithms, and compare their quality and their runtime efficiency.

## 3.6 Experiments

On the next few pages, we evaluate our algorithms with respect to the quality of their results and the time required to obtain these solutions. Some of our algorithms are parameter-free, which makes the evaluation straightforward. We can simply measure the quality and runtime on representative sample images, and take these measurements as a basis of comparison. However, other algorithms such as implicit convolution or box filters, possess additional parameters which steer a tradeoff between quality and runtime. In order to compare them to the remaining methods, we must first optimise their parameters. To this end, our experiments are grouped into four parts. After we answer the question for a reasonable ground truth to compare

against, we optimise free parameters for all of our approaches to yield a solid basis for comparison. In the following benchmark, we compare the approximation quality of all approaches against each other. The final and most important series of experiments aims at the runtime of all algorithms. We compare these GPU-based methods against each other, and against their counterparts on the CPU. This gives us important insights about the absolute runtimes, but also about the scaling behaviour of our algorithms.

### 3.6.1 Ground Truth

Before we begin with the evaluation and comparison of the presented algorithms, let us address the fundamental question of a reliable reference solution for our experiments. As it turns out, this question is less trivial than it seems at first glance. Our setup suffers from a number of inaccuracies such as the imprecision of floating point arithmetics. Even if we use very high quality settings for algorithms such as explicit diffusion or spatial convolution, these impairments cause the solutions obtained from both experiments to be different.

In order to get a feeling for the effects occurring in this context, let us perform an experiment on the CPU. On a 1-D unit grid, we create a signal of length 2001 which has a unit peak at 1000, and which is zero elsewhere. We fix the stopping time to $T = 1000$, and perform explicit and implicit diffusion with different time step sizes, as well as spatial Gaussian convolution with truncation $c = 10$. In the latter case, we mirror the signal sufficiently often at its boundaries such that the full support of the kernel lies within a valid range. For the implicit discretisation, we use a Thomas algorithm to solve the equation system arising for each time step [vR09]. Note that we use sequential hardware and a 1-D signal in this experiment, such that this algorithm can be efficiently applied.

Our hope is that the explicit and the implicit method yield a similar solution if they are configured with 'suitable' parameters. We can then call this solution a ground truth of the process, and can be confident that a similar configuration gives equally reliable results for more complex inputs. Spatial Gaussian filtering, in contrast, is likely to yield a different solution. This is because of discretisation artefacts that cause the continuous solution not to carry over to the discrete case (see [Lin90] for details). Nevertheless, Gaussian filtering possesses a very simple algorithmic structure which helps it to dampen rounding errors to a minimum. Depending on the impact of rounding errors on the quality of the result, this property makes Gaussian filtering an interesting alternative to the actual discretisations of the linear diffusion equation. In addition to this algorithmic alternative, we compare

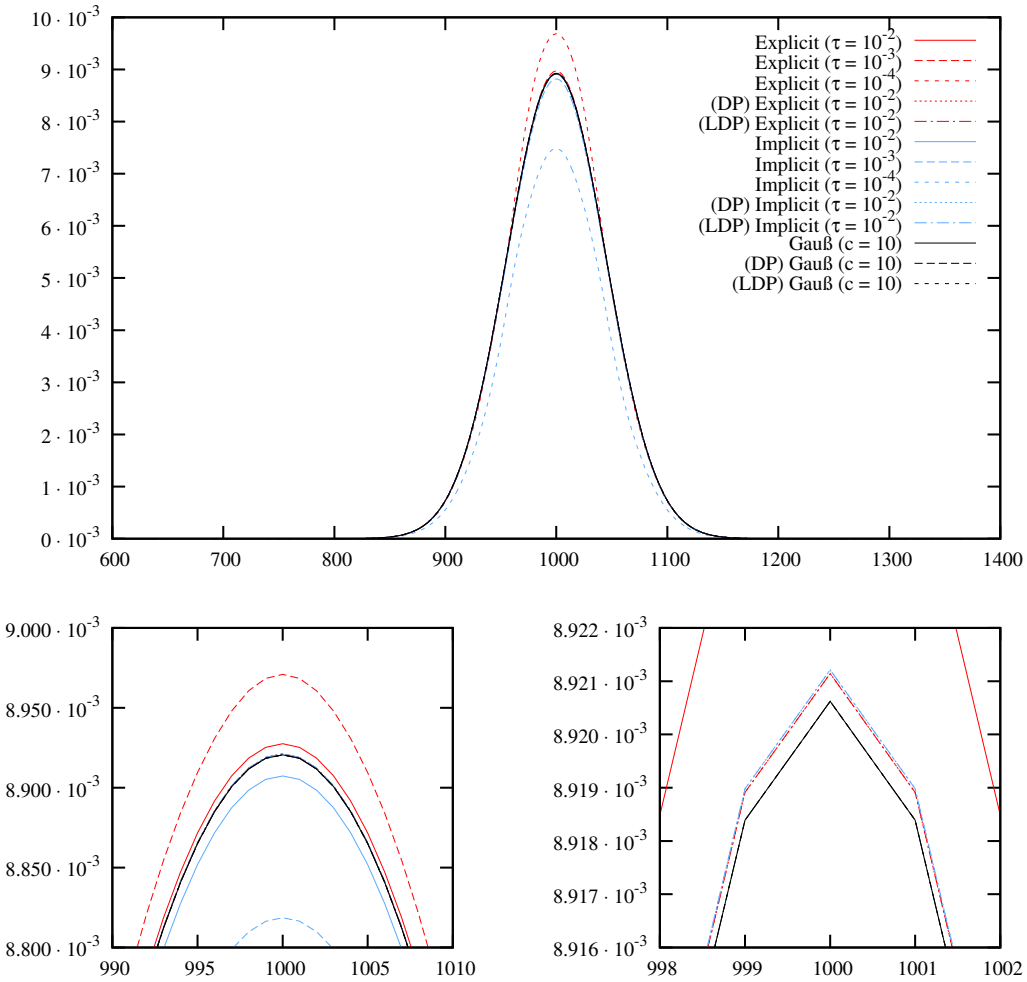



**Figure 3.8**: Linear diffusion with stopping time T=1000 on a 1-D signal containing a unit peak centred at 1000. Computations are performed in double (DP), long double (LDP), or single precision floating point arithmetics (else).

the results of all algorithms in single, double, and long double floating point arithmetics. This gives us a feeling for the impact of rounding errors on the whole process.

The results of this experiment are shown in Figure 3.8. In single precision arithmetics, we apply both the explicit and the implicit solver with time step widths $\tau = 10^{-2}$, $\tau = 10^{-3}$, and $\tau = 10^{-4}$. Since we use a first-order discretisation in time, we expect smaller approximation errors the closer $\tau$ approaches zero. However, the converse happens. While for $\tau = 10^{-2}$, the solutions for both schemes are less than $2.5 \cdot 10^{-5}$ apart, their maximal distance grows to more than $2.0 \cdot 10^{-3}$ for $\tau = 10^{-4}$. This

error is significant, since it affects the absolute value by more than 20%. It is a result of rounding errors which accumulate over the course of the process [GO96]. Smaller time step widths cause more iterations, and each iteration causes a certain rounding error. As a consequence, the absolute error increases due to rounding, although numerics promises a dampening of the error.

In order to account for these inaccuracies, we re-perform the experiment for $\tau = 10^{-2}$ in double and long double arithmetics. As it turns out, both methods yield indistinguishable results up to the 9th decimal place. Consequently, their graphs in Figure 3.8 are superposed. Compared to the single-precision case, the implicit and the explicit result are much closer to each other. Their maximal distance shrinks to $6.7 \cdot 10^{-8}$.

Let us now compare the result of Gaussian filtering against these solutions. As a first striking property, we see that spatial convolution with a Gaussian is almost invariant under rounding inaccuracies. The three graphs for single, double, and long double precision are superposed. This is a result from the limited rounding error taking place in this operation. A stopping time of 1000 corresponds to a standard deviation $\sigma \approx 45$ on a unit grid. Considering a truncation $c = 10$, each value of the solution is computed as a weighted sum over 895 samples. This value is in striking contrast to $10^{10}$ weighted additions per output sample for explicit diffusion, or even more for implicit diffusion. For a stopping time of 1000 and a time step $\tau = 0.01$, these discrete diffusion algorithms require 100 000 iterations. Moreover, implicit diffusion requires one run of the Thomas algorithm per iteration, which is additionally prone of accumulating rounding errors.

However, we can also see that Gaussian convolution does not exactly yield the same results as the discretisations of the diffusion equation. For the analysed stopping time $T = 1000$, the maximal error between the average of the diffusion processes and Gaussian filtering amounts to $4.29 \cdot 10^{-6}$. There are different interpretations for this observation. On the one hand, it could be related to the fact that although the results obtained with double precision arithmetics are more *precise*, they are not *accurate* with respect to the optimal solution of the discrete diffusion process. The discrepancy between precision and accuracy for floating point arithmetics is impressively shown in [Rum88]. On the other hand, it is more likely that the observed error between these algorithms is related to the inadequacy of Gaussian filtering as an algorithm for discrete linear diffusion (see [Lin90]). An indicator arguing in favour of this hypothesis is given by a reference experiment for a stopping time $T = 10$. In this case, the error grows to $5.74 \cdot 10^{-4}$. This behaviour for small stopping times is also reported in [Lin90, Footnote 14]. An alternative could in these cases be given by a kernel as in [Nor60].

To this end, we can draw several conclusions from this experiment. On GPUs, we are interested in a low runtime. Since double precision arithmetics slows down the process significantly (see [NVi11b]), there is little option other than using single precision floating point arithmetics. Moreover, our main focus is on large diffusion stopping times. This is a natural consequence from the opportunity of fast computations on a GPU. Using this computing power, we can in the same time process much larger images than on a CPU. This allows us to increase the resolution of an image to provide a higher accuracy for a superordinate algorithm. In this case, we also require higher stopping times $T$, since $T$ scales inversely to the grid spacing $h$. Also from a technical perspective, large stopping times are more interesting than smaller ones. Since input data required to process one pixel does not fit into fast on-chip memory, we require algorithms that reduce the memory intensity rather than the computational intensity. Such requirement is usually the more challenging case on GPUs.

As a result from these considerations, spatial Gaussian filtering with a truncation such as $c = 10$ provides the best ground truth for our experiments. However, keep in mind that for small stopping times, this ground truth contains a systematic error which makes results less reliable.

### 3.6.2 Parameter Configuration

In the following, we optimise free parameters for the presented algorithms. Since these parameters all represent trade-offs between accuracy and speed, we aim at configurations which still yield a 'sufficient' quality, but which do not significantly squander runtime of the process. As a basis for comparison, we use the *Boat* image shown in Figure 3.9.

#### Gaussian Convolution

Let us adhere to spatial Gaussian convolution for a moment, and investigate the influence of the truncation parameter $c$ on the quality of the results. For an optimal runtime, we require $c$ to be as small as possible. It is the goal of this experiment to find a lower bound for $c$ which is still sufficient for real-time applications.

From a theoretical analysis in the continuous 1-D setting, we expect a value of about $c \approx 3$ to be sufficient. In such case, we truncate about $2.7 \cdot 10^{-3}$ of the mass of the kernel. By the normalisation in (3.23), this error is distributed over the full support of the kernel. For larger $c$, this error even decreases further, such as $6.3 \cdot 10^{-5}$ for $c = 4$, or $5.7 \cdot 10^{-7}$ for $c = 5$. By separability, the same approximation errors carry over to the $n$-D

**Figure 3.9**: *Boat* test image from the USC-SIPI image database (`http://sipi.usc.edu/database/`), 512×512 pixels. This image is used to compare algorithms presented in this chapter.

case. As a measure for similarity, we apply the *mean square error*, which is given by [LK11]

$$MSE(\boldsymbol{a}, \boldsymbol{b}) \;=\; \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{a}_i - \boldsymbol{b}_i)^2 \,, \tag{3.94}$$

where $N$ describes the number of pixels. The vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are more similar to each other, the smaller the MSE between them is.

Figure 3.10 shows the MSE between the result and the ground truth, depending on the choice of the standard deviation $\sigma$. The colour coding of the plot reflects our intuitive interpretation of the quality of the filtering result. Green corresponds to an MSE that is smaller than about 0.2. Such errors are typically invisible in actual applications. The more the colour fades to red and black, the higher is the error. Such deviations from the exact solution can often be spotted, in particular in a direct comparison with the exact solution. Our expectation that a truncation $c = 3$ is sufficient to dampen the error significantly is confirmed by the real-world experiment. A much lower setting for $c$ does not make sense, the error already grows to about 1.0 for $c = 2$. Errors of this magnitude are in general even visible by the human eye. We also notice that apart from very small standard deviations $\sigma$, potential discretisation errors have no significant impact on

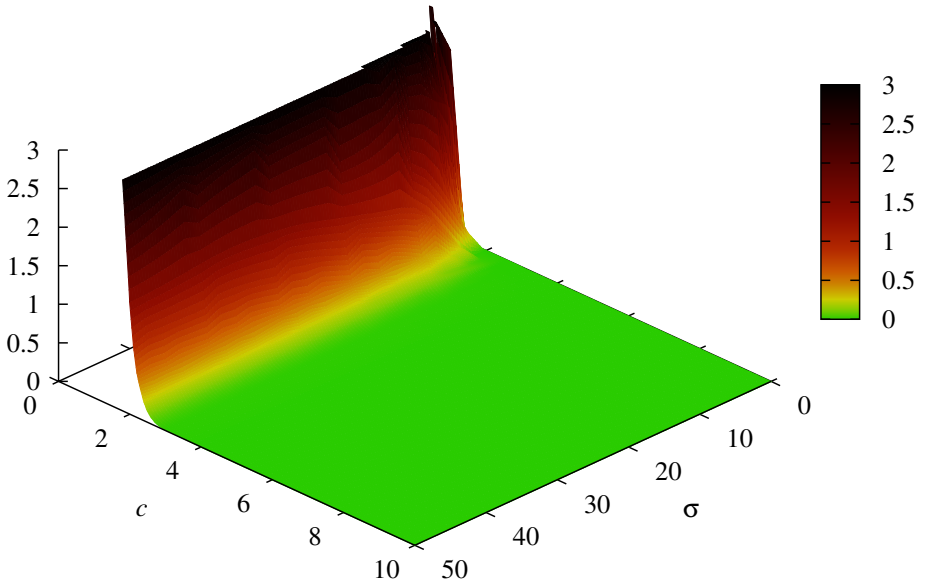



**Figure 3.10**: Mean square error for spatial Gaussian filtering on *Boat*, 512×512 pixels, depending on standard deviation $\sigma$ and cutoff parameter $c$.

the optimal choice of the truncation parameter. The graph is sufficiently smooth. The reason that we obtain even lower errors for small $\sigma$ is related to the rounding of the absolute truncation $c\frac{\sigma}{h}$ to full integers. In Section 3.6.3, we perform a more detailed analysis on a logarithmic scale, and a comparison against other techniques. Besides the choice $c = 3$, we additionally evaluate for the 'high-quality' setting $c = 5$ which gives results that are almost indistinguishable from our ground truth.

### Implicit Diffusion

Our implicit diffusion algorithm possesses two free parameters which steer the quality but also the runtime of the process. The first parameter of this kind is the time step size $\tau$ for each implicit step which implies the number of 'outer' iterations $k_{\max}$. Equivalently, we can fix the latter to a certain number and compute $\tau$ from the stopping time $T$ as $\tau = {}^{T}/_{k_{\max}}$. Small $\tau$ in general lead to better solutions. The second parameter $l_{\max}$ controls the approximation quality of each individual step of size $\tau$. It describes the number of iterations of the internal Jacobi solver. Apart from accumulative errors, we can state that more iterations in general lead to a better approximation.

From the perspective of runtime, we also do not want to choose $k_{\max}$ and $l_{\max}$ too large. Their product yields the absolute number of steps, and

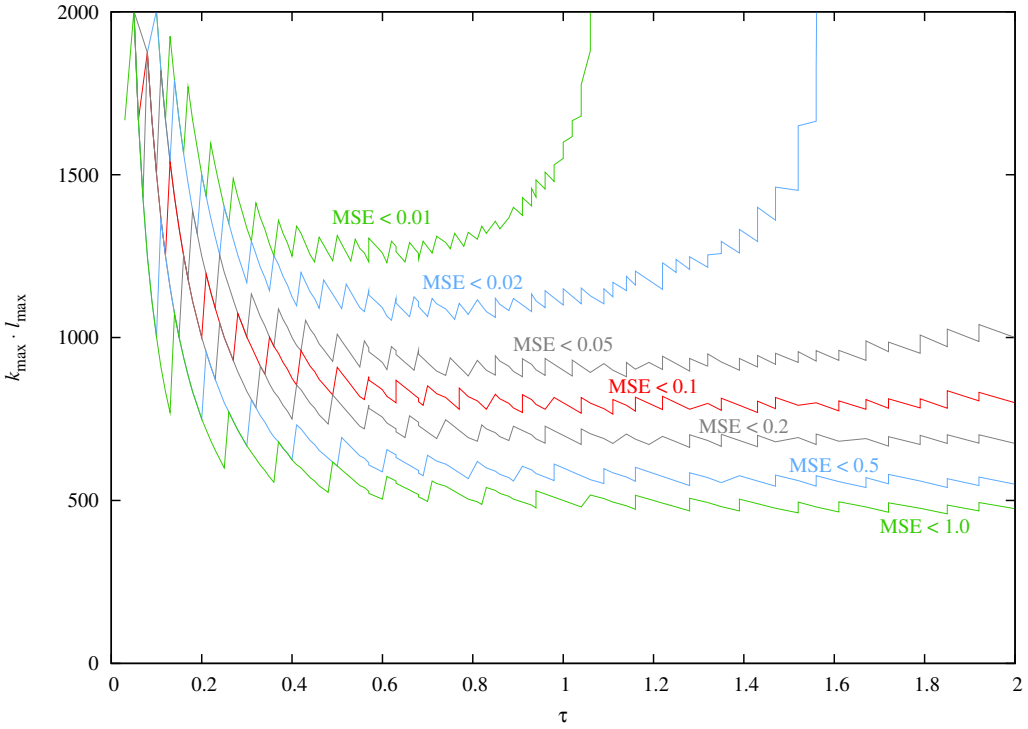



**Figure 3.11**: Minimal number of iterations ($k_{\max} \cdot l_{\max}$) of the Jacobi solver for implicit diffusion with given $\tau$ such that a desired MSE is reached. As a basis for comparison, we use a stopping time $T = 50$ on *Boat*.

determines the total runtime of the process. The aim of this experiment is to jointly optimise these two parameters with respect to a desired maximal error. This setting then serves as a basis of comparison in runtime experiments later in this chapter.

Using the *Boat* image from Figure 3.9, we choose a stopping time $T = 50$, vary $\tau$, and run implicit linear diffusion with different 'outer' and 'inner' steps $k_{\max}$ and $l_{\max}$. For a collection of selected MSEs, we visualise the minimal number of total steps required to yield a solution with an equal or smaller error.

This is shown in Figure 3.11. If we are interested in moderate errors smaller than about 0.5, a large interval of time step sizes yields agreeable results in about the same runtime. This changes the more we restrict the maximal admissible error. For a bound of 0.01, for example, only time steps in the interval $\tau \in [0.4, 0.8]$ yield results with a minimal number of steps.

As a consequence of this experiment, we fix the time step for all following experiments to $\tau = 0.6$ and set the number of inner steps to $l_{\max} = 13$. This is motivated from different considerations. For this particular example with

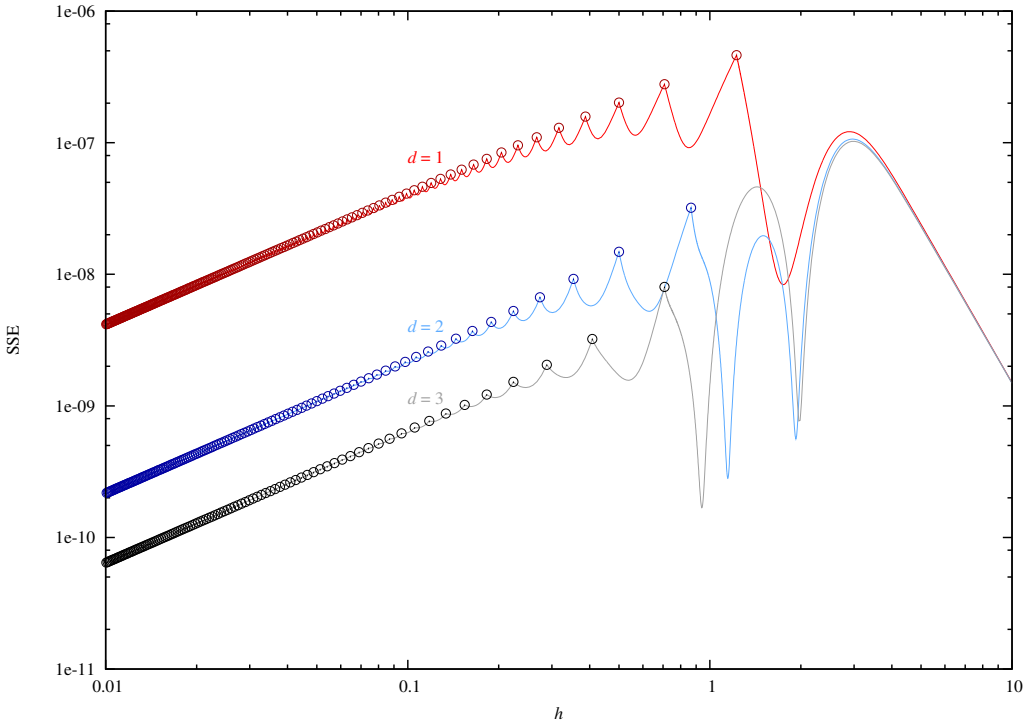



**Figure 3.12**: SSE between impulse responses of box filters and corresponding Gaussians for $\sigma = 1$ under varying grid spacings $h$. Lines correspond to extended box filters, circles to traditional box filters.

$T = 50$, we obtain an MSE that is lower than 0.05 with a total number of 1092 steps. Since we target at speed, this setup seems to be a reasonable tradeoff. Moreover, there is good evidence that our insights carry over to other input images and other stopping times. In this context, $l_{\max}$ is the most uncritical setting, as it only depends on $\tau$ and the desired quality. On the other hand, even the optimal choice for $\tau$ seems to be relatively stable under varying stopping times, unless $T$ is chosen close to 0. Other experiments show that *Boat* possesses characteristics that are typical for many real-world images, such as many high and low frequencies, and a high value range. To this end, we can safely assume that $\tau = 0.6$ serves most purposes. If for real-world applications, a smaller error or smaller runtime should be desired, this tradeoff can easily be established by an adaptation of $l_{\max}$. The resulting deviation in runtime can then be read out of Figure 3.11.

### Consistence of Traditional and Extended Box Filtering

Let us now focus on box filters. As the first experiment in this series, we evaluate the approximation quality of the extended box filter and its relation to traditional box filtering. From Theorem 3.3, we know that the extended box filter comes down to the simpler traditional box filter if its length is an odd integer. As a consequence, we expect its approximation error to a linear diffusion process to continuously interpolate the error of the traditional box filter sampled at locations where it is defined. This consideration leads immediately to the question of the quality of this interpolation. We expect it to be much better than a simple rounding as proposed in Section 3.5.6 for traditional box filters.

As a representative example for the behaviour of extended box filter, we observe its impulse responses for a given number of iterations $d$. In close correspondence to the theoretical motivation from Section 3.4.2, we vary the spatial grid width $h$ in a simple 1-D setting and normalise the standard deviation $\sigma$ to 1. As a basis for comparison, we use a signal with a single unit peak in the central sample. The length of this test signal is chosen large enough such that potential wrap-around errors are below the limit of machine precision.

Concerning the ground truth for this experiment, it would be nice to have solutions of continuous box filtering with the given number of iterations $d$. However, such results do not exist on a discrete grid. Computing them analytically and sampling them on the target grid is no valid choice either, since this procedure introduces new errors. In fact, the extended box filter by construction already represents one such discretisation.

Instead, we use the fact that a sufficiently large number of iterations with a box filter leads to a Gaussian filter and compare the box filtering result to this desired solution. While it is clear that a higher number of iterations $d$ leads to an absolutely better solution, this solution suffices to provide a direct comparison of extended and traditional box filters with respect to their relative errors. As a measure for similarity, we use *sum of squares errors* (SSEs):

$$\mathrm{SSE}(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{N} (\boldsymbol{a}_i - \boldsymbol{b}_i)^2 \,, \tag{3.95}$$

where $N$ is the number of samples in $\boldsymbol{a}$ and $\boldsymbol{b}$. Note that we do not normalise by the number of samples. This makes sense because we expect the impulse responses to be well localised, and allows to abstract from the actual length of the signal. It is always chosen large enough to fit a Gaussian with truncation $c \geqslant 10$.

Figure 3.12 shows the SSE for box filters with $d \in \{1, 2, 3\}$ iterations against the ground truth. While the results produced by the extended box filter are visualised by lines, the results for traditional box filters are overlaid by circles. The rightmost circles in each colour correspond to a $B_3^d$, i.e. to a $d$-times iterated box filter of kernel length $3h$. Let us first focus on the measurements to the left of these points. These results are obtained with box kernels that are larger than $3h$. As expected, the extended box filter possesses the same error as the traditional box filter when it comes to setups for which the latter is defined. However, in between these configurations the extended box filter approximates the Gaussian much better than we would expect from a simple linear interpolation of the given samples. This is a consequence of the characteristic shape of the extended box kernel. Its additional weights of height $hw$ at both sides cause a much more smoothing behaviour than expected from a box filter. Such behaviour acts in favour of Gaussian filtering, and is consequently reflected in a lower error.

For all box filters, the error drops for $h \to 0$. This is because we have an increasing number of samples as $h$ approaches 0, while the convolution kernels still possess unit sum. As a result, we obtain low but elongated impulse responses. Consequently, the measured error between consistent kernels also approaches 0.

Let us now discuss the case $h < \Lambda < 3h$, which corresponds to the partition to the right of the circles in Figure 3.12. If we recall that all filters approach the identity for $h \to \infty$, we can easily explain the asymptotic behaviour of the error. The filters hardly touch the signal in this case, such that we actually compute the error between two copies of the input. The dominant oscillations for $\Lambda \to 3h^-$, in contrast, deserve more attention. By Theorem 3.5, we can lead them back to approximation errors in second-order finite difference approximations. The error for this approximation of the Laplacian takes the form

$$\frac{h^2}{12} \frac{\partial^4 u}{\partial^4 x} + \mathcal{O}(h^3)\,, \tag{3.96}$$

where $u(x)$ denotes the 1-D signal [Saa03]. For large $h$, this term can become significant, which explains the maxima in the plot. However, we also observe striking minima which represent errors of a magnitude that we do not expect in these orders of $h$. As it turns out, these minima occur because for particular configurations, the extended box kernel closely resembles the corresponding Gaussian. For example, consider the minimum for $d = 1$. For $h = \sqrt{3}$, we set the length of the extended box kernel to $\Lambda = {}^3\!/_2 h$ to obtain a standard deviation of $\sigma = 1$. As it turns out, the

arising kernel closely resembles a discrete Gaussian truncated at $c = 1$:

$$[K_1]^1 (0 \cdot \sqrt{3}) \approx 0.69144 \qquad [K_1]^1 (1 \cdot \sqrt{3}) \approx 0.15428 \tag{3.97}$$

$$\boldsymbol{E}_{1.5} (0 \cdot \sqrt{3}) \approx 0.66667 \qquad \boldsymbol{E}_{1.5} (1 \cdot \sqrt{3}) \approx 0.16667 \tag{3.98}$$

The left column in (3.97)–(3.98) refers to the central weights of the two 3-point stencils, the right column to the outer weights. The SSE between these configurations is $9.2 \cdot 10^{-4}$ – which agrees with our observation from Figure 3.12. By a similar argument, we have two minima for the measurements $d = 2$ and $d = 3$. In this case, the iterated extended box filter matches discrete Gaussians with truncation $c = 1$ and $c = 2$.

### Box Filters: Rotational Invariance

Linear diffusion is *rotationally invariant*. This property follows from the shape of the (continuous) Laplacian, and states that a filtering of a rotated image is equivalent to a rotation of the diffused original image by the same angle. Rotational invariance is also an important property of discrete implementations, since it confirms that no grid-dependent discretisation artefacts are introduced. Most filters presented in this chapter either solve the diffusion equation directly in 2-D, or they use the separability of Gaussian filtering. If these approximations are accurate, the arising filter is rotationally invariant by construction.

Box filters are prone to violate this requirement, because the elementary case $d = 1$ has a square-shaped support. As a consequence, their rotational invariance can only arise from a high number of iterations. In this experiment, we want to determine the minimal number of iterations $d$ such that this property is fulfilled.

Figure 3.13a shows the image of a black circular disc with diameter 768 pixels on a plane of size 1024×1024 pixels. The size of this image has been chosen large enough to smooth out potential discretisation artefacts arising from the input. A rotationally invariant algorithm is supposed to transport grey values perpendicular to the edge, i.e. to propagate information in concentric circles. Since small variations in the grey value are almost invisible to the unaided eye, we visualise the results with their isolines. Two neighbouring isolines enclose an interval of $^{15}/_{255}$ of the full grey scale. For the ground truth shown in Figure 3.13b, we see that the rotational invariance property is perfectly fulfilled. All isolines describe concentric circles. This is different for one iteration of a box filter. Here, the isolines are distorted towards an axis-aligned structure, as can in particular be seen for the outermost line. Moreover, in a direct comparison with the ground truth we also

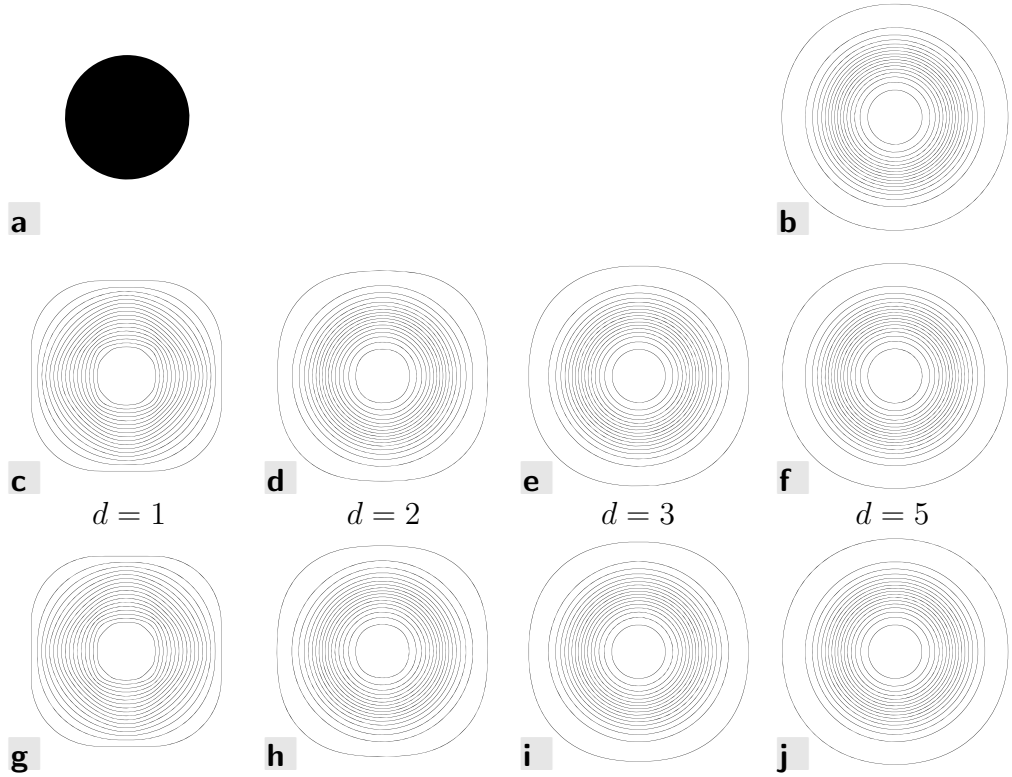



**Figure 3.13**: Rotational invariance of box filtering in comparison to Gaussian convolution. Results show isolines of a filtering of **a.** the *Disc* image, 1024×1024 pixels with $\sigma = 80$. **b.** Ground truth. **c.**–**f.** Traditional box filtering. **g.**–**j.** Extended box filtering.

observe that gradients are largely estimated wrongly. While the ground truth features different distances between isolines, they are almost equispaced for the box filter. Both problems disappear the more iterations we prescribe. With $d = 3$, the result already looks perceptually similar to the ground thruth, although the grid bias is still noticeable in the distance between the two outer isolines. The solution for $d = 5$ is almost equivalent to the ground truth, and does not contain any of the aforementioned artefacts.

As it turns out, the results for the extended box filter are indistinguishable from the ones for the traditional box filter. Consequently, it contains visual inadequacies for $d = 1$ which are completely resolved for $d = 5$ iterations. This indicates that the new discretisation involved in this case does not introduce unexpected new artefacts when the filter is applied to multi-dimensional settings.

Hence, in view of the rotational invariance of the process, already 3 to 5

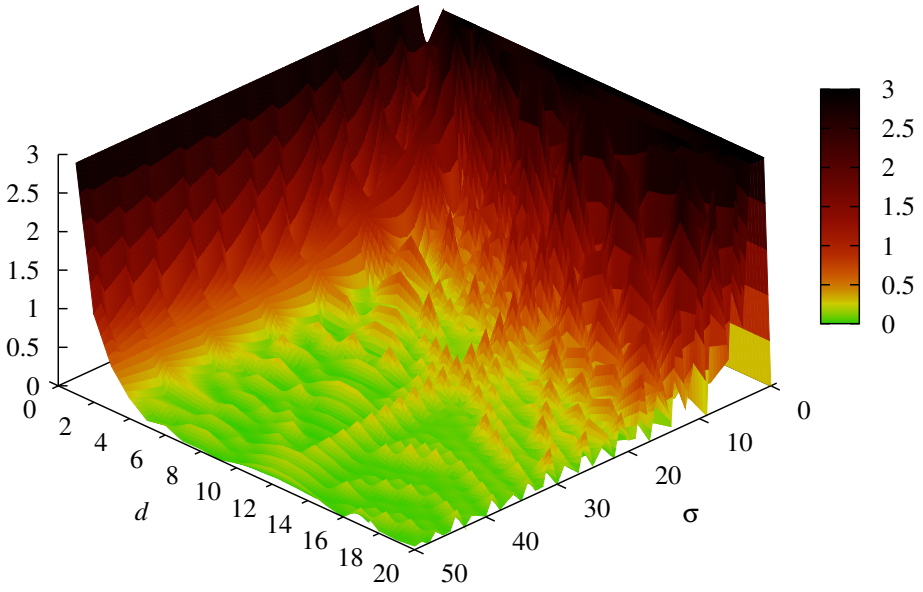



**Figure 3.14**: Mean square error for traditional box filtering on *Boat*, 512×512 pixels, depending on standard deviation $\sigma$ and iterations $d$.

iterations of a box filter suffice to yield very good results. In the following, we check whether this insight carries over to the approximation quality on real-world data, or whether we require even more iterations to yield a low approximation error.

### Box Filters: Quantitative Analysis

In this experiment, we analyse two different aspects of box filtering. In the first part, we measure the impact of the number of iterations $d$ on the quality produced by the different box filtering algorithms. In the second part, we then keep $d$ fixed and perform a direct comparison of the three methods. The latter helps us to decide which discrete instance of a box filter is best suited for a particular problem, while the first one yields insights about the optimal configuration of this instance.

Let us begin with a measurement of the MSE for traditional box filtering using $d \in \{1, \ldots, 20\}$ iterations. As a basis for comparison, we again apply the *Boat* image from Figure 3.9, and apply a standard deviation $\sigma$ ranging from 0 to 50. Whenever there is no box kernel with the desired standard deviation $\sigma$ available, we choose either the next smaller or next larger kernel, depending on which provides the better approximation (see (3.89)). As a reference solution, we use again a spatial Gaussian filtering with $c = 10.0$.

The mean square error between the result and the reference solution is visualised in Figure 3.14, using the same colour scheme as before. We observe high errors for small $d$, but also for fairly small $\sigma$. The first fact is immediately clear if we recall that very few iterations of a box filter do not suffice to approximate a Gaussian well (see (3.60) and [Wel86]). This effect is well understood in the continuous setting, and carries over to the discrete case.

In contrast, the high errors for small $\sigma$ and larger $d$ are more interesting. Each of the peaks corresponds to an interval of standard deviations which cannot accurately be represented. It is easy to explain why these effects occur most dominantly for small standard deviations. Let us refer once more to Figure 3.2 which explains that a traditional box filter is in general either higher and narrower than its continuous prototype, or lower and wider. Without loss of generality, let us assume the latter setting. Then, the rounding step from (3.89) causes the traditional box filter to be at most one sample longer than necessary. This sample has width $h$ and height $h/L$ (by (3.64)). Since the kernel is normalised to unit sum, all weights that are placed into the surplus sample are missing over the length of the kernel, such that the total error is doubled. To this end, we obtain a maximal absolute approximation error on the kernel of

$$\mathrm{AE}_{\max} = 2h\frac{h}{L} = 2h\frac{h}{h(2l+1)} = 2h^2\sqrt{\frac{d}{12\sigma^2 + dh^2}}\,, \tag{3.99}$$

where the third equality follows from (3.67). If in (3.99), we let $\sigma \to 0$ or $d \to \infty$, the maximal AE approaches $2h$. Whenever a box kernel is convolved with an image, its AE is introduced to every pixel, scaled by the image grey value at this point. This explains the significant errors depicted in Figure 3.14.

Let us now repeat this experiment for a box filter with correction. The MSE on *Boat* is shown in Figure 3.15. We observe that the error almost vanishes for $d > 3$, independent of the standard deviation $\sigma$. This is bought by relatively large correction steps by Gaussian filtering, in particular when it comes to small $\sigma$ and large $d$. Note that independent of $d$, filtering results with a very small standard deviation can accurately be reproduced. This is because the box filters $B_1$ used in this case simply equal the identity. In such cases, we should no longer speak of box filtering, but attribute the effect to the Gaussian filter.

Figure 3.16 shows the corresponding experiment with an extended box filter. The plot is very similar to the one for box filters with correction, except for an additional error for very small standard deviations $\sigma$. This

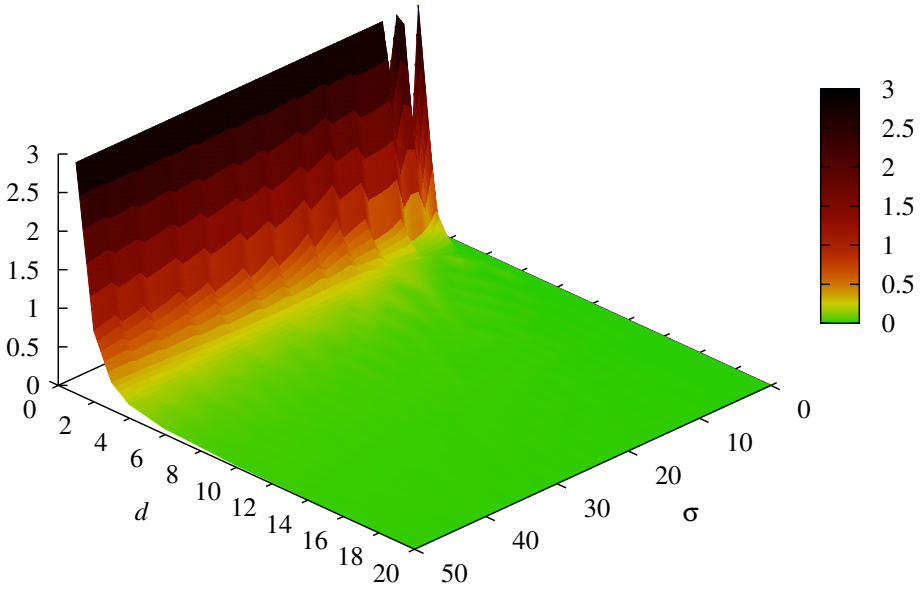



**Figure 3.15**: Mean square error for box filtering with correction on *Boat*, 512×512 pixels, depending on standard deviation $\sigma$ and iterations $d$.

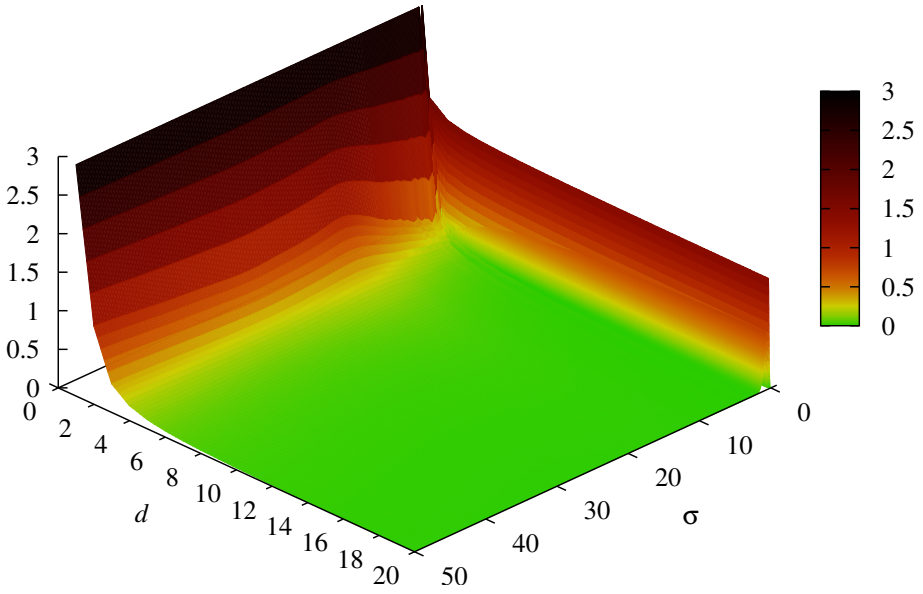



**Figure 3.16**: Mean square error for extended box filtering on *Boat*, 512×512 pixels, depending on standard deviation $\sigma$ and iterations $d$.

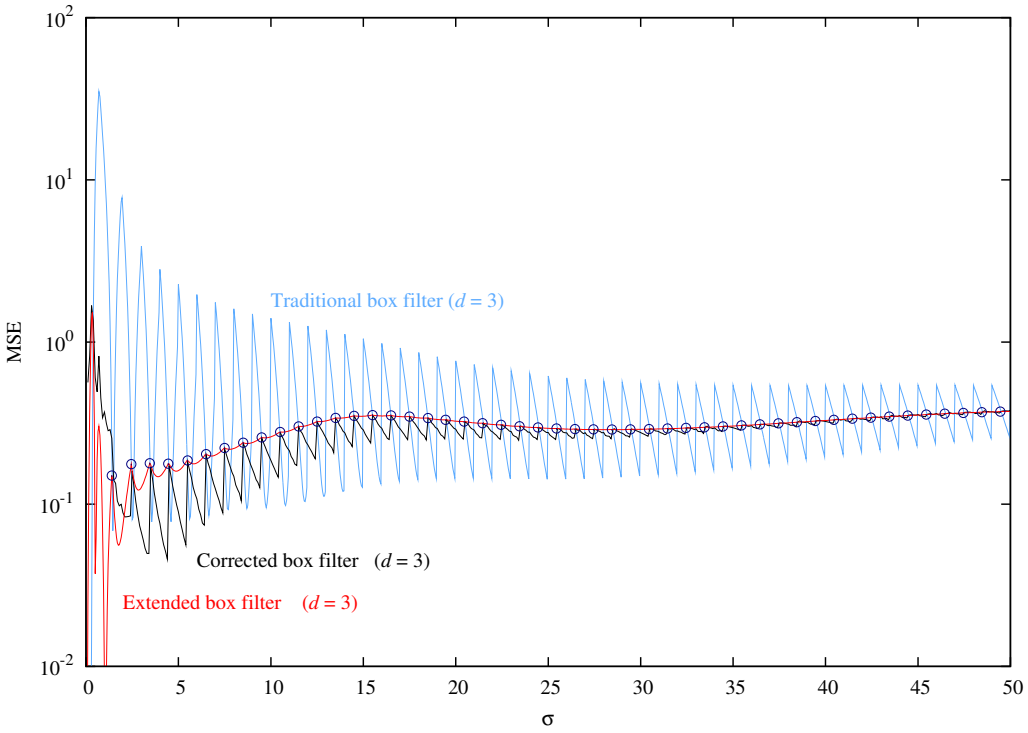



**Figure 3.17**: Mean square errors of different box filtering algorithms for $d = 3$ iterations under varying standard deviations $\sigma$, on *Boat*, 512×512 pixels. Circles refer to positions where the traditional box filter is defined.

error has already been discussed on pages 65ff in context of 1-D extended box filtering, and is caused by approximation errors for $\sigma/h \to 0$ (see also Figure 3.12).

Up to this point, we have already convinced ourselves that both corrected and extended box filtering are superior to traditional box filtering when it comes to real-world applications, since significant errors are dampened sufficiently well. Let us now analyse the differences between these methods in detail. To this end, we plot the errors for the exemplary case $d = 3$ on a logarithmic scale. Figure 3.17 shows the results of this experiment. It depicts traditional box filtering in blue, the corrected box filter in black, and the extended box filter in red. In addition, all locations on which the discrete box filter is exactly given are also marked with dark blue circles.

Surprisingly, this experiment does not yield the outcome expected from the theoretical analysis in Section 3.4.2 and the experiment from pages 65ff. In general, the error of all methods seems to approach a constant rather than

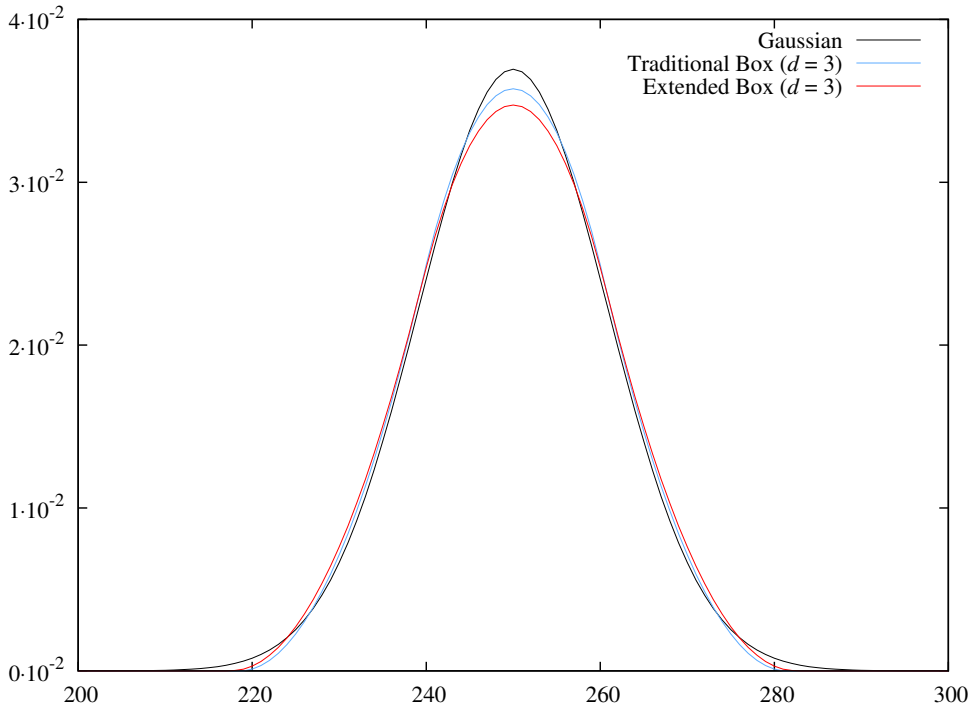



**Figure 3.18**: Impulse response of box filtering for $\sigma = 10.8$ and $d = 3$ on a unit peak centred at 250, and result of Gaussian filtering.

dropping off towards 0 as $\sigma$ approaches $\infty$. So far, this observation does not contradict the insights obtained on pages 65ff. While the absolute error for each impulse response drops as $\sigma$ is increased, the effective support of each impulse response grows by about the same amount. Consequently, smaller errors are superposed in a larger area, such that the overall error to be measured remains about constant. If, in contrast, we choose $\sigma$ much larger than 50, the solution approaches a uniform image with the same average grey value as the original. Given that no methods accumulate numerical inaccuracies, all errors approach 0.

Within this slightly modified setup, we also find that both the extended and the corrected box filter behave consistent to the traditional box filters at defined positions (marked by circles). In case of the extended box filter, the lower error in between these samples is easily explained by the smoothing behaviour of the external weights. The corrected box filter possesses a lower error the higher its partition of Gaussian filtering is.

Most unexpectedly, however, the traditional box filter partially manages to yield significantly lower errors than both modified techniques. This happens when the desired standard deviation is slightly higher than those of

the 'most similar' traditional box filter. In such cases, it seems to pay off to use the next smaller traditional box filter rather than a kernel with the correct standard deviation. The error even drops significantly compared to the 'consistent' case where the standard deviation of the iterated box filter matches the one of the desired Gaussian.

In order to understand this anomaly, let us compare the impulse responses of the filters for a setting in which the traditional box filter is more accurate than the extended one. For a standard deviation of $\sigma = 10.8$ and $d = 3$ iterations, this is shown in Figure 3.18. Although all kernels have unit sum and the same standard deviation, the result for the extended box filter has a lower maximum. The mass missing in this region is instead distributed over the rising and falling slopes of the function. This explains the high error. The 'smaller' traditional iterated box filter does better. Because its variance is smaller, the impulse response becomes narrower. Since by construction, it still possess unit sum, the central weight is higher. To this end, the smaller traditional box filter draws advantage from a better approximation of the *shape* of the kernel rather than from a more accurate *standard deviation*.

For the extended box filter, in particular, this surprising observation is not only a challenge but also a chance. The starting point for this method is the traditional box filter, which we canonically assume to yield best results when the standard deviation is exactly fulfilled. However, we found that these points do *not* coincide with the minima in the error plot (cf. Figure 3.17), but rather with some intermediate error. If we manage to compute these minima, we can attach the extended box filter to this new process and improve the approximation error where the traditional box filter fails. However, finding the exact positions for these minima in a closed form expression without a direct comparison of impulse responses can be tedious and remains to be addressed in future work.

Let us conclude this experiment with a brief summary of obtained insights. Both the extended and the corrected box filter possess a lower *guaranteed* error bound than a traditional box filter. For many applications, the decrease of the maximal MSE by more than one order of magnitude causes the results of extended and corrected box filtering to yield visually much more pleasing results than traditional box filtering. However, if we are instead interested in the absolute minimal error, we should be aware that under special circumstances, traditional box filtering yields better results than either of the other methods.

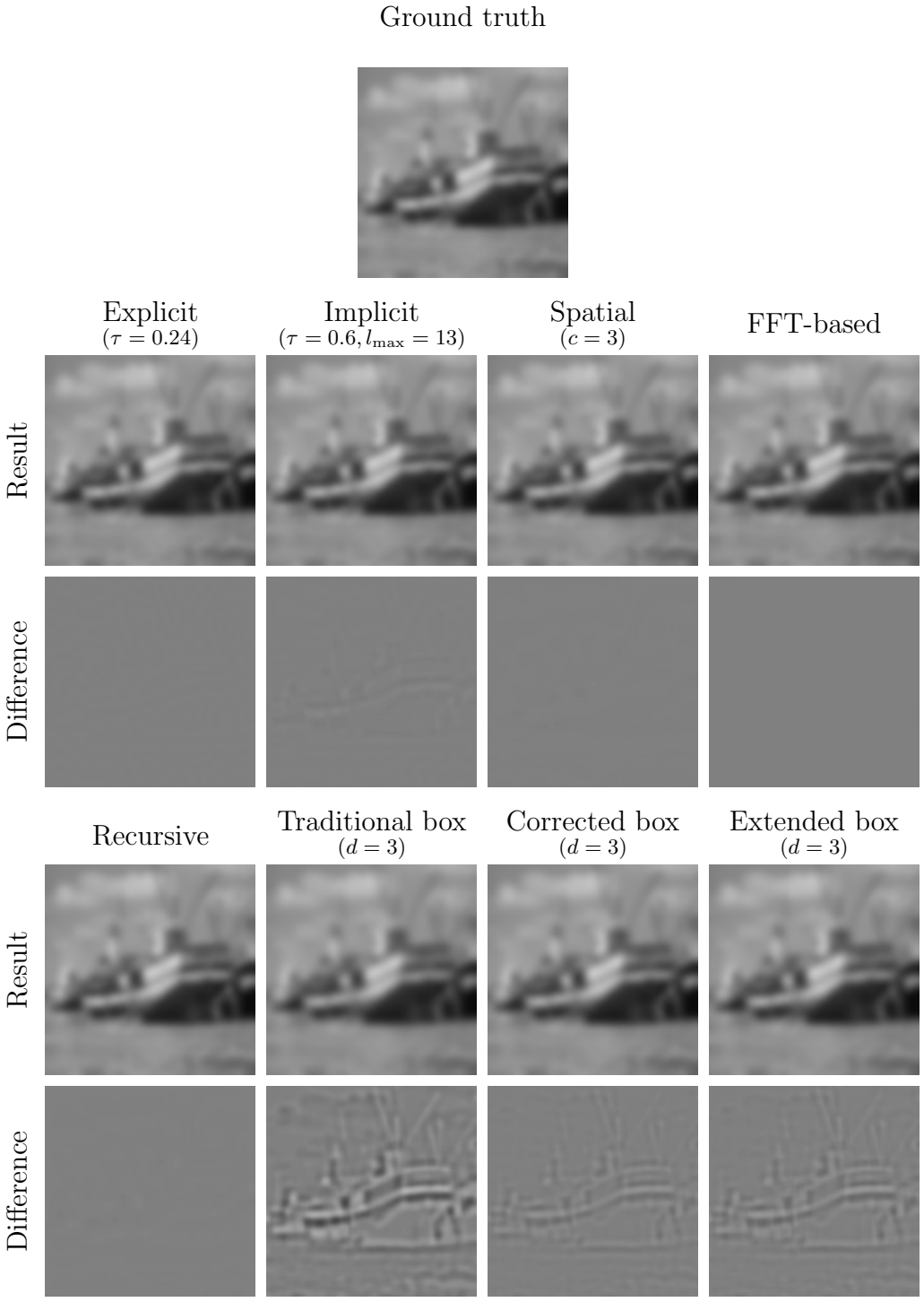



**Figure 3.19**: Visual comparison of linear diffusion algorithms with 'fast' parameters on *Boat*, $512 \times 512$ pixels, for a stopping time $T = 50$. Difference images to the ground truth are scaled by a factor 10 for a better visibility. The zero-error level is depicted in 50% grey.

### 3.6.3 Quality Comparison

#### Visual Similarity

Our aim in this chapter is to find a fast, but accurate GPU-based implementation for linear diffusion. As the first experiment to evaluate which method fulfils this goal best, we visually compare exemplary results of all methods. Regarding the approximation quality, this is the simplest comparison that we can perform, as it involves no further error measure than our intuitive understanding of a similarity. Nevertheless, it is also a very efficient method to find 'obvious' problems in visual computing algorithms.

Figure 3.19 depicts filtering results of the *Boat* image from Figure 3.9 using a linear diffusion process with stopping time $T = 50$. Since we are particularly interested in the points where these methods fail, we steer them with their runtime-optimised parameter set (where applicable). All methods except for the FFT-based approach and recursive filters can be improved in their quality if more time-consuming parameter settings are used.

Visually, all results look similar. Neither do we note striking deviations from the ground truth, nor do the results contain artefacts. This indicates that all presented algorithms in their 'fast' parametrisation are well suited for applications in which only a visual similarity is required.

However, there are indeed differences between the results obtained by the different methods and the ground truth. In order to visualise them, we subtract both solutions from each other, scale the result by a factor 10, and shift it to a zero-error level of 50% grey. This is shown in the second rows of Figure 3.19. The largest differences among these error plots are observed for traditional box filtering. We see most of the structure of the original image which indicates that image edges were not blurred accurately. This is primarily a problem of the rounding from (3.89), as can be seen for the two other box filtering variants. Both corrected and extended box filtering show a much lower error level. Nevertheless, they still reveal a significant amount of structure, such that we can additionally state that $d = 3$ iterations are still too few to smooth all errors sufficiently out.

Among all other tested methods, only the error plot for implicit diffusion hints at the structure of the original image. This is clearly a sign of suboptimal parameter choices, since we already convinced ourselves about the high quality of discrete diffusion algorithms in Section 3.6.1. However, we see later in this chapter that a high-quality parametrisation of implicit diffusion is infeasible in a reasonable runtime.

Explicit diffusion, spatial convolution with a Gaussian, a multiplication

in the frequency domain, and recursive filtering all provide about the same solution, with no visible errors in the difference images. We detail more on the quality of these methods in the next section. As a final remark, note that none of the depicted error images shows artefacts along the image boundaries. This important property is for some classes of algorithms not self-evident. Recall that several instances of recursive filtering have significant problems with image boundaries as a matter of principle (see Section 3.3.5 for details).

### Quantitative Evaluation

Let us now substantiate the visual impressions obtained in the previous section by a measurement of the mean square error to the ground truth. In order to provide a good comparability of all methods we do not fix the stopping time to one value as done before, but observe the behaviour over a range of admissible values. Without loss of generality, we use the standard deviation $\sigma$ of the corresponding Gaussian as a free parameter for this purpose. This accounts for several aspects:

- Most of the presented algorithms approximate Gaussian convolution and depend canonically on the standard deviation.
- The standard deviation is an intuitive measure for the average distance over which grey values are transported. As such, it corresponds to our intuitive understanding of 'blurriness'. In context of arts and image manipulation software, this intuitive measure is often called a 'radius of effect' [dJ07].
- If we convolve an image with a kernel of standard deviation $\sigma$ and resize the solution by halving its edge length, we obtain a similar solution as convolving the resized original by a kernel with standard deviation $\sigma/2$. In contrast, the stopping time scales with the number of pixels in the image.

By (3.19), this notion can easily be translated to the stopping time $T$. As a ground truth, we use again a spatial convolution with truncation $c = 10.0$, and normalise our measurements with respect to a unit grid spacing $h = 1$. Again, we perform our experiments on the boat image, since it represents a 'typical' example for a real-world image.

Figure 3.20 shows the results of this experiment. Note that this graph contains the results for box filtering with $d = 3$ from Figure 3.17. The quality obtained in this experiment can be improved further if we increase

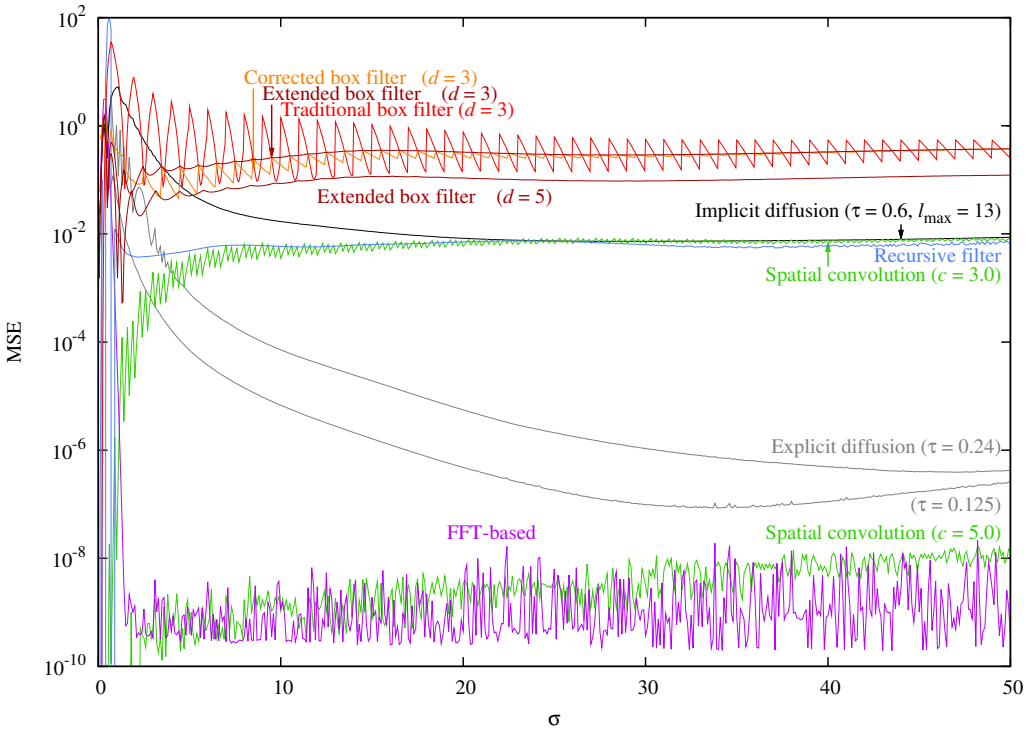



**Figure 3.20**: Mean square errors of different linear diffusion algorithms under varying standard deviations $\sigma$, on *Boat*, 512×512 pixels.

$d$ to 5 iterations. For visualisation purposes, Figure 3.20 depicts only the graph for extended box filtering. However, the missing graphs for traditional and corrected box filtering show a similar behaviour to the case $d = 3$.

Note that many methods evaluated in this experiment yield high or heavily oscillating errors for about $\sigma < 2$. Depending on the method, this is either related to the order of consistency of the technique, to numerical approximation errors for small standard deviations, or due to the fact that for small standard deviations, the discrete solution of the heat equation differs significantly from the ground truth of Gaussian filtering (see Section 3.6.1). However, as we are going to see in more detail later in this chapter, solutions for small standard deviations (or stopping times) can be efficiently computed. Depending on the application, this works either by using explicit diffusion schemes with small stopping times, or by using Gaussian convolution with small kernels. To this end, we can safely ignore that our current parametrisation leads to the observed effects for extremely small standard deviations, and concentrate on the more challenging and computationally intense configurations instead.

For $\sigma > 20$, implicit diffusion with time step $\tau = 0.6$ and $l_{\max} = 13$ inner iterations, spatial convolution with truncation $c = 3$, and recursive filtering yield almost equivalent results. These are roughly one order of magnitude better than those for box filtering, which comes down to MSEs of about $10^{-2}$. However, if we choose $\sigma$ from the interval $[2, 20]$, we notice that implicit diffusion looses accuracy. This is a consequence of the unfavourable ratio of small stopping times and large time steps, which leads to higher approximation errors [Saa03]. By spending more time on outer iterations, i.e. decreasing $\tau$, this problem can be remedied. In contrast, the approximation quality of spatial convolution with a truncated Gaussian becomes better the smaller the standard deviation is. This is because for a constant truncation $c$, the normalisation from (3.23) always redistributes the same amount of grey values onto the remaining samples of the truncated kernel. However, since the kernel is narrower for small $\sigma$, the relative error induced to each sample is also lower than for large $\sigma$.

Explicit diffusion surpasses the quality of the aforementioned methods by up to 2 orders of magnitude. Motivated by the considerations from Section 3.3.1, we compare a quality-optimised parametrisation of $\tau = 0.125$, as well as a runtime-optimised setup $\tau = 0.24$. While even for small stopping times, both measurements indicate a higher quality than for the aforementioned methods, the error drops even further if we consider larger standard deviations. For $\sigma = 50$, explicit diffusion yields an MSE which is four orders of magnitude below those of recursive filtering. The impact of the variation-diminishing setup with $\tau = 0.125$ is clearly visible in the plot. In general, it yields results which are one order of magnitude below the runtime-optimised setting. However, for large standard deviations, the error for this method rises again. This is a result of accumulating approximation errors over the much higher number of iterations $k_{\max}$ [GO96]. Note that by (3.12) and (3.19), $k_{\max} \sim \sigma^2$.

The best approximation quality is obtained by a multiplication in the frequency domain, denoted by 'FFT-based', and by spatial convolution with a truncation $c = 5$. Both create an MSE below $10^{-8}$ which is about the lowest error level that can still be distinguished in single precision floating point arithmetics, given that the image to be analysed has a value range $[0, 255]$. As a consequence, the graphs for both techniques are severely distorted by noise. Once more, our experiment indicates that a truncation of the Gaussian at $c > 5$ does indeed not make sense on single precision floating point arithmetics, because under these circumstances, the error can not be reduced significantly further by increasing $c$.

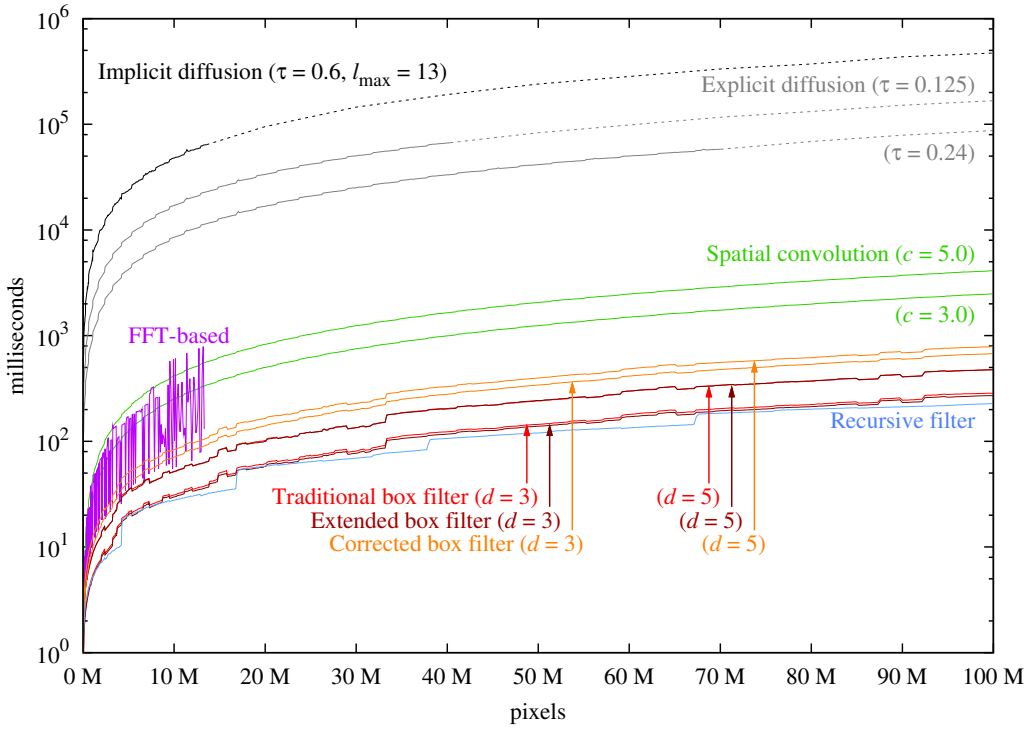



**Figure 3.21**: Runtimes of different linear diffusion algorithms with a standard deviation $\sigma = 50$ over square images containing a varying number of pixels. Benchmarks are executed on a GTX 480.

## 3.6.4 Runtime

### Scaling Over Pixels

Let us now benchmark the runtime requirements of the presented algorithms. As a platform for our benchmarks, we use a modern NVidia GeForce GTX 480 graphics card with 1.5GB RAM on an Intel Core2 Duo E8200 with 2.66 GHz. In the first experiment of this kind, we are interested in the runtime of the algorithms over a varying size of images. For all experiments in this chapter, we assume data to be already residing in GPU memory, such that the data flow over the PCI bus is not accounted for. This assumption is in particular valid if we recall that in real-world applications, linear diffusion is often only one small step of a more complex algorithm. In such cases, we can assume that the entire framework sets up on GPU-based algorithms.

Figure 3.21 shows the measured runtimes obtained in this experiment. In accordance with our considerations from Section 2.3, these benchmarks are performed on squarish images of increasing size and filtered by a

trimmed mean over 3 out of 5 independent measurements.

Note that parts of the graphs are visualised with dotted lines. At these locations, the timing graphs possess a lower resolution than in the remaining parts. Due to the high absolute runtimes of the algorithms, an equally dense test series as for the solid graphs cannot be acquired in a feasible amount of time. As a consequence, these graphs still correctly visualise the general runtime behaviour of the analysed method, but does no longer reveal fine details such as the characteristic 'staircasing' in GPU-based programs.

Such discontinuities in the runtime occur whenever the CPU dispatches a new bunch of blocks to the GPU. Before it can do so, it must wait for the kernel call to return, compute the dimensioning of the next bunch, and re-call the kernel. This costs a constant amount of time in which no pixel can be processed. Hence, whenever the image size grows over a boundary of $n$ bunches, the runtime increases by this constant. In Figure 3.21, this behaviour is particularly well visible for the recursive filter and for the discrete diffusion processes. In latter case, continuous segments are very small, because the number of bunches is frequently increased whenever the number of pixels $n^2$ exceeds the size of one bunch. The recursive filter, in contrast, is separated into two 1-D processes, such that the discontinuities occur for the side length $n$ exceeding the size of a bunch. This explains the much longer continuous segments in this case.

Due to the high number of iterations required to solve the discrete diffusion equation, the implicit diffusion scheme with Jacobi relaxation has the slowest runtime among all evaluated methods. It requires more than 8 minutes to filter an image of size $10\,000 \times 10\,000$ pixels with a stopping time $T = 1\,250$. Of course, the Jacobi method applied as a relaxation of the linear systems of equations is not the optimal solution, even on a GPU. With modern numerics, such as with multigrid methods, the time required per time step can be further reduced [Hac85, GT08]. Unfortunately, this effort hardly pays off for linear diffusion. Although this approach would allow for larger time step sizes, the solution for each step is still expensive, in particular on parallel hardware. Assume a time step size $\tau = 10$ which leads to $k_{\max} = 125$ steps. Even if each of these steps required only the time of one explicit time step, the process would take about 1.25 seconds for an image with 100 megapixels. We see on the next pages that this performance is easily surpassed by other, less challenging algorithms.

Depending on the selected time step width, the explicit diffusion scheme performs a factor 4 to 8 better. With the high-quality setting $\tau = 0.125$, the algorithm processes the same 100 megapixels image in less than 3 minutes. Using the runtime-optimised setting $\tau = 0.24$, this time is reduced to about 1.5 minutes.

All remaining algorithms are significantly faster than these discrete diffusion processes. For this particular standard deviation $\sigma = 50$, spatial convolution with a Gaussian outperforms these algorithms by more than one order of magnitude, depending on the setting of the cut-off parameter. With $c = 5$, our 100 megapixels image requires about 4 seconds. If we use $c = 3$, already 2.5 seconds suffice. Note that in Section 3.5.3, two different algorithms for spatial convolution were presented. The results presented above are exclusively obtained by the algorithm for large standard deviations. The second algorithm cannot be benchmarked in this experiment, because the kernel footprints occurring for $\sigma = 50$ and $c \in \{3, 5\}$ exceed the shared memory limits. We see the relation between these two algorithms in the next experiment.

The application of box filters reduces the runtime even further. In general, we do not notice a significant difference in the runtimes for traditional and extended box filtering. Both the absolute values of the measurements, as well as the scaling behaviour is almost identical. This shows that the performance is not bounded by computational limits, but only by the low memory bandwidth. Since the additional weights read and evaluated by our extended box filter reside in registers, they do not affect the overall runtime. To this end, both the extended and the traditional box filter process the 100 megapixels image with 5 iterations in 476 milliseconds. In case of 3 iterations, the extended box filter requires 269 milliseconds, which is even slightly faster than the traditional box filter that needs 284 milliseconds. This anomaly can either be related to inaccuracies in the time measurement, or to a slightly different timing of memory requests.

The performance of box filters is only surpassed by the recursive filter, which represents the fastest algorithm in our experiment. Using this algorithm, the 100 megapixels image can be filtered in less than 230 ms, which corresponds to a frame rate of about 4.4 FPS. This corresponds to a filter throughput of 440 millions of pixels per second, and allows to compute images up to a size of $4096 \times 4096$ with more than 25 FPS. To this end, the GPU-based recursive filter can process images as they are acquired from commodity DSLR cameras in real-time, and yields more than 140 FPS on Full HD video frames.

It remains to discuss the runtime performance of the FFT-based convolution approach. In Figure 3.21, the graph lives in the interval between the spatial convolution algorithm and the box filters, but shows severe oscillations. This is a consequence of the FFT itself, which performs best on images with power-of-2 side lengths. Other factorisations are possible, but those transformations are much more expensive. Note that for many applications, it is acceptable to pad the mirrored image sufficiently to obtain an

image plane with power-of-2 edges. This strategy offends the periodisation of the image, but is often not severely reflected in the approximation error. To this end, we can achieve the lower runtime bound if we accept a slightly worse result. However, another problem remains. The graph ends already at about 13 megapixels. The reason for this is that the algorithm exceeds the memory limitations already at this size. Beyond this mark, only 6 image sizes whose edge lengths can be factored to small primes can be processed. The largest image computable on a GTX 480 is $4096 \times 4096$ pixels large. This size does often not suffice for practical applications, considering that many modern digital cameras are already equipped with larger sensors. Hence, this limitation is another argument to use other implementations such as the recursive filter.

With results shown in Figure 3.21, our experiment also indicates that our recursive filter on the GPU outperforms other GPU-based algorithms from the literature. The approach of Riley, which sets up on Gabor filters, yields about 100 FPS for images of size $640 \times 480$ pixels [Ril09]. The recursive filter implementation presented in this chapter processes images of the same size with about 320 FPS. Fialka and Čadík proposed an FFT-based convolution which is very similar in structure to the one presented in this chapter [Fv06]. However, in their experiment they only obtain 9 frames per second on an image of size $1024 \times 1024$ pixels. This is surpassed by our FFT-based approach which yields 103 FPS on the same image size, and finally outperformed by the recursive filter which yields 191 FPS. However, due to the rapid progress in the development of graphics cards, it is hard to tell which portion of this runtime difference is related to algorithmic issues, and which is only caused by the speed of the graphics card. Regarding the latter case, it is valid to assume that the two FFT-based algorithms nowadays require about the same time, such that our recursive filter implementation should again be about a factor 2 faster.

### Scaling Over the Stopping Time

Now that we obtained a feeling for the absolute runtimes of the algorithms and their scaling behaviour over different image sizes, we analyse the scaling behaviour over different stopping times. We know that grey values are transported further in the image domain the larger the stopping time $T$ of the diffusion process is. Hence, it is not surprising if some algorithms reflect this different workload in their runtime. Similar to the quality benchmarks, we normalise this benchmark to standard deviations of the kernel rather than to the stopping time of the corresponding diffusion process.

Figure 3.22 shows runtime benchmarks on a blank image of size $4096 \times$

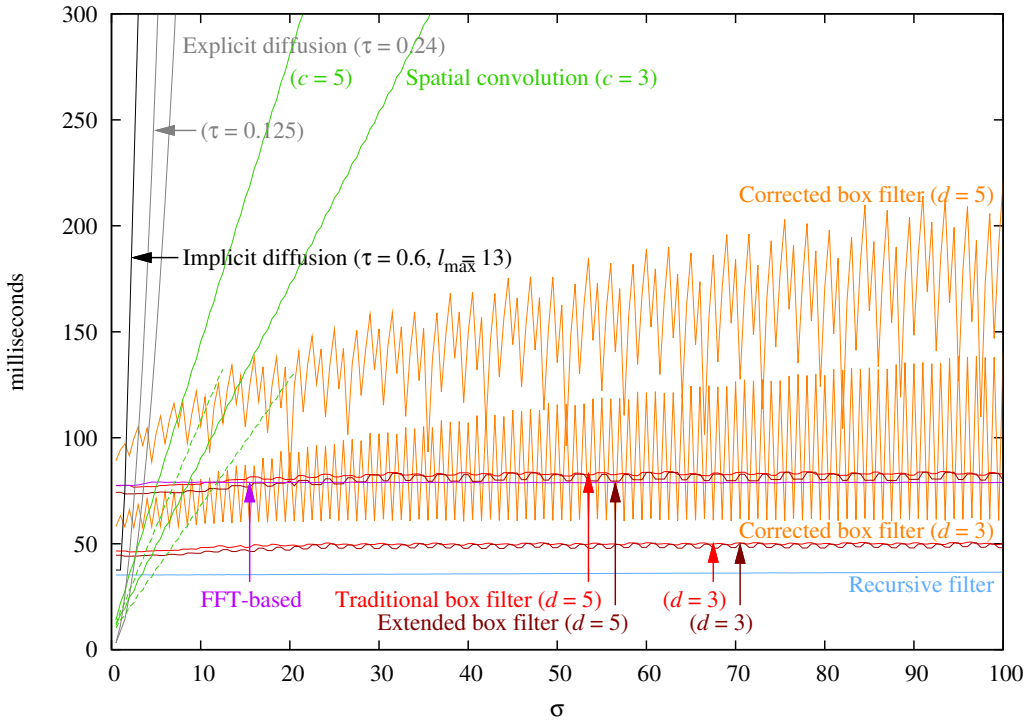



**Figure 3.22**: Runtimes of different linear diffusion algorithms over varying standard deviations $\sigma$, on an image of size $4\,096 \times 4\,096$ pixels. Dashed lines correspond to the cache-based spatial convolution algorithm. All benchmarks are executed on a GTX 480.

4096 pixels. In general, we observe four different behaviours among the tested methods:

1. Our recursive filter, the extended and traditional box filters, as well as the frequency-based convolution algorithm are almost constant in the standard deviation $\sigma$. The overhead for mirroring in case of the box filters and the recursive filter largely vanish in the total runtime. Note that since the side length of the image is a power of two, the FFT-based method performs better than average, and even requires less time than the box filters with $d = 5$.

2. Spatial convolution algorithms scale linear in $\sigma$. The results performed with the cache-based convolution algorithm for small standard deviations are denoted with dashed lines. As expected, this variant is faster than the more general implementation, but the neighbourhood footprint $c\sigma$ is limited to the size of three CUDA blocks.

3. Box filters with correction represent a tradeoff between the aforementioned runtime classes. Since the amount of post-processing changes with increasing $\sigma$, the graphs are noisy.

4. Explicit and implicit diffusion algorithms scale quadratic in the standard deviation, which corresponds to a linear dependency in the stopping time $T$. For large standard deviations, this property makes them very slow in practice.

As a main conclusion, we learn that apart from small standard deviations, a recursive filter on the GPU seems to be the most efficient algorithm. Its break-even points with spatial convolution are at $\sigma = 4.62$ and $\sigma = 2.75$ for truncations $c = 3$ and $c = 5$, respectively. In both cases, we take the faster cache-based algorithm as a basis. Note that we should ignore the fact that explicit diffusion seems to outperform the runtime of spatial convolution for $\sigma < 1$. Our parametrisation with a step size of $\tau = 0.24$ or $\tau = 0.125$ is usually not well suited for such small stopping times (see Section 3.6.3 for details).

For regularisation purposes in context of optic flow computations in chapter 6, we come back to the results of this experiment. There, we use a fast cache-based spatial convolution with $c = 3$ in the common case that $\sigma < 4.62$, and fall back to a recursive filter above this threshold.

### Comparison to CPU

Let us now compare the runtime of our GPU-based algorithms to their corresponding CPU-based variants. In all cases, we use the same type of numerics on the CPU and the GPU, and hand-optimise the algorithms with respect to their required runtime. For example, the role of the `cuFFT` library on CUDA is played by the `FFTW3` library on the CPU, which provides a similar degree of functionality and optimisation to compute fast Fourier transforms of arbitrary size.

Note that in some cases, such as for implicit diffusion, more efficient numerics on the CPU exist. Here, it might help to choose larger time steps $\tau$, and solve the arising system of equations by means of bidirectional multigrid methods [Hac85]. However, benchmarking against such advanced strategies on the CPU bears the risk of a lacking comparability. A detailed analysis of the advantages and drawbacks of these techniques and potential consequences to more advanced implicit diffusion algorithms on the GPU go far beyond the scope of this thesis.

Our experiments are performed four different image sizes. These input images are all square-shaped, and have edge lengths of 256, 1024, 4096,

and 8192 pixels. Due to a better occupancy, we expect the speedup of the GPU-based algorithm to be higher the larger the image.

Table 3.1 shows the comparison of runtimes for the presented algorithms, as well as the speedups between both architectures. Depending on the complexity of the algorithm, we obtain speedups up to a factor 138 which is a lot even for graphics processors. However, we also notice that the analysed algorithms behave very differently.

In total, the lowest speedup is obtained for traditional box filtering. Even for very large images, it does not grow beyond a factor 14. In this case, the reason is obvious. Traditional box filtering is a strictly memory-bound algorithm with almost no arithmetic computations, such that the runtime is only limited by the memory throughput. This is also reflected in the extended box filter which requires significantly more time than the traditional variant on the CPU, but about the same runtime on the GPU. As a result, we obtain speedups up to about 35.

By similar arguments, we measure the largest speedups for recursive filtering. While on the CPU, reads to the stencil require a significant amount of time, the GPU holds all relevant data in the fast on-chip memory. This allows a high data throughput and might be one reason for the high performance of recursive filters on the GPU.

A similarly high speedup can be observed for the fast Fourier transform which dominates the frequency-based convolution algorithm. Since both the CPU and the GPU variant set up on complex third-party libraries, it is hard to tell which optimisations are in the end responsible for this efficiency. However, it is likely that the closed-source `cuFFT` library is closely tailored to the specific memory structure and scaling behaviour of GPUs, and that is computes large portions of the algorithm on the fast shared memory of the device. However, note that neither the CPU-based variant nor our GPU-based algorithm can compute the 67 megapixels image, such that this row is omitted in Table 3.1.

Although they are also memory-bound, the two iterative discrete diffusion algorithms still yield a 60 times lower runtime on the GPU than on the CPU. This is because of the efficient texturing units on graphics processors which supply data in 2-D aware caches. This is fundamentally different on CPUs, where applications of the outer off-diagonals of the penta-diagonal system operator usually require additional reads to the RAM. In the next chapters, we come back to this observation and exploit the efficient scaling of those stencil-based operations for more advanced numerics.

Finally, let us have a look on the scaling behaviour of the spatial convolution algorithm. Since the chosen standard deviation is too large to use the faster cache-based algorithm, we read all inputs from textures. Never-

**Table 3.1**: Runtime comparison of filters with standard deviation $\sigma = 50$ on CPU and GPU.

| | Size | CPU (ms) | GPU (ms) | Speedup |
|---|---|---|---|---|
| **Explicit** ($\tau = 0.24$) | 65 536 | 3 481.94 | 127.69 | 27.27 |
| | 1 048 576 | 55 806.57 | 935.54 | 59.65 |
| | 16 777 216 | 893 450.98 | 13 843.68 | 64.54 |
| | 67 108 864 | 3 572 479.98 | 55 217.33 | 64.70 |
| **Implicit** ($\tau = 0.6$, $l_{\max} = 13$) | 65 536 | 18 113.45 | 703.27 | 25.76 |
| | 1 048 576 | 291 700.40 | 5 247.96 | 55.58 |
| | 16 777 216 | 4 659 460.45 | 77 970.11 | 59.76 |
| | 67 108 864 | 18 627 402.34 | 311 709.51 | 59.76 |
| **Spatial conv.** ($c = 3$) | 65 536 | 32.43 | 1.87 | 17.34 |
| | 1 048 576 | 520.60 | 26.40 | 19.72 |
| | 16 777 216 | 8201.34 | 417.19 | 19.66 |
| | 67 108 864 | 32783.46 | 1 668.98 | 19.64 |
| **FFT-based** | 65 536 | 16.98 | 2.12 | 8.01 |
| | 1 048 576 | 388.00 | 9.68 | 40.08 |
| | 16 777 216 | 9 316.78 | 78.86 | 118.14 |
| **Recursive filter** | 65 536 | 4.63 | 1.57 | 2.96 |
| | 1 048 576 | 251.41 | 5.22 | 48.16 |
| | 16 777 216 | 4 963.19 | 35.87 | 138.37 |
| | 67 108 864 | 19 766.75 | 145.06 | 136.27 |
| **Traditional box** ($d = 3$) | 65 536 | 2.54 | 1.20 | 2.11 |
| | 1 048 576 | 40.06 | 5.35 | 7.49 |
| | 16 777 216 | 634.21 | 49.96 | 12.69 |
| | 67 108 864 | 2 583.34 | 190.64 | 13.55 |
| **Corrected box** ($d = 3$) | 65 536 | 17.22 | 1.67 | 10.29 |
| | 1 048 576 | 283.66 | 9.62 | 29.49 |
| | 16 777 216 | 4 611.05 | 115.72 | 39.85 |
| | 67 108 864 | 19 406.41 | 451.24 | 43.01 |
| **Extended box** ($d = 3$) | 65 536 | 3.96 | 1.09 | 3.63 |
| | 1 048 576 | 57.94 | 5.02 | 11.54 |
| | 16 777 216 | 929.76 | 48.12 | 19.32 |
| | 67 108 864 | 3 775.50 | 180.55 | 34.78 |

theless, memory patterns are very irregular in this case and the size of the kernel often causes cache misses. As a consequence, we obtain moderate speedups ranging from about 17 to 20 for all image sizes.

## 3.7 Summary

In this chapter, we have designed and compared many different algorithms for homogeneous diffusion on the GPU. The best performance among these methods is obtained by a novel GPU-based algorithm which is based on the recursive Young-van-Vliet filter with an additive implementation as suggested by Hale [YvV95, Hal06b]. On a modern Geforce GTX 480 graphics card, this algorithm yields a data throughput of 440 millions of pixels per second, regardless of the chosen stopping time of the diffusion process. This allows to compute more than 140 full HD video frames (of size $1920 \times 1080$ pixels) per second, and the algorithm still achieves real-time performance on images in the size of current DSLR sensors. Moreover, the proposed method only requires two buffers in the size of the input image in graphics RAM. This guarantees that even very large images containing more than 100 millions of pixels can be processed on-the-fly without the necessity to swap out data into CPU RAM, or even to the hard disk.

Compared to a similar recursive filter on the CPU, the GPU-based algorithm obtains a speedup of more than 130. Even if we compare against a less accurate, 3 times iterated box filter on the CPU, the proposed method still runs 18 times faster. For any application with a tight time constraint, this speedup allows to process images with more than a quadruplicated edge length in the same time as before.

Besides this evaluation, we have also discussed extended box filtering as an inexpensive novel approach to improve the quality of traditional iterated box filtering. Although this method is slightly less accurate than recursive filtering, it enjoys a similar runtime. Compared to recursive filtering, however, it possesses an intuitive, mathematically well-founded structure that makes it easy to implement even on massively parallel hardware. There is no need to evaluate complex-valued coefficients, or to optimise parameters numerically before filtering, such as it is for the scaling variable $q$ in recursive filtering. Because extended box filtering also works with only two buffers of input image size, this new method is also very well suited for the processing of very large images.

As a general conclusion from our experiments, we can state that one of the main challenges for homogeneous diffusion on massively parallel hardware is given by the dominant memory-boundedness of the operation.

Those algorithms which process the image with the least global memory accesses achieve the largest speedups – almost regardless of the number of performed operations. This shows that if the current development of massively parallel hardware continues as before, there might soon be a time when simple diffusion algorithms no longer scale with increasing computational power. Instead, it becomes more and more important to increase the memory bandwidth from the processor cores to the device RAM, or to enlarge the fast on-chip caches.

However, there is also potential room for improvement for some of the presented algorithms. Let us focus on the two most promising approaches: Extended box filtering and recursive filters. As we have seen in Section 3.6.2, the cases in which the standard deviation of a traditional box filter equals the standard deviation of a Gaussian are not necessarily the cases in which this filter provides the best approximation. If instead, we succeed to find a relation between the minima of the graph from Figure 3.17, we can select the traditional box filter based on this new measure. Then, there is also no objection against defining the extended box filter as an interpolation of these new instances of the traditional filter – and benefit from a much lower approximation error.

A potential improvement for recursive filters follows from observations in our runtime experiments on the GPU. For box filters, as well as for the recursive filter, we found that the runtime depends largely on the number of memory loads and stores, but only marginally on the number of arithmetic operations. For certain applications, it might thus be interesting to increase the support of the recursive update to allow higher-order filters. Since these operations are likely to be performed in fast on-chip memory, this measure should yield a marginal runtime overhead, but a significant increase in quality.

# Chapter 4

# Anisotropic diffusion

*I can't change the direction of the wind,*
*but I can adjust my sails to always reach my destination.*
Jimmy Dean

## 4.1 Motivation

In the last decade, many PDE-based algorithms in visual computing enjoyed a significant boost in quality. One main ingredient responsible for this success are *nonlinear diffusion schemes*. Such techniques act smoothing within relatively flat image regions, but preserve dominant image edges. Hence, they are very useful to smooth out high-frequent noise, while image structures are mostly untouched.

We distinguish two classes of nonlinear diffusion schemes. *Isotropic* techniques such as the approach of Perona and Malik weight the flux along the image gradient by the magnitude of this gradient [PM87]. As it turns out, such strategy yields the desired behaviour in flat regions, but often handles edges overcautiously: Since the smoothing behaviour of the method approaches zero in regions with edges, noise along the edge cannot be removed sufficiently well (see [Wei98, Figure 5.2]). This is fundamentally different for *anisotropic* diffusion schemes. In addition to the magnitude of the image gradient, such approaches also consider its direction [Wei98]. This allows to smooth along image edges, while diffusion across such edges is inhibited.

While for many applications, nonlinear diffusion yields more favourable results than homogeneous diffusion, it is also computationally more chal-

lenging. Fast convolution-based algorithms as presented in Chapter 3 are usually not applicable, such that there is often no way other than solving the discretised diffusion equation directly. Since the system operator is now nonlinear, it must be repeatedly evaluated over the course of the evolution. This requires image gradient information, and causes another runtime overhead.

In view of this situation, it seems reasonable to design a GPU-accelerated algorithm for nonlinear diffusion. With similar arguments as for the previous chapter, there is a justified hope that such an algorithm possesses a higher bandwidth than its CPU-based variant. This allows to compute much larger images in the same runtime, and helps to lift the bound for the maximal image size to be computed in real-time.

Despite this wide applicability and promising speedup involved in such a GPU-based approach, there are few algorithms proposed in the literature. Most techniques that are efficiently parallelised and evaluated on GPUs target at nonlinear isotropic diffusion models such as the Perona-Malik approach [PM87] (see e.g. [RS01, SWD05, Pat07, How10, Pus11]). Such models are computationally less expensive than anisotropic approaches, but as it is described above, also less suited for a wide range of applications. Compared to anisotropic schemes, they are fairly easy to implement and parallelise.

Other works from the literature aim at the solution of an anisotropic diffusion model, but circumvent the computationally expensive solution process. By using fast alternatives, these methods yield results which only look alike, but do not resemble, anisotropic diffusion. While the method from [Fel08] sets up on iterated adaptive filtering, the technique from [SKB+11] uses a bilateral filter. Such approximations are very fast: On modern hardware, both approaches yield realtime performance on images of size $1\,024 \times 1\,024$ pixels.

Real anisotropic diffusion processes for GPUs are rare. An example for such algorithms is the GPU-based 3-D edge-enhancing diffusion algorithm from [BLF+07]. Although the original paper gives only very little details about the implementation, it seems that the authors used a standard explicit solver to compute the results. This results in a sub-optimal runtime.

It is the goal of this chapter to design a simple but efficient scheme for anisotropic diffusion on GPUs. In this context, we are not interested to search for the *optimal* algorithm under particular conditions, as this goes far beyond the scope of this thesis: Because the model depends on more parameters, it is also likely that the number of fundamentally different approaches to the problem is also significantly higher. Instead, it is our goal to present a very *flexible* and *efficient* approach which is still *simple*

enough to allow an easy adaptation to other PDEs. This philosophy is carried over to the following two chapters, in which we discuss an application of the algorithm derived here to much more complex PDE-based methods in visual computing.

## 4.2 Edge and Coherence Enhancing Diffusion

Let us begin with a review of the *edge and coherence enhancing diffusion (ECED)* model by Weickert [Wei98, Wei11a]. Given an image $u \in \Omega \to \mathbb{R}$, it is modelled by the diffusion equation

$$\partial_t u = \operatorname{div}\big((\boldsymbol{D}(J_\rho(\boldsymbol{\nabla} u_\sigma)))\ \boldsymbol{\nabla} u\big)\,, \tag{4.1}$$

where

$$u_\sigma = K_\sigma *\ u\,, \tag{4.2}$$

$$J_\rho(\boldsymbol{\nabla} u_\sigma) = K_\rho *_c (\boldsymbol{\nabla} u_\sigma \boldsymbol{\nabla} u_\sigma^\top)\,, \tag{4.3}$$

and where $K_\sigma$, $K_\rho$ denote Gaussians with standard deviations $\sigma$ and $\rho$, respectively. In this context, we use $*_c$ to denote the component-wise convolution. The $2 \times 2$ diffusion tensor $\boldsymbol{D}$ steers the anisotropic behaviour of the process. We want to express it in an orthonormal base that aligns to the edges in the image, such that we can control the diffusivities along and across the edges separately [Wei98]. Hence, we choose its eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ as the eigenvectors $\boldsymbol{j}_1, \boldsymbol{j}_2$ of the smoothed structure tensor $J_\rho(\boldsymbol{\nabla} u_\sigma)$ [FG87]. The idea behind this selection becomes obvious if we let $\rho = 0$. Then, we have

$$\boldsymbol{v}_1 \,||\, \boldsymbol{\nabla} u_\sigma, \qquad \boldsymbol{v}_2 \perp \boldsymbol{\nabla} u_\sigma\,. \tag{4.4}$$

By a smart choice of the eigenvalues $\lambda_1, \lambda_2$ of this system, we can steer the diffusion across and along the edge separately. The additional smoothing of the structure tensor by $K_\rho$ allows to adapt to coherent structures rather than directly to edges [Wei98]. Note that the variance $\rho$ of the Gaussian kernel determines the scale on which these edges are sought.

We want a strong smoothing along dominant edges, but a vanishing smoothing across them. Assuming $\mu_1$ and $\mu_2$ to be the eigenvalues of the smoothed structure tensor $J_\rho(\boldsymbol{\nabla} u_\sigma)$, this leads to the configuration

$$\lambda_1(\boldsymbol{\nabla} u_\sigma) \ :=\ g\left((\mu_1 - \mu_2)^2\right), \tag{4.5}$$

$$\lambda_2(\boldsymbol{\nabla} u_\sigma) \ :=\ 1\,. \tag{4.6}$$

The diffusivity $g$ is a decreasing function which takes the value 1 for the argument 0. Hence, it reduces diffusion across large image gradients, but makes the process act similar to homogeneous diffusion in flat regions. In this chapter, we use a diffusivity as proposed by Weickert [Wei98]:

$$g(s^2) = \begin{cases} 1, & s^2 = 0 \\ 1 - \exp\left(\frac{-C_m}{\left(s^2/\lambda^2\right)^m}\right), & s^2 > 0 \end{cases}, \tag{4.7}$$

with $m = 4$ and $C_m = 3.31488$. The contrast parameter $\lambda$ steers which slopes are considered an edge, and which are considered to be a flat image region. To this end, it defines a threshold above which gradients are sharpened, while gradients with a lower magnitude are blurred.

Given the eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ and their eigenvalues $\lambda_1, \lambda_2$ as above, the diffusion tensor $\boldsymbol{D}$ is given by

$$\boldsymbol{D} = (\boldsymbol{v}_1, \boldsymbol{v}_2) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \end{pmatrix}. \tag{4.8}$$

Note that this model reduces to the famous *edge-enhancing diffusion (EED)* for $\rho = 0$ [Wei98]. If we choose the eigenvalues of $\boldsymbol{D}$ according to [Wei98, Section 5.2], we obtain the *coherence enhancing diffusion (CED)*.

Let us now briefly address the discretisation of this process. In order to obtain the structure tensor, we use a discretisation as in [WSW08, Equation (29)]. This corresponds to a linear combination between central and one-sided differences with a weight $\alpha = 1/2$, evaluated at positions $\pm 1/2$.

After an adjustment of the Eigenvalues according to (4.5) and a component-wise tensor smoothing as indicated in (4.1), it remains to discretise the actual diffusion process. Here, we use an explicit discretisation in time, such that we obtain

$$\frac{\boldsymbol{u}^{k+1} - \boldsymbol{u}^k}{\tau} = \boldsymbol{A}(\boldsymbol{u}_\sigma^k)\, \boldsymbol{u}^k\,, \tag{4.9}$$

where $\boldsymbol{u}^k$ describes the discrete image at time step $k$, and $\tau$ is an artificial time step size. The nonlinear nona-diagonal operator $\boldsymbol{A}(\boldsymbol{u}_\sigma^k)$ is obtained by a discretisation scheme suggested by Weickert [Wei11c]. It follows from a similar energy minimisation as in [WSW08], but uses a more advanced coupling of central and one-sided differences.

Compared to traditional discretisation approaches, this scheme yields less unintentional blurring across edges, but is also computationally more expensive. The algorithmic consequences from these characteristics are very

different to the homogeneous diffusion algorithms discussed in the previous chapter. Hence, this scheme is an ideal candidate for our parallelisation efforts in this chapter. The insights to be obtained in the experimental evaluation are going to give a good overview about a wide range of diffusion and diffusion-like algorithms in visual computing.

## 4.3 Fast Explicit Diffusion

Let us now focus on the numerical solution of the aforementioned process. Starting with an equation as in (4.9), an standard explicit update represents the simplest approach for this purpose. As we have seen in context of homogeneous diffusion, these schemes also yield reasonable speedups of about 30–60 during parallelisation to the GPU. However, we have also seen that a standard explicit scheme suffers from rigid limitations of the maximal time step size which makes it comparably slow in practice.

A remedy to this problem is given by the *Fast Explicit Diffusion (FED)* scheme which was recently proposed by Grewenig *et al.* [GWB10]. It decomposes the stopping time $t$ for an explicit scheme into a special series of non-uniform time steps $\tau_i$. Although the full FED process is provenly stable, individual steps are not: about 50% of all steps are larger than the maximal stable time step size $\tau_{\max}$. We see later that the stopping time $T$ of a process depends quadratically on the number of steps $n$, such that this technique reaches a given stopping time faster than a standard explicit scheme.

The idea behind FED is related to box filtering (see Section 3.3.6). A 1-D box filter $\boldsymbol{B}_{2n+1}$ can be decomposed into a series of $n$ linear diffusion steps with individual time step sizes $\tau_i$ [GWB10]:

$$(\boldsymbol{B}_{2n+1})_h = \prod_{i=0}^{n-1} (I + \tau_i \boldsymbol{\Delta}_h)\,. \tag{4.10}$$

In this context, $\boldsymbol{\Delta}_h$ denotes the discrete Laplacian operator (using central differences and reflecting boundary conditions):

$$(\boldsymbol{\Delta}_h)_k = \frac{\boldsymbol{u}_{k-1} - 2\boldsymbol{u}_k + \boldsymbol{u}_{k+1}}{h^2}\,, \tag{4.11}$$

The individual time steps $\tau_i$, $i \in \{0, \dots, n-1\}$, are then computed as

$$\tau_i = \frac{h^2}{4\cos^2 \pi \frac{2i+1}{4n+2}}\,. \tag{4.12}$$

It can be shown that the stopping time $t$ of this FED *cycle* is given by

$$t := \sum_{i=0}^{n-1} \tau_i = \frac{h^2}{3} \binom{n+1}{2}, \tag{4.13}$$

which is in general larger than $n \cdot \tau_{\max}$ (see [GWB10] for details).

In Section 3.3.6, we have seen that an iterated box filter with sufficiently many iterations approaches a Gaussian filter. As a consequence, we can iterate FED cycles to obtain a discretisation of a homogeneous diffusion process. Note that up to this point, we have not gained an algorithmic benefit, since each FED cycle can be computed more efficiently by a sliding window approach as in (3.65).

However, FED becomes algorithmically interesting if the discrete Laplacian $\boldsymbol{\Delta}_h$ is replaced by an *arbitrary* semi-definite matrix [GWB10]. In order to maintain the stability of a cycle, this exchange only requires a rescaling of the time steps $\boldsymbol{\tau}_i$ by the eigenvalue of largest absolute value $\mu_{\max}$ of the new operator $\boldsymbol{A}$. By this, we can rewrite (4.12) as:

$$\boldsymbol{\tau}_i(\mu_{\max}) = \frac{1}{|\mu_{\max}| \cos^2 \pi \frac{2i+1}{4n+2}} \,. \tag{4.14}$$

As it was shown by Grewenig *et al.* [GWB10], this modification allows multi-dimensional, non-linear, and even anisotropic processes to be efficiently computed by means of FED. A non-linear operator such as the one from (4.9) is then approximated by a series of linear operators which are kept stable over the course of an FED cycle. After each cycle, it is possible to use the intermediate solution to update the system matrix. Hence, the iteration over FED cycles both increases the approximation quality and takes the role of a time-lagged diffusivities scheme.

A crucial aspect for implementations of this scheme on real hardware are rounding inaccuracies introduced by the limited machine precision. In practice, this leads to different results of the process depending on the order in which the time steps from (4.11) and (4.12) are applied, although they are supposed to yield equivalent results from a numerical perspective. In many cases, these approximation errors are even significantly amplified and destroy the numerical stability of the process. To circumvent this problem, Grewenig *et al.* suggested to re-order the time steps such that the error is dampened best possible (see [Gen79, GWB10, GZG+10]).

In experiments, it turned out that the optimality of a given permutation of time steps only depends on the number of FED steps $n$ per cycle, but not on the underlying problem. This allows to precompute suitable permutations off-stream based on a test problem, and to look up the corresponding

reordering at runtime. An inexpensive way to do this is to select a suitable positive integer $\kappa$ which is relatively prime to $n$, and to permute the steps $\tilde{\boldsymbol{\tau}}_i$ as

$$\tilde{\boldsymbol{\tau}}_i = \boldsymbol{\tau}_j, \quad \text{with } j = \kappa i \bmod n\,. \tag{4.15}$$

For each $n$, this reordering is uniquely determined by only one value for $\kappa$ that must be retrieved during runtime [GWB10, GZG+10].

Let us conclude this section with a sketch of the complete algorithm to solve (4.1) for a given stopping time $T$. We additionally assume to be given the number $d$ of 'outer' FED cycles, which steer the approximation quality of the scheme. The initial state $\boldsymbol{u}(\boldsymbol{x}, T)$ of this process is the input $\boldsymbol{f}(\boldsymbol{x})$.

1. Given $d$ cycles, use (4.13) to compute a (real-valued) minimal number of steps $\tilde{n}$ as
$$\tilde{n} = \sqrt{\frac{24T}{dh^4|\mu_{\max}|} + \frac{1}{4}} - \frac{1}{2}\,. \tag{4.16}$$
Note that because we assume an arbitrary process, $T$ is scaled according to (4.12) and (4.14).

2. Set $n = \lceil \tilde{n} \rceil$, and use (4.14) to find a series $\boldsymbol{\tau}_i(\mu_{\max})_{i\in\{0,\dots,n-1\}}$ of time steps. Since in general $\tilde{n} \neq n$, rescale each of these steps by $\tilde{n}/n$. This procedure dampens errors, such that its stability is preserved.

3. Using a suitable $\kappa$ that is retrieved from a pre-computed lookup table, use (4.15) to reorder these time steps.

4. For each $c \in \{0, \dots, d-1\}$:

   a. Apply a Gaussian filter to smooth the current solution $\boldsymbol{u}(\boldsymbol{x}, ct)$ and use the discretisation from [WSW08] to set up the structure tensor. Using (4.5)–(4.7), rescale its eigenvalues and apply (4.8) to compute $\boldsymbol{A}(\boldsymbol{u}_\sigma^{cn})$ as in (4.9). Note that this step involves a component-wise Gaussian filtering of the diffusion tensor. From this matrix, it suffices to store 4 off-diagonals. While the remaining off-diagonals follow by symmetry, the main diagonal can be computed by the unit row sum property of the operator.

   b. For each $i \in \{0, \dots, n-1\}$:

      - Update the intermediate solution $\boldsymbol{u}^{cn+i}$ by a time step adaptive explicit scheme as in (4.9):
      $$\boldsymbol{u}^{cn+i+1} = (\boldsymbol{I} + \tilde{\boldsymbol{\tau}}_i \boldsymbol{A}(\boldsymbol{u}_\sigma^{cn}))\boldsymbol{u}^{cn+i}\,. \tag{4.17}$$
      Note that this scheme uses the reordered time steps from 3.

## 4.4 Implementation on a GPU

Let us now discuss the efficient implementation of this ECED scheme on a modern GPU. In the algorithm above, we identify the operations taking place in step 4 as massively parallel operations, while the initialisation of the FED scheme (1–3) is a predominantly sequential process which requires access to the look-up table for $\kappa$. Hence, we prepare the reordered array of time steps on the CPU, upload it to the GPU, and perform the remaining operations on the GPU. Note that the size of this array usually vanishes in the size of the original image, such that this proceeding does not cause a significant overhead.

As it turns out, the GPU kernels required in this context can largely be derived from those which were used for homogeneous diffusion (see Section 3.5 for details).

- Our **explicit scheme** used in step 4b constitutes the central operation with the most calls and the highest runtime requirements. The algorithm used in this case does not differ much from the one for homogeneous diffusion which we have discussed in Section 3.5.1. However, it possesses a much higher memory bandwidth. For each pixel a dense $3 \times 3$ neighbourhood of input values must be read, in addition to eight stencil weights.

  These memory loads are very inexpensive if we read data from CUDA textures. Four textures holding the off-diagonals of the operator $\boldsymbol{A}(\boldsymbol{u}_\sigma^k)$ are queried twice per pixel. Dirichlet boundaries ensure that no information is diffused out of the image domain. Following these reads, the main diagonal of the operator is efficiently computed in the fast on-chip shared memory. The intermediate solution from the previous time step must be retrieved 9 times per pixel and channel. However, these requests are also very efficient because of the 2-D aware caches of graphics cards, such that cache misses are reduced to a minimum.

- The computation of the entries of the **diffusion tensor** $\boldsymbol{D}$ is performed in a two-stage algorithm. In the first step, we compute the structure tensor $(\boldsymbol{\nabla} \boldsymbol{u}_\sigma \boldsymbol{\nabla} \boldsymbol{u}_\sigma^\top)_i$ in each pixel, whose eigenvalues and eigenvectors are used in the second step to obtain the diffusion tensor. Both CUDA kernels process the image in 128 pixels wide stripes (see Figure 4.1). Within these stripes, they maintain a lookahead of 1 pixel in either direction in shared memory. This suffices to resolve the memory dependencies stemming from the discretisation as

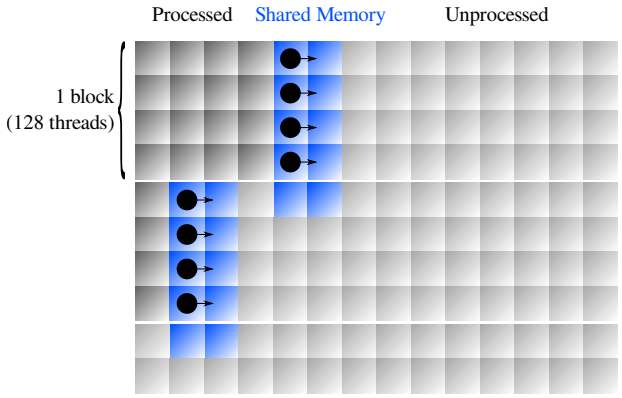



**Figure 4.1**: Processing scheme to set up diffusion tensor entries with the discretisation from [WSW08]. Note the single additional row and column in shared memory.

in [WSW08], and turns out to be slightly more efficient for this application than reading data from textures.

- It remains to find a suitable algorithm for **Gaussian filtering**, such that we can compute $\boldsymbol{u}_\sigma = \boldsymbol{K}_\sigma * \boldsymbol{u}$ and $\boldsymbol{K}_\sigma * (\boldsymbol{\nabla u}_\sigma \boldsymbol{\nabla u}_\sigma^\top)$. We have discussed different algorithms for this purpose in context of homogeneous diffusion (see Chapter 3). Following the experimental results from this chapter, we use the fast cache-based Gaussian convolution from Section 3.5.3 with truncation $c = 3$. If this algorithm fails to provide the required standard deviation because the neighbourhood does not fit into the limited shared memory, we fall back to a GPU-based recursive filter as given in Section 3.5.5.

## 4.5 Experiments

### 4.5.1 Visual Comparison

Let us now evaluate the performance of the presented algorithm. As a first step, we perform an evaluation with respect to the quality of the results. Different to the case of homogeneous diffusion, however, it is much harder to find a suitable ground truth to compare against. The reason lies in the non-linearity of the process. Even small numerical errors in the process can steer the solution to a fundamentally different result.

An example for this challenge is shown in Figure 4.2. Based on the *Boat* image from Figure 3.9, it presents three results of edge and coherence

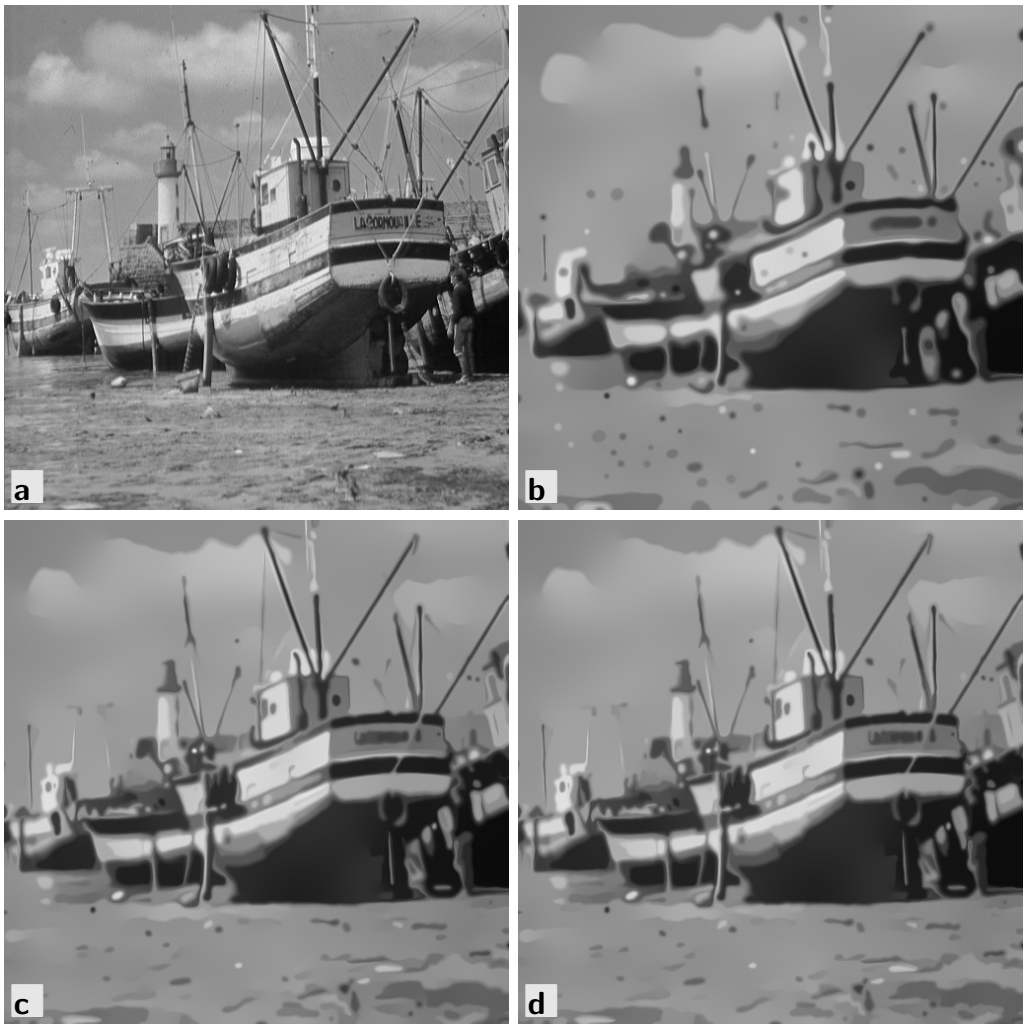



**Figure 4.2**: Edge and Coherence Enhancing Diffusion on **a.** the *Boat* image from Figure 3.9. **b.** Explicit diffusion with $T = 250$, $\tau = 0.125$, $\lambda = 1.0$, $\sigma = 1.0$, $\rho = 1.0$. **c.** Implicit diffusion, same parameters, but $\tau = 50.0$. **d.** FED, same parameters as in b., with $d = 3$ cycles.

enhancing diffusion with a stopping time $T = 250$. Besides CPU-based variants of the classical explicit and implicit schemes, we also apply our new GPU-based FED algorithm.

In this direct comparison, the result of the FED algorithm is almost indistinguishable from the implicit solution, while the explicit result strikingly differs from these outcomes. The reason for this observation are the significantly different numbers of iterations performed with either approach, over which discretisation artefacts and numerical inaccuracies accumulate and steer the process in different directions. In the explicit case, the time step size $\tau$ is limited, such that the process requires 2 000 iterations with $\tau = 0.125$ to yield the desired stopping time. Hence, we use the opportunity to update the non-linear operator after each iteration. This yields a better non-linear behaviour, but is also subject to an increased blurring of edges. In contrast, the implicit approach permits larger time step sizes, which allows us to linearise the present problem in only a few steps. As a consequence, edges are preserved much more prominently. Note that we obtain an equivalent result to Figure 4.2b if we run the implicit scheme with $\tau = 0.125$.

The same arguments carry over to our GPU-based FED algorithm. While it is capable of very large time steps, it depends on the application if the resulting behaviour is intended or not. However, once the time step per FED cycle is reduced, this algorithm gradually fades into a standard explicit solver. This affects both the appearance of the results and the runtime of the algorithm.

As a consequence of these considerations, let us regard FED as an alternative to implicit solvers which can easily be parallelised to massively parallel architectures such as GPUs. In the following, we convince ourselves about the good scaling behaviours of this algorithm. Because the GPU-based version yields the same results as an equivalent CPU-based variant, we refer to [GWB10] for a detailed quality analysis.

### 4.5.2 Runtime Scaling on Image Size

Let us first evaluate the scaling behaviour of our algorithm over images in different sizes. To this end, we create square-shaped images in different sizes, and diffuse them with ECED using the parameters $\lambda = 1.0$, $\sigma = 1.0$, and $\rho = 1.0$, and a stopping time $T = 500$ with $d = 3$ FED cycles. This setting is chosen as a typical configuration for many real-world problems, and gives us a rough idea of the scaling behaviour of our algorithm over different image sizes. As a benchmarking platform, we use again the NVidia GeForce GTX 480 on a Intel Core2 Duo E8200 platform.

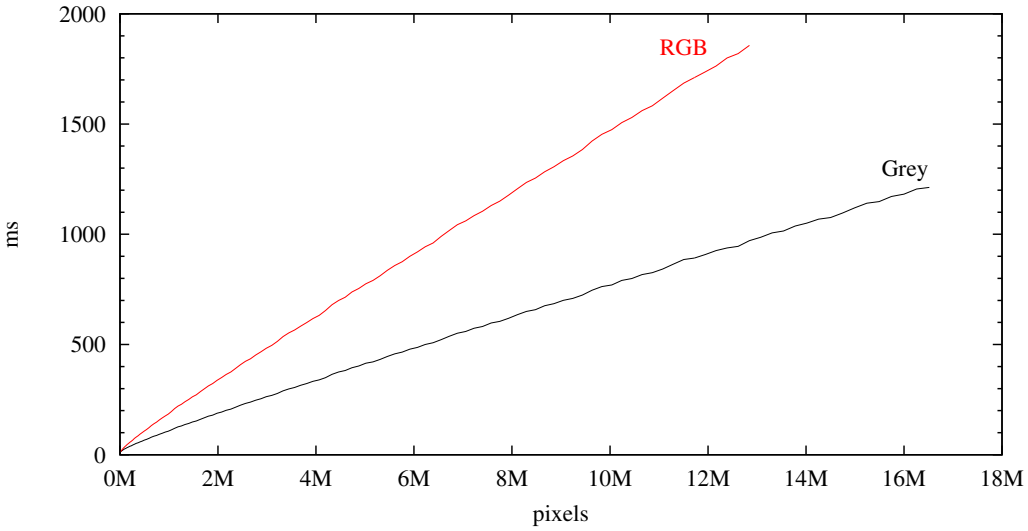



**Figure 4.3**: Scaling of the anisotropic diffusion algorithm over the image size on squarish grey-valued and colour-valued images, with parameters $T = 500$, $d = 3$, $\lambda = 1.0$, $\sigma = 1.0$, $\rho = 1.0$.

Figure 4.3 visualises this benchmark on grey-valued images, and on colour-valued three-channel images. In both cases, we obtain a linear scaling in the image size. Colour images are processed in less than twice the time than grey images of the same size. This is because the additional effort is largely required in the explicit solver kernel, where all operator entries can be re-used from on-chip memory for all image channels.

Moreover, note that the maximal image size possible to process with our algorithm is limited by the memory resources of the graphics card. On the GTX 480 with 1.5GB RAM, this means that the largest grey image that can be computed has the size of $4\,096 \times 4\,096$ pixels. Colour images are even limited to a maximal size of $3\,584 \times 3\,584$ pixels, which is a consequence of the higher memory consumption in this case.

However, despite its simplicity, our FED-based algorithm seems to be much faster than the approach from [BLF+07]. On a volume of size $256 \times 256 \times 128$ voxels, they report a runtime of 0.51 seconds per iteration. If we assume that this performance can be carried over to the 2-D case, and if we double the performance due to the developments in the GPU market, we obtain about 250 milliseconds for one iteration on an image with 8.4 megapixels. Even if we again double this performance to compensate for the omitted operations in the third dimension, their approach must require less than five iterations to have a similar runtime as FED. It is unlikely that any algorithm can yield the desired quality with only four (inner) iterations.

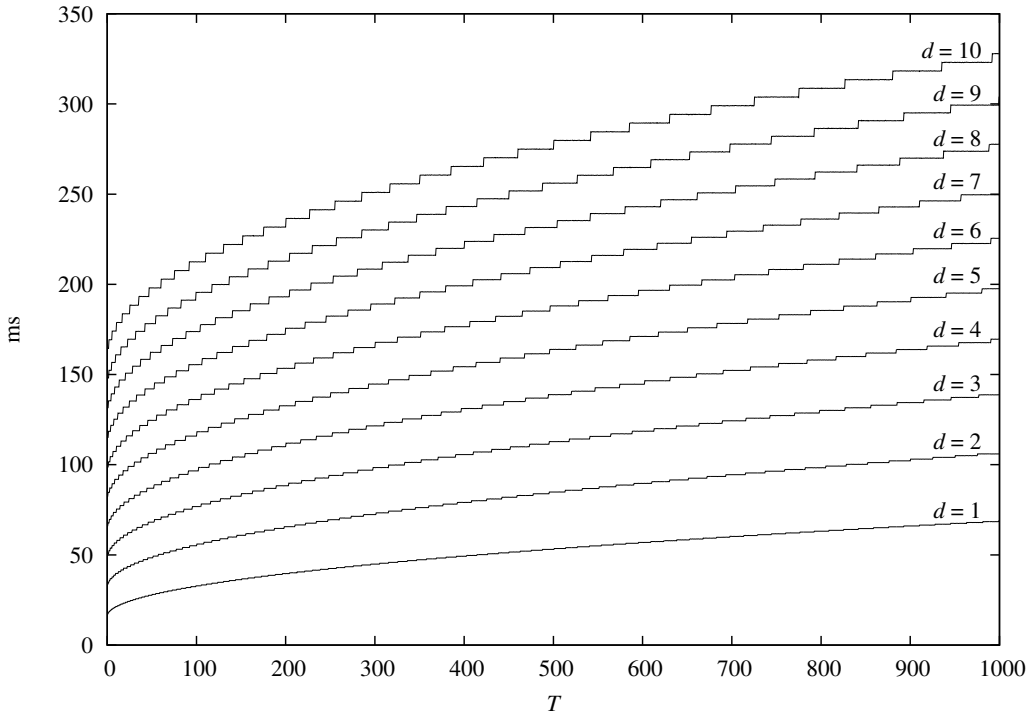



**Figure 4.4**: Scaling of anisotropic diffusion over the stopping time $T$ on a grey-valued image of size 1024×1024 pixels, with $\lambda = 1.0$, $\sigma = 1.0$, $\rho = 1.0$.

### 4.5.3 Runtime Scaling on Stopping Time

As a next step, we evaluate the scaling behaviour of our algorithm under different varying stopping times $T$ and FED cycles $d$. The choice of these two parameters is application-dependent, which is in particular troublesome as both $T$ and $d$ also take the main responsibility for the runtime of the process. Hence, this experiment shows how the algorithm behaves on different characteristics of inputs, rather than on different sizes.

For a better visibility, we first compare grey-valued images in the size $1\,024 \times 1\,024$ pixels. This is shown in Figure 4.4. By (4.13), the stopping time $T$ depends quadratically on the number of steps $n$. Because the runtime of the process depends linear on $n$, we observe a characteristic sublinear dependency between $T$ and the runtime. The step-wise increase of these graphs is related to the rounding in the algorithm from Section 4.3. Each step corresponds to a fixed number $n$ of inner steps.

The scaling behaviour over the number of FED cycles $d$ is dominated by the constant overhead of the operator update. This becomes obvious if we consider the linear scaling in $d$ for very small stopping times, and compare

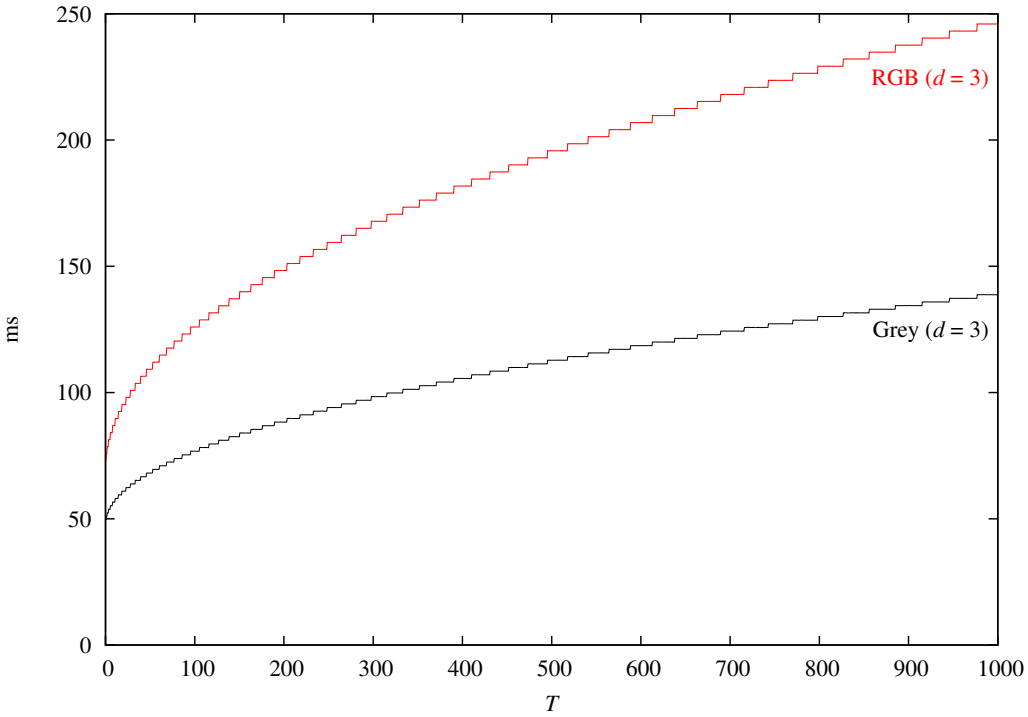



**Figure 4.5**: Scaling of 3 FED cycles for anisotropic diffusion over the stopping time $T$ on grey-valued and colour-valued images of size 1024×1024 pixels. Again, we use a contrast parameter of $\lambda = 1.0$, and smoothing $\sigma = 1.0$, $\rho = 1.0$.

the relative differences with the ones for large stopping times.

Figure 4.5 compares the grey-valued setting for $d = 3$ from Figure 4.4 with the colour-valued variant. The latter scales similar to the grey-valued variant, but has higher time requirements. These overheads are composed from two contributions. The time needed to update the nonlinear operator over more than one channel appears as a constant overhead which can particularly be seen for small stopping times $T$. In contrast, the additional time lost during the explicit update step appears as a contribution which is variant in the stopping time. It causes the two graphs to differ more the larger the stopping time is chosen.

As a final remark to this experiment, note that we have evaluated the impact of all runtime relevant parameters. In this chapter, we analysed the stopping time $T$, the number of FED cycles $d$, and the image size. Besides these, there are only two other parameters which have a minor impact on the runtime of the process: the standard deviations $\sigma$ and $\rho$ for the smoothing at local scale and at integration scale, respectively. Their influence has already been thoroughly evaluated in Chapter 3.

### 4.5.4 Runtime Comparison to CPU

In the last experiment for this chapter, we compare the runtime of our novel GPU-based algorithm against those of classical and efficient algorithms on the CPU [1]:

1. The **explicit scheme** with time step $\tau = 0.125$ and operator updates after every step represents the simplest, but potentially slowest method to solve anisotropic diffusion models. While these methods are very well suited to describe the non-linear process most accurately, they are also potential subject to discretisation artefacts and approximation inaccuracies (see Section 4.5.1).

2. An **implicit scheme** approximates the process with larger time steps, in our case with $\tau = 50$. Each of these steps requires the solution of a large system of equations, which is done in this case by using a conjugate gradient approach [Mei05]. Because of the approximation of the non-linear process by large linear steps, this process is prone to yield a higher approximation error to the desired solution, but is also less prone to numerical inaccuracies.

3. The CPU-based **FED scheme** also uses large linear time steps to yield results that are similar to the ones obtained by implicit diffusion schemes. Based on our experiment from 4.5.1 where FED with $d = 3$ cycles yielded very similar results to the implicit solutions, we use the same parameter in this experiment. However, note that this method represents one of the fastest approaches for this purpose on the CPU and forms the reference for our speedup computations. Hence, any increase of $d$ for numerical reasons does have an impact to the absolute runtimes, but not to the measured speedup (as long as it is performed on the CPU and the GPU simultaneously).

For this experiment, we again apply all algorithms on an Intel Core2 Duo E8200, in which the GPU-based method is executed on an NVidia GeForce GTX 480. All algorithms were adapted such that they use the same discretisations and optimisation levels. As a sample stopping time, we again choose $T = 500$.

Table 4.1 shows the results of this experiment. Although FED is much faster than the implicit and the explicit scheme on the CPU, it is again outperformed by the new GPU-based algorithm. While the speedups are

[1] My thanks go to Joachim Weickert and Sven Grewenig for providing reference implementations running on the CPU.

**Table 4.1**: Runtimes for anisotropic diffusion with FED using $T = 500$ on the GPU, compared to the corresponding algorithm on the CPU, and to explicit and implicit diffusion on the CPU. All times are given in seconds.

| | | CPU | | | GPU | Speedup |
|---|---|---|---|---|---|---|
| **Size** | **Type** | **Explicit** | **Implicit** | **FED** | **FED** | **FED** |
| **$256^2$** | **Grey** | 48.71 | 0.77 | 0.07 | 0.02 | **2.91** |
| | **RGB** | 62.06 | 2.24 | 0.08 | 0.03 | **2.83** |
| **$1\,024^2$** | **Grey** | 878.93 | 16.41 | 1.19 | 0.11 | **10.52** |
| | **RGB** | 1 140.59 | 46.12 | 1.46 | 0.20 | **7.43** |
| **$2\,048^2$** | **Grey** | 3 549.53 | 68.78 | 4.82 | 0.35 | **13.78** |
| | **RGB** | 4 620.50 | 186.50 | 5.88 | 0.65 | **9.00** |
| **$4\,096^2$** | **Grey** | 14 287.85 | 280.92 | 19.84 | 1.23 | **16.13** |
| | **RGB** | 18 655.53 | 761.42 | 24.12 | n/a | **n/a** |

moderate for small image sizes, they quickly grow to 10–16 for images in the range of $1\,024^2$ to $4\,096^2$ pixels. This is in particular remarkable if we consider that FED is already several orders of magnitude faster than classical algorithms such as explicit and implicit diffusion. On grey images of size $4\,096 \times 4\,096$ this causes the GPU-based FED algorithm to be more than 200 times faster than implicit diffusion, and even more than 10 000 times faster than explicit diffusion on the CPU.

Although the parallelisation speedup is significant, is does not reach the values obtained in context of homogeneous diffusion (see Section 3.6.4). Recall that two orders of magnitude are possible for algorithms such as recursive filtering. The reason for this striking discrepancy is the higher data throughput for anisotropic diffusion. Not only several iterations, but also frequent operator updates require data to be loaded from and stored to global device memory. In every case, multiple memory buffers are involved, such as for providing the input image, the operator weights, or image derivatives. Moreover, the arithmetic operations performed on these buffers are in general very inexpensive, such that load latencies cannot be effectively hidden behind arithmetics.

As it turns out, the runtime on colour-valued images deserves an additional discussion. On the CPU, we observe that the runtime for the implicit scheme expectedly grows by about three times, while the factor is much lower for the explicit scheme and the FED approach. The reason for the better runtime of the latter two might be their simple algorithmic structure,

such that compiler optimisations can be applied more effectively. Note in this context that operations on the CPU are in general computationally bound, such that the additional memory bandwidth does not significantly affect the measured runtime. This is fundamentally different for the GPU-based algorithm, where memory latencies play the most fundamental role for the runtime. Nevertheless, the runtime only increases by less than 100% when going from one to three channels, which is an indicator that concepts such as the re-use of operator weights from on-chip memory are successful. Still, the different cause of acceleration in both cases also affects the speedup factor, which is in general about 30% lower than for grey-valued images.

As a final remark to colour-valued images, note the missing entry for images of the size $4\,096 \times 4\,096$ pixels. As was already discussed in Section 4.5.2, it occurs because the memory requirements in this case exceed the physical resources of 1.5 GB. With future graphics cards and larger GPU memories, it will be straightforward to perform computations on images in this size.

## 4.6 Summary

In this chapter, we have created and evaluated a novel GPU-based algorithm for anisotropic diffusion. It is based on the recent Fast Explicit Diffusion (FED) scheme by Grewenig *et al.* [GWB10]. Our algorithm is designed for a very general setting that can be specialised to many modern diffusion schemes. As a model, we use the new Edge and Coherence Enhancing Diffusion (ECED) by Weickert [Wei11a]. It constitutes a natural combination of the famous anisotropic diffusion models by Weickert [Wei98], and is as such also a generalisation of isotropic diffusion models. This versatility also carries over to the discretisation of the problem. For this purpose, we use a new discretisation scheme by Weickert [Wei11c] which generalises ideas from [WSW08]. This discretisation is algorithmically very challenging, and can easily be specialised to many traditional finite difference discretisations.

On a modern graphics card, this new algorithm yields speedup factors of up to 16 over an FED scheme on the CPU. Because FED is already very fast compared to classical numerical schemes, this means that our new algorithm is up to 200 times faster than an implicit scheme, and even up to 10 000 times faster than an explicit scheme on the CPU. The key to the efficiency of this new algorithm is a combination of well-scalable components. Both the FED scheme, as well as all steps that are required to evaluate the non-linear system matrix, are data-parallel. As such, they are perfectly

suited for massively parallel architectures such as GPUs. Besides these, our algorithm only consists of homogeneous diffusion for the smoothing at local scale and integration scale, which can be efficiently computed with algorithms presented in Chapter 3.

However, we also observe that our anisotropic diffusion algorithm does not scale as well as many GPU-based algorithms for homogeneous diffusion. This is the result of a higher memory throughput between graphics memory and the GPU. To this end, our parallelisation efforts also illuminate a fundamental problem of many PDE-based algorithms in visual computing. They are largely memory-bound, such that the real problem for modern parallel hardware is not the supply of compute resources, but the provision of a suitable memory bandwidth. Algorithms can benefit from this insight if they require a smaller memory throughput as a competing method. In the following chapters, we frequently come back to this essential insight.

# Chapter 5

# PDE-Based Image Inpainting

*But nature flies from the infinite,*
*for the infinite is unending or imperfect,*
*and nature ever seeks an end.*
Aristotle

## 5.1 Motivation

All algorithms that we have designed so far solve parabolic PDEs. Hence, they are well suited to describe the evolution of a diffusion process, as they yield the solution for any given stopping time. However, the requirements of some applications even go far beyond this goal. Instead of the solution at a finite time, *elliptic* approaches aim at a steady state that is established as the time approaches infinity. In this chapter, we discuss a GPU-based algorithm that approximates infinite diffusion stopping times in a very small runtime.

Elliptic processes of this kind are of great use when acquired data is non-dense, or when partitions of this data are unreliable. A prominent example are image regularisation techniques which allow to handle ill-posed problems [TA77, BPT88]. Such variational methods enjoy a multitude of applications, in particular for problems in the field computer vision. In motion estimation approaches, so-called optic flow methods, they smooth away outliers in the measured data and fill in information where there is none available [HS81, NE86]. Efficient regularisers are still important ingredients for modern, highly accurate optic flow methods [ZBW11b]. Chapter 6 is concerned with details about an efficient optic flow algorithm on the GPU.

Closely related to optic flow are variational stereo vision applications which compute 3-D surfaces from a series of 2-D input data [ADSW02, BAS10]. Both techniques can even be combined to yield structure and 3-D optic flow simultaneously [HD07, VBZ+10].

In the recent years, the class of *PDE-based image inpainting* techniques enjoyed a high popularity. Such methods recreate missing parts of an image by filling in information from surrounding areas. As a fundamental difference to the aforementioned regularisation approaches, however, these methods are supplied by a binary confidence mask which indicates missing image regions. This allows to fill in holes while the given image information is left unaltered.

Image inpainting is a robust tool for image reconstruction purposes [MM98]. Impressive examples for the disocclusion of manually annotated defect locations can be found in [MM98] and [BSCB00]. However, image inpainting is also applied in a number of automatic and semi-automatic schemes for special reconstruction applications. One example is the dejittering of interleaved television footages [GBW09].

A major factor for the recent popularity of PDE-based inpainting strategies might be their application for image compression. Such lossy compression schemes use the fact that a few selected pixels suffice to describe the whole image with very low errors [BL77, AD96]. The remaining areas are reconstructed by diffusion inpainting algorithms [GWW+08, BBBW09]. In special cases, such diffusion-based image compression schemes even outperform modern conventional approaches such as JPEG 2000 [SWB09, MW09, MBWF11].

Similar ideas are applied in gradient domain editing [EG01, OBW+08, MP08]. These approaches do not exploit the fact that edges can be compactly stored, but that they contain most descriptive features of the image. By deleting, modifying, or adding edges, the image can be edited without destroying its general appearance. Low-frequent differences such as shading or shadow are automatically reproduced by the inpainting operation.

In view of this multitude of applications and the complexity of the problem, it is not surprising that there are several GPU-based algorithms in the literature. In [GWL+03], the authors design a bi-directional multigrid solver for the Poisson equation with a traditional GPGPU approach on the OpenGL pipeline. This corresponds to performing linear diffusion in the holes, while the known pixels act as non-trivial Dirichlet boundaries to the process. Similar, but more simplified and thus faster algorithms were proposed in [OBW+08] and [MP08]. In [JCW09] the authors propose to combine discrete Laplacian stencils in different sizes to transport data faster over the image domain.

Nevertheless, it seems there is no GPU-based anisotropic diffusion image inpainting algorithm available in the literature so far. A related work uses distance maps to propagate pixel data according to a set of predefined directions [RFSB10]. However, this procedure is not PDE-based. Moreover, its algorithmic structure is closely related to parallel solvers for the Eikonal equation [JW07] such that this approach has striking similarities with hyperbolic rather than with elliptic ideas.

In this chapter, we design an efficient GPU-based algorithm for arbitrary PDE-based image inpainting approaches. In a similar spirit as for nonlinear diffusion, we formulate a very general framework that can later be specialised to concrete applications. Our approach is anisotropic, and it uses a technically complex discretisation scheme. Later in this chapter, we additionally perform a specialisation to homogeneous diffusion inpainting which is more related to the previous work from the literature.

## 5.2 Image Inpainting

Let us now briefly review the mathematical background of PDE-based image inpainting. Given an image domain $\Omega \subset \mathbb{R}^2$, we distinguish two subdomains:

1. In $K \subset \Omega$, we assume pixel values to be known. These are the undistorted pixels in case of image reconstruction, or the stored pixels in case of image compression.
2. The set $\Omega \setminus K$ contains those pixels whose values are unknown. They must be reconstructed by means of diffusion image inpainting.

The affiliation of a point to either of these subsets is determined by the characteristic function $c$:

$$c(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in K \\ 0, & \text{else} \end{cases} . \tag{5.1}$$

Again, we use an anisotropic diffusion process as in (4.1). However, following the works from [GWW+08, SWB09], we do not perform tensor smoothing, but set $\rho = 0$. This leads to edge-enhancing diffusion (EED) in the domain $\Omega \setminus K$:

$$0 = -(1 - c(\boldsymbol{x})) \operatorname{div}\big(\big(\boldsymbol{D}(\boldsymbol{\nabla} u_\sigma \boldsymbol{\nabla} u_\sigma^\top)\big)\ \boldsymbol{\nabla} u\big) \ + \ c(\boldsymbol{x})(f - u)\,, \tag{5.2}$$

where $u$ is the sought solution and $f$ is the input image [GWW+08]. It is easy to check that a solution $u$ to this elliptic diffusion equation equals

the input $f$ at the known points, while it describes the steady state of a diffusion process in the unknown domain $\Omega \setminus K$.

Throughout our numerical experiments, we choose $\boldsymbol{D}$ similar to described in Section 4.2, but allow small changes. Besides the aforementioned elimination of the smoothing at integration scale, we also use the *Charbonnier diffusivity* [CBAB97] instead of (4.7):

$$g(s^2) = \frac{1}{\sqrt{1 + \frac{s^2}{\lambda^2}}}\,. \tag{5.3}$$

Both modifications are in accordance with the choices in [GWW+08] and [SWB09] which were shown to yield very good results for the application of image compression for real-world images.

Besides these settings, we discuss a slightly different setup in Section 5.6. These efforts aim at an efficient solution of the heat equation within $\Omega \setminus K$, which corresponds to $\boldsymbol{D}$ being set to the identity. Results of this type are less computationally intense than the anisotropic model, and are very well suited for the compression of cartoon-like images [MW09, MBWF11]. Moreover, note that this instance is comparable to the objectives for the gradient-domain works from [EG01, OBW+08, MP08]. This allows a fair comparison of the novel GPU-based algorithm against other parallel approaches from this field.

## 5.3 Cascadic FED

A common strategy to solve elliptic equations such as the one from (5.2) is to regard them as an energy function on $u$, and to minimise this energy by a gradient descent. This leads to the parabolic process

$$\partial_t u = (1 - c(\boldsymbol{x}))\,\mathrm{div}\big(\big(\boldsymbol{D}(\boldsymbol{\nabla} u_\sigma \boldsymbol{\nabla} u_\sigma^\top)\big)\ \boldsymbol{\nabla} u\big)\ +\ c(\boldsymbol{x})(u - f)\,. \tag{5.4}$$

This parabolic process finds the minimum of the energy for its stopping time $T$ approaching infinity. Hence, we need a numerical scheme which obtains large stopping times in a very short runtime.

As it turns out, the FED scheme is again a good candidate for a numerical scheme that fulfils these constraints. Similar to the procedure in Section 4.2, we use the new spatial discretisation scheme by Weickert [Wei11c], and complement it by an FED discretisation in time. In order to further accelerate this process, we embed the solver into a hierarchic coarse-to-fine strategy. This comes down to the *cascadic fast explicit diffusion (CFED)*

scheme from [GWB10]. In this section, we briefly review the essential steps necessary to use this scheme for anisotropic image inpainting.

In analogy to the FED update equation (4.17), we first write down a standard (non-cascadic) FED update equation for (5.2):

$$\boldsymbol{u}^{cn+i+1} = (\mathbf{1} - \boldsymbol{c}) \odot \left( (\boldsymbol{I} + \tilde{\boldsymbol{\tau}}_i \boldsymbol{A}(\boldsymbol{u}_\sigma^{cn})) \boldsymbol{u}^{cn+i} \right) + \boldsymbol{c} \odot \boldsymbol{f} \,. \tag{5.5}$$

In this context, we use $\odot$ to denote the component-wise vector product, and $\mathbf{1}$ for a vector of the same length as $\boldsymbol{c}$ whose entries are all 1.

Note that the steady state obtained by the diffusion on $\Omega \setminus K$ does not depend on the initialisation. Instead, it can be uniquely determined by the location and value of the pixels in $K$. However, the closer the initialisation resembles the desired solution, the less time requires the system to converge to this solution. This gives rise to use a cascadic approach which initialises a process on one scale with the prolongated outcome from a coarser scale [BD96, GWB10]. On coarse grids, long distances between known pixels are sampled with less points, such that they can be computed much more efficiently.

Given an image $\boldsymbol{f}$ and a mask $\boldsymbol{c}$, cascadic FED first restricts both inputs to a coarser scale. Let $\boldsymbol{I}_h^H$ denote a restriction operator which transfers an image from a grid with spatial distance $h$ to one with spacing $H$ [TOS01]. While the concrete form of this operator is implementation dependent, we require it to preserve a certain intuition of similarity between the representations on both grids.

As it turns out, this required similarity can be a problem for the restriction of the binary mask $\boldsymbol{c}$, because $\boldsymbol{I}_h^H \boldsymbol{c}$ is in general real-valued. While $\boldsymbol{I}_h^H \boldsymbol{c}$ can easily be re-binarised, such modification must account for the compatibility to $\boldsymbol{I}_h^H \boldsymbol{f}$. *Normalised convolution* is a convenient way to handle this shortcoming [KW93, Bru10, MBWF11]. Let superscripts $(h)$ and $(H)$ denote the representations of an image on the fine and coarse scale, respectively. Then, we compute the coarse image and mask by

$$\boldsymbol{f}_{i,j}^{(H)} = \begin{cases} \frac{\left(\boldsymbol{I}_h^H (\boldsymbol{f}^{(h)} \odot \boldsymbol{c}^{(h)})\right)_{i,j}}{\left(\boldsymbol{I}_h^H \boldsymbol{c}^{(h)}\right)_{i,j}} \,, & \left(\boldsymbol{I}_h^H \boldsymbol{c}^{(h)}\right)_{i,j} \neq 0 \\ 0 \,, & \text{else} \end{cases} , \tag{5.6}$$

$$\boldsymbol{c}_{i,j}^{(H)} = \begin{cases} 1 \,, & \left(\boldsymbol{I}_h^H \boldsymbol{c}^{(h)}\right)_{i,j} \neq 0 \\ 0 \,, & \text{else} \end{cases} . \tag{5.7}$$

Note that (5.7) represents the intuitive binarisation in which all non-zero entries are reset to 1. The resampling of the image in (5.6) respects this new setting by expanding the area of known pixels to the new area. Hence,

a new image pixel is created by the weighted average of the known pixels contained in it, or by 0 if it does not contain known pixels.

As a next step, let us discuss a suitable way to prolongate a fine-grid solution $\boldsymbol{u}^{(h),0}$ out of a coarse-grid solution $\boldsymbol{u}^{(H),t}$. Since we know the optimal solution at known grid points – they simply correspond to the input – we adapt the prolongation operation to the fine grid mask. This leads to [Bru10]:

$$\boldsymbol{u}^{(h),0} = (\mathbf{1} - \boldsymbol{c}^{(h)}) \odot \left(\boldsymbol{I}_H^h \boldsymbol{u}^{(H),t}\right) + \boldsymbol{c}^{(h)} \odot \boldsymbol{f}^{(h)}, \tag{5.8}$$

where superscripts are again chosen as above.

The steps described above can be applied recursively up to an arbitrarily coarse representation of the problem description. To this end, we obtain an algorithm as follows:

1. If desired, obtain initialisation by:

   a. Restrict $\boldsymbol{c}$ and $\boldsymbol{f}$ by (5.6) and (5.7).
   b. Perform recursive call on coarse grid.
   c. Prolongate solution by (5.8) and initialise fine-grid solution.

   Else, initialise solution with $\mathbf{0}$.
2. Perform FED with given parameters on (5.5).

The evaluation of the condition in 1. depends on algorithmic considerations. Since (5.6) has the side effect of densifying the image, it is usually a good idea on sequential hardware to restrict the problem down to the point on which all pixels are known. On massively parallel hardware such as GPUs, however, the algorithm scales worse the less pixels it is executed on. Hence, it is usually a better choice here to limit the minimal element counts in both dimensions to a constant.

## 5.4 GPU-Based Algorithm

The design of an efficient GPU-based algorithm requires us to focus on two critical aspects. On the one hand, we must parallelise all operations, and take care of a beneficial scaling behaviour on the chosen scales. We have already designed algorithms for most operations in context of anisotropic diffusion (see Chapter 4). Hence, a new layout is only required for the new resampling operator. On the other hand, we must pay attention to the memory bandwidth of the FED solver on each level. As it turns out, this throughput is much higher for inpainting than it is for diffusion. We go into detail about this aspect in the following paragraph.

### 5.4.1 FED

Let us first analyse the parallelisation of the FED solver from (5.5), and the data patterns occurring in this case. On a GPU, we want all active threads to perform the same operation at the same time. This procedure protects the algorithm from branch divergence, and thus allows for a high efficiency. As a consequence, we evaluate both summands of (5.5) at all points. This causes intermediate results to be always computed, even if they are later discarded by a multiplication with zero. In order to optimise the memory bandwidth for this process, we can pursue two strategies:

1. We can input the symmetric system matrix for diffusion as in Section 4.4, plus the mask and the old solution in a $3 \times 3$ neighbourhood. Since the weights from the matrix are uniquely representable by 4 values per pixel, we obtain $9+9+4 = 22$ loads for a pixel of a grey-valued image, or $3 \cdot 9 + 9 + 4 = 40$ loads for a colour-valued image. Note that the mask and each image channel are read from one texture, each, such that most reads are cached.

2. Alternatively, we can already store the 'masked' weights during the setup phase of the operator. Since the operator is kept stable over several steps, the additional load latencies from this stage can usually be neglected. Because the resulting matrix is no longer symmetric, we need to load at least 8 entries per pixel. This leads to a minimum of $9 + 8 = 17$ loads per grey-valued pixel, or $3 \cdot 9 + 8 = 35$ loads per colour-valued pixel.

Although in the first case, many data points are read from the same texture cache, the totally lower bandwidth of the second case makes the process run slightly faster. In experiments, it even turns out that loading the main diagonal of the operator instead of computing it inline pays off in the runtime. This might be due to reduced data dependencies within the solver. Hence, we use the second strategy within our anisotropic algorithm, but leave the first one as a more efficient option for special applications such as linear diffusion inpainting.

### 5.4.2 Resampling

As the most important complement to the algorithm from Chapter 4, we now discuss efficient restriction and prolongation operators. Since the additional normalisation from (5.6)–(5.8) are straightforward, we focus on a technique to realise the real-valued operators $\boldsymbol{I}_h^H$ and $\boldsymbol{I}_H^h$. The CUDA kernel

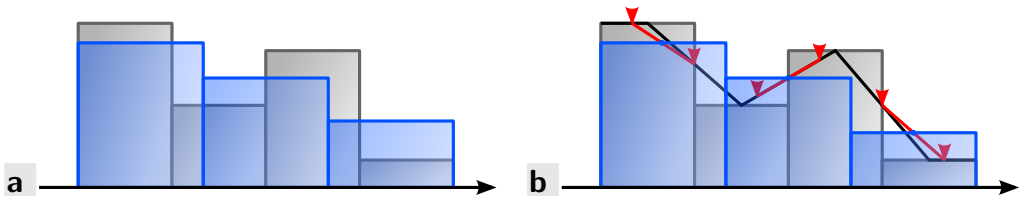



**Figure 5.1**: Application of 1-D restriction operators (from 'black' to 'blue'). **a.** Area-based resampling [BWF+05]. **b.** Fast texture-based resampling.

for each operator then handles both the mask and the image jointly, which also allows for a fast normalisation in shared memory.

Restriction and prolongation are crucial operations in multi-scale algorithms. On the one hand, they must be very accurate in finding a 'suitable' representation of a given problem on another grid. Even small errors induced by this process must later be smoothed out by expensive iterative solvers, which involves a high runtime of the overall process. On the other hand, they must not consume too much runtime by themselves, as their runtime constitutes a significant partition of the process runtime. Hence, let us briefly focus on an efficient technique for this purpose on the GPU. We begin with the restriction operator $\boldsymbol{I}_h^H$.

Restriction operators from the literature can be classified by their functionality. Often, it is assumed that both the input and the output have side length $2^n + 1$ with some integer $n$ [TOS01]. Transferring a signal from a grid of size $2^{k+1} + 1$ to $2^k + 1$ then works by simply discarding even-indexed grid points, or by a smart averaging on a $3 \times 3$ basis [TOS01]. However, once differently sized images or arbitrary resampling factors are given, this technique is not applicable. In such cases, works from the literature often use an area-based restriction scheme [BWF+05]. Such methods do not only take into account the location of a pixel, but also its area in the image domain. A coarse representation of an image is then found by a re-distribution of masses. Figure 5.1a visualises this idea with a simple 1-D example. As a first step, both pixel grids are normalised to the same width, while their individual sampling frequencies are different. Then, each pixel area of the coarser grid takes the integral over grey values in the corresponding area of the finer grid. This yields a canonical representation of one image on a different scale.

We need a resampling strategy of the second kind for our application to diffusion inpainting, because we want to allow input images to have arbitrary sizes. However, area-based restriction is very costly on GPUs. Even if we assume restriction factors in the interval $(1, 2]$, one output pixel

can depend on 4, 6, or 9 input pixels. Depending on whether some pixel cells of both grids coincide or not, there are even 16 different cases that must be handled individually. In a massively parallel layout, this case distinction causes a high degree of sequentialisation. Each case is executed by those threads for which the condition holds, while the remaining pixels are idle.

As a very fast alternative to this approach, we exploit the bilinear interpolation capabilities of the texturing unit. Figure 5.1b shows a simple 1-D example of this novel strategy. The input image is read as an *interpolating* CUDA texture, which is visualised with a black line. Similar to area-based resampling, we normalise the width of both grids. This comes down to a translation of pixel coordinates of the output grid into the texture coordinate system of the input grid. To avoid aliasing, we obtain the value of a pixel $\boldsymbol{x}$ as the average over texture requests at locations $\tilde{\boldsymbol{x}} = (\boldsymbol{x}_1 \pm 1/4, \boldsymbol{x}_2 \pm 1/4)^\top$. This is denoted by red arrow heads and lines. Carried over to the 2-D setting, this strategy only requires the retrieval of four original samples per new sample. Because most of these memory loads are buffered by the 2-D aware texture caches, this operation is very fast. Moreover, the oversampling with 2 input samples per output sample allows aliasing-free restriction for arbitrary factors in the interval $(1, 2]$. Larger factors can be obtained by an increase in the number of sampling points.

In any case, all threads perform the same operation at the same time, such that texture-based resampling can be performed very efficiently on GPUs. Nevertheless, it yields different results to area-based resampling. This is also visible in Figure 5.1. While the values for the first two samples are identical for both strategies, the third grey value is lower than for area-based resampling. However, experiments confirm that both resampling strategies seem to yield equally reasonable representations of a problem, such that the convergence rate in both cases is very comparable.

The corresponding prolongation operator $\boldsymbol{I}_H^h$ can even be realised with less computational effort. Values for pixels on the new, finer grid can conveniently be sampled on a bilinearly interpolated texture. Note that aliasing cannot occur in this case.

## 5.5 Experiments

### 5.5.1 Quality and Parameters

As a first experiment with our developed algorithm, we would like to analyse its convergence behaviour over a varying number $d$ of FED cycles per level. While most of the remaining parameters are problem dependent, $d$ is

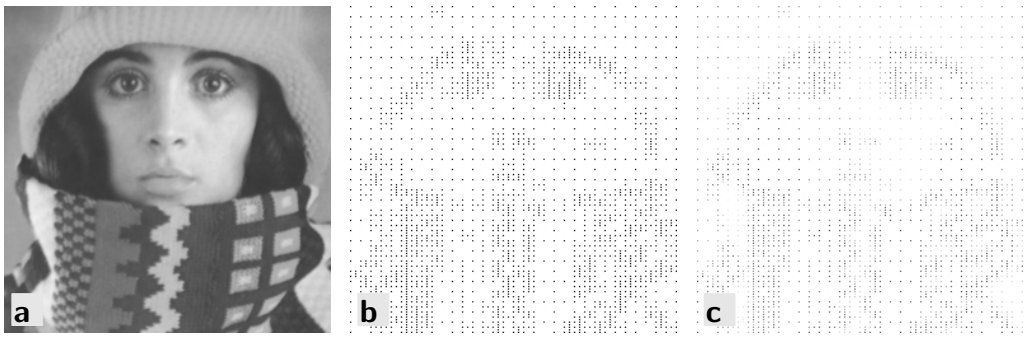



**Figure 5.2**: Inpainting experiment from [SWB09]. **a.** *Trui*, extended to 257×257 pixels. **b.** Inpainting mask $\boldsymbol{c}$. **c.** Mask pixels, overlaid with image grey values.

a FED-specific variable which we can expect to be largely invariant under the input. To this end, we take the inpainting mask from [SWB09, Figure 1], and perform EED inpainting as in the original publication. This setup is also shown in Figure 5.2. For different $d$, we perform a series of benchmarks with varying stopping times. Each of these configurations leads to a different runtime of the algorithm. We want to reproduce the results from [SWB09] with our algorithm, but with the same model parameters as in the original paper. Hence, we search a configuration with the lowest runtime that obtains a reconstruction with a similar error as their cascadic successive over-relaxation (CSOR)[1].

The results of this experiment are visualised in Figure 5.3. The sought error obtained by the CSOR reference implementation is denoted by a runtime-independent line. We can see that no CFED configuration with less than $d = 8$ iterations reliably obtains the anticipated error for a wider range of configurations. Because the graphs for $d \geqslant 8$ all reach this error level, $d = 8$ seems to be a good choice for our further experiments.

Note that the presented graphs have different supports. The reason for this is that implementations of FED on real architectures suffer from numerical inaccuracies which cause the process to become unstable for very large stopping times per cycle (see Section 4.3 for details). This drawback seems to be amplified by the cascadic approach described in this chapter. Hence, the supports of the graphs account for the admissible range of parameters in which our implementation produces stable results.

It remains to determine a suitable stopping time which yields the desired approximation error with a low runtime. For any fixed $d$, the runtime scales

[1] My thanks go to Christian Schmaltz and Andrés Bruhn for providing a sample implementation of the CSOR algorithm from [SWB09].

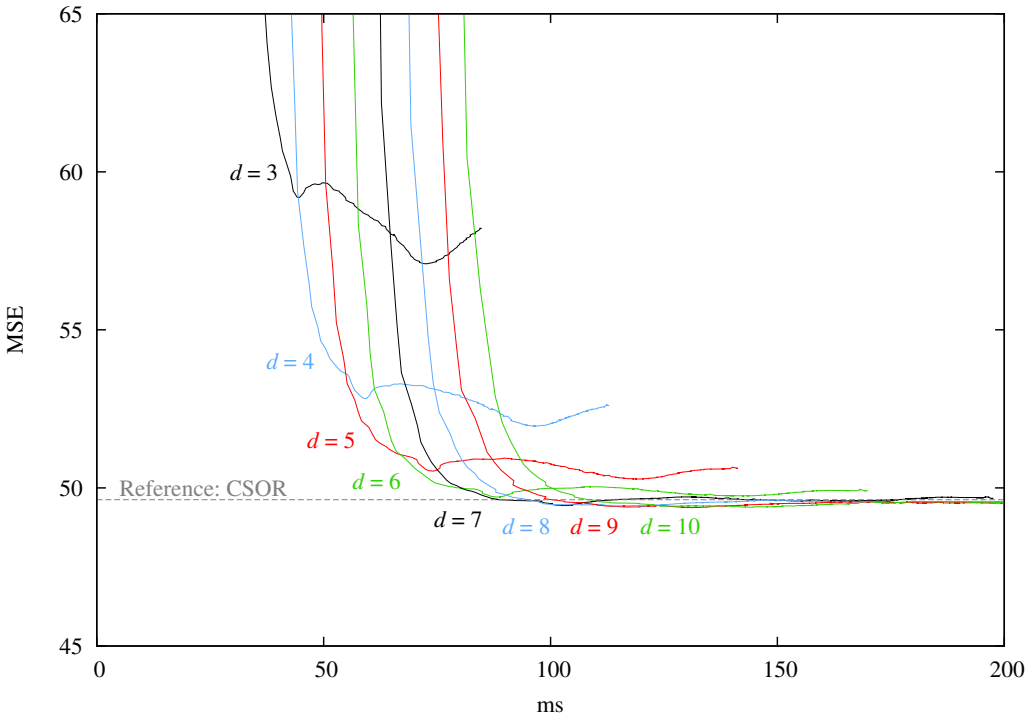



**Figure 5.3**: Reconstruction error on *Trui* with different numbers of iterations $d$ on the runtime of the algorithm.

by a monotonic function on the stopping time of the process. This allows to plot the stopping time against the approximation error, and to pick the smallest stopping time as a configuration for the smallest runtime of the process. This experiment is shown in Figure 5.4. Again using the CSOR reference as the desired approximation error, we find that a stopping time of $T = 2000$ is fully sufficient to obtain the desired error.

This leads to inpainting results as shown in Figure 5.5. As expected, the solutions obtained by the CFED method are indistinguishable from the CSOR reference. As a consequence, both methods obtain about the same reconstruction error to the original.

## 5.5.2 Runtime

### Comparison to CPU

Let us now compare the runtimes of the novel GPU-based algorithm against two sequential algorithms on the CPU:

1. The first method to compare against is a cascadic FED implemen-

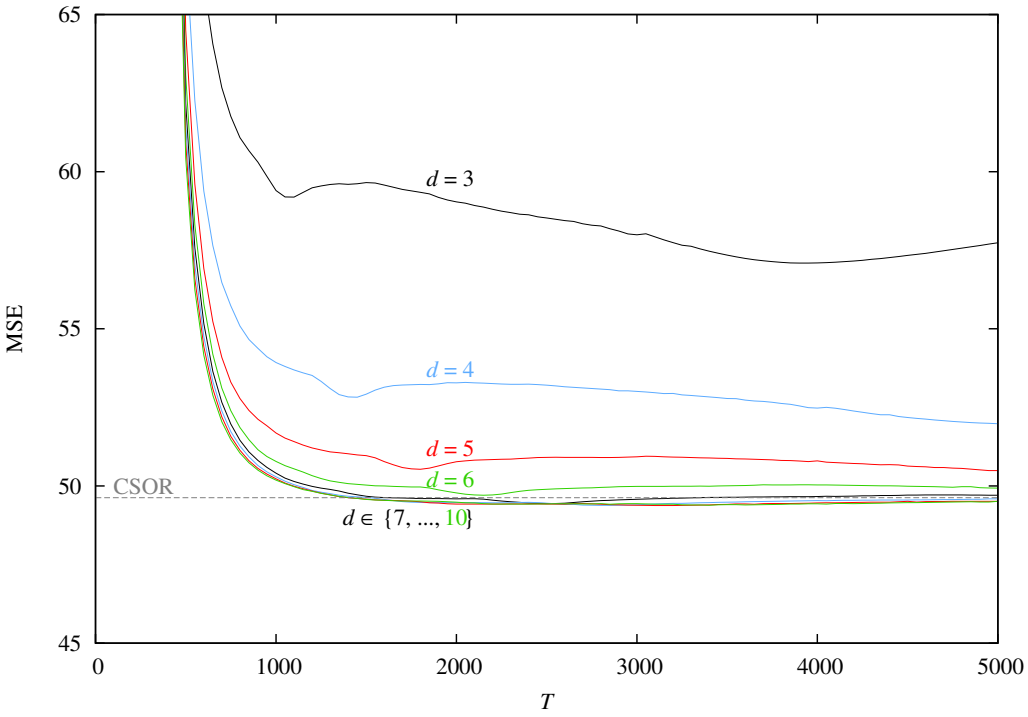



**Figure 5.4**: Reconstruction error on *Trui* with different numbers of iterations $d$ on the stopping time $T$ of the algorithm.


**Figure 5.5**: Inpainting results with different numerics. **a.** *Trui* (257×257 pixels) from Figure 5.2. **b.** CSOR, parameters as in [SWB09], 211 ms on CPU, MSE = 49.62. **c.** CFED, $T = 2000$, $d = 8$, 136 ms on GPU, MSE = 49.59.

tation similar to the one from [GWB10], but with the discretisation from [Wei11c]. The speedup obtained in this experiment tells us the plain parallelisation benefit.

2. Secondly, we perform the same benchmarks with the cascadic successive over-relaxation (CSOR) method used in [SWB09], which is again extended by the more complex discretisation scheme. This solver belongs to the state-of the art methods for image compression with anisotropic PDEs. Hence, the observed speedup in this case corresponds to the overall benefit of the novel algorithm against one of the best algorithms on the CPU.

Note that these two methods have fundamentally different mathematical properties. While the CFED algorithm approximates the elliptic process by large parabolic steps on each level, CSOR directly solves the elliptic problem. In the latter case, the known points in $K$ play the role of Dirichlet boundary values.

In order to provide a fair comparison, we select the parameters for all algorithms such that they obtain the same reconstruction error on the *Trui* image shown in the last section. On each level, this corresponds to $d = 8$ iterations and a stopping time $T = 2000$ for the CFED algorithms, and 5 iterations with 10 relaxations for the CSOR-algorithm as used in [SWB09]. These parameters are applied to images in different sizes. Without loss of generality, we use white images and random masks with a coverage of 10% for this experiment, since we are only interested in the runtime (but not the outcome) of this process.

Note that using a fixed parameter set for arbitrary images is in general not an optimal choice. In fact, settings such as the stopping time depend on the percentage of known points in the discrete image domain, as well as on the maximal distance between two known points. However, this is the same for all evaluated algorithms, such that we can expect relative speedups to remain valid. Moreover, this fixed parameter set allows us to analyse *algorithmic* rather than *numeric* behaviours, which is in accordance to the main focus of this work.

Table 5.1 shows the results of this experiment. Note that because of the high memory consumption of all algorithms, the supplied 1.5 gigabytes of GPU memory do not suffice to execute three-channel images in the size $4\,096 \times 4\,096$ pixels. As a consequence, the corresponding benchmarks are missing in the table.

Let us first discuss the direct parallelisation benefit, i.e. the speedup between the GPU-based and the CPU-based CFED algorithms. While for small images, this factor is rather moderate, it quickly grows to 30 to 50 for

**Table 5.1**: Runtimes and speedups for cascadic FED on the GPU, compared to the corresponding algorithm on the CPU ('Speedup CFED') and to cascadic SOR on the CPU ('Speedup CSOR'). All times are given in milliseconds.

| | | CPU | | GPU | Speedup | |
|---|---|---|---|---|---|---|
| | | **CSOR** | **CFED** | **CFED** | **CFED** | **CSOR** |
| $\mathbf{256^2}$ | **Grey** | 202.19 | 711.74 | 107.56 | **6.62** | **1.88** |
| | **RGB** | 433.95 | 1 952.42 | 129.57 | **15.07** | **3.35** |
| $\mathbf{1\,024^2}$ | **Grey** | 4 063.54 | 13 195.41 | 580.96 | **22.71** | **6.99** |
| | **RGB** | 8 401.91 | 36 985.29 | 949.78 | **38.94** | **8.85** |
| $\mathbf{2\,048^2}$ | **Grey** | 16 534.26 | 54 228.03 | 1 772.06 | **30.60** | **9.33** |
| | **RGB** | 34 423.84 | 150 348.34 | 3 108.26 | **48.37** | **11.07** |
| $\mathbf{4\,096^2}$ | **Grey** | 66 638.59 | 218 958.42 | 6 299.53 | **34.76** | **10.58** |
| | **RGB** | 139 431.14 | 604 902.74 | n/a | **n/a** | **n/a** |

images of the size $2\,048 \times 2\,048$ pixels. Since we truncated the data structures in both cases such that they do not exceed a minimal size of $64 \times 64$ pixels per layer, this speedup again indicates the good scaling behaviour of GPUs on explicit schemes. However, note that this approach does not reach the speedups measured in context of linear diffusion (see Table 3.1), which is a result of the higher memory dependencies involved in this case.

This memory-boundedness becomes also obvious if we compare the behaviours of both CFED algorithms on grey-valued and on colour-valued images. While the CPU-based variant scales almost linearly with the number of channels, the GPU based technique computes three-channel images in less than twice the time of a one-channel image. This is because entries for the non-linear operator can be re-used for all channels, and because computations on on-chip memory require a negligible partition of time claimed by the many memory loads.

One principal caveat of the novel approach can be seen if we compare the runtimes of the CPU-based reference runs. On this architecture, CFED is more than 3.2 times slower than CSOR when it comes to obtaining the desired low reconstruction error. This drawback also carries over to our GPU-based implementation. It is still about one order of magnitude faster than the best method on the CPU which justifies the use of GPUs for this purpose. However, this speedup is relatively low if we compare it to the pure parallelisation benefit.

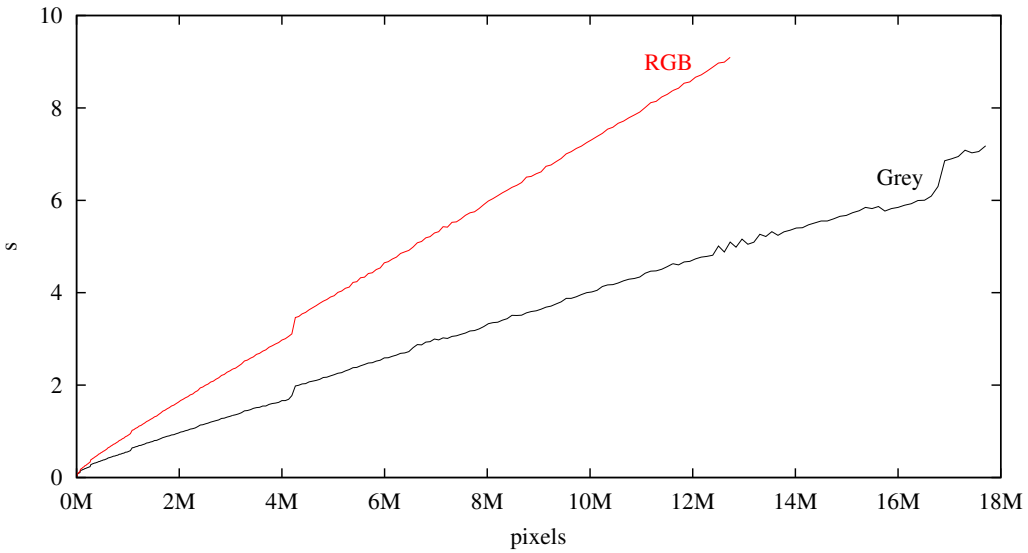



**Figure 5.6**: Scaling of CFED with $d = 8$ iterations per level over grey-valued and RGB-valued images in different sizes.

Despite this apparent restriction, the basic idea of FED seems to bear an amazing potential, even for elliptic processes. In this chapter, we have used FED-based numerics over SOR-based solvers because they possess a parallelisation-friendly algorithmic structure. While SOR methods are *per se* sequential, FED is entirely data-parallel. However, the difference between the observed runtime does not seem to be related to the same property, but to the gap between solvers that are tailored to the parabolic case on the one hand, and those that perform best on elliptic problems on the other. As a consequence, it might be worthwhile to focus more on the direct solution of the elliptic problem, rather than approximating it by a gradient descent with parabolic solvers. Promising candidates for this purpose are over-relaxed Jacobi schemes [Mei05, OBW+08]. Here, it could even be possible to use FED time steps to drive the over-relaxation of the scheme [Wei11b]. Evidence for the principal functionality of such approaches is given in the next chapter. However, a detailed analysis and evaluation of the applicability and runtime of such Fast Jacobi schemes to PDE-based image inpainting remains as future work.

### Scaling on Image Size

For a better understanding of the runtime behaviour of our novel algorithm, let us briefly analyse its scaling behaviour over different image sizes. Fig-

ure 5.6 visualises these results. As in the previous experiment, we use a constant setting of $d = 8$ and $T = 2000$ iterations. The overall linear scaling behaviour is only interrupted by characteristic discontinuities whenever the CUDA block grid exceeds multiples of the maximal extent. As it was observed before, the version taking colour-valued images scales up to a size of about 12.5 megapixels, while grey-valued version still works on images with about 17.5 megapixels. Above these limits, the memory requirements exceed the available resources.

## 5.6 Application: Realtime Video Inpainting

### 5.6.1 Scenario

In the introduction to this chapter, we have seen that many applications which require PDE-based image inpainting do not necessarily require complicated anisotropic PDEs. Instead, many approaches are based on homogeneous diffusion inpainting. Besides gradient domain techniques such as [EG01, OBW+08], this holds in particular for image compression of cartoon-like images [MW09, MBWF11]. One main criterion for using this simpler PDE over more complex ones is the existence of an analytical solution. Besides other aspects, this property allows a reliable designation of interpolation points.

In this section, we develop an interactive demonstrator for image compression with homogeneous diffusion. Our framework shall read a raw video stream from either a web-cam or the console input. This data is reduced to a few interpolation points using either the technique from [BBBW09], or the one from [MBWF11]. This sparse data is then interpolated by linear diffusion inpainting. A graphical frontend visualises all necessary intermediate solutions and the final outcome.

Note that we do not compress and decompress the sparse data in terms of entropy coding. These intermediate steps are *per se* sequential and closely related to a storage on hard discs. However, neither of these stages are interesting from a purely vision-motivated perspective. Moreover, they infer significant memory latencies to the system, since data must be additionally copied between the host machine and the graphics device.

### 5.6.2 Semantic and Analytic Image Compression

In the following, we distinguish two different strategies to thin the image down to a few significant inpainting points:

1. The **analytic** approach from [BBBW09] sets up on a continuous theory which relates the approximation quality during reconstruction to the preservation of defects of harmonicity. Such regions are identified by the magnitude of the Laplacian $\Delta u$. Hence, the approximation with homogeneous diffusion becomes better the more image regions with a large Laplacian magnitude are retained.

   In [BBBW09], the authors show that this theory can be carried over to a discrete setting. They suggest to rescale $|\boldsymbol{\Delta u}|$ such that its average value equals the one of an image that contains just as many white pixels on black ground as the desired number of mask pixels. The mask can then be obtained by *halftoning*, i.e. binarising, the rescaled magnitude. Note that we do not go into detail about halftoning at this point, since this field of research is subject of Chapter 7.

2. The **semantic** approach used in [MW09] is motivated by an analysis of the human perception of an image. Our visual system performs well in detecting edges, but does not pay much attention to flat regions. This motivates to reduce the image to thin lines on both sides of a dominant image edge.

   Following the original approach from [MW09], we obtain edges by a Marr-Hildreth edge detector with hysteresis thresholding [MH80, Can86]. This comes down to a detecting edges as zero-crossings of the discrete Laplacian $\boldsymbol{\Delta u}$, and to discard mismatches based on the local image gradient. The sought mask is obtained as the set of pixels to both sides of an edge.

In either case, the resulting mask $\boldsymbol{c}$ is multiplied to the dense image to obtain the sparse image $\boldsymbol{f}$ which could now be encoded and stored to hard disk. Instead of performing this encoding, we immediately continue with the inpainting of $\boldsymbol{f}$.

### 5.6.3 Implementation

#### Memory Transfer

One of the main challenges in the implementation of this real-time demonstrator is the low bandwidth between the CPU and the GPU. Unlike before, it does not suffice to measure the time the algorithm needs to inpaint the image. Instead, all that counts for a human observer is the latency between capturing a frame and displaying the result on the screen. This is particularly interesting if we use a web camera connected to the system.

As it turns out, such cameras provide a very compact representation of the video data they acquire. In the planar YV12 format they produce, each colour pixel occupies only 1.5 bytes. Recall that a single precision float representation of an RGB buffer requires 12 bytes per pixel. Hence, a good strategy is to upload this raw data to the GPU, and to decode it with CUDA kernels on the device.

For the visualisation of the results, we do not even need to copy buffers at all, since both the computation and the visualisation are performed on the same graphics card. CUDA offers a set of OpenGL interaction routines which allow to map buffers to the address spaces of both frameworks [NVi11b].

### GPU Kernels

A main difference to the previous setting is the exchange of the anisotropic diffusion model by homogeneous diffusion. As it turns out, we cannot apply any of the highly efficient linear diffusion techniques presented in Chapter 3 in this context. Even if we decouple the problem into a series of diffusion steps alternating with a reconstruction of the known points, this process already takes seconds to converge on relatively small images.

Instead, we apply a CFED strategy similar to the one for the anisotropic case. However, since we use a homogeneous diffusion model, this approach becomes very similar in spirit to iterated box filtering. In this context, the cascadic application of the solver simply corresponds to box kernels in different sizes. Moreover, since the process on each level is linear there is no need to update an operator during the course of the operator. This reduces the memory bandwidth for the explicit solver, and additionally allows to yield good results with already few FED cycles per level.

The operators required to create the masks in the analytic or semantic approach are data-parallel and can straightforwardly be transformed into GPU kernels. The only exception from this rule is given by the halftoning step that is required in context of the analytic approach from [BBBW09]. The authors of the original paper suggest to use the accurate but inherently sequential dithering algorithm by Floyd and Steinberg [FS76]. On the GPU, we replace it by the less optimal, but parallel approach by Purgathofer *et al.* [PTG94]. In Chapter 7, we go into detail about the differences between these methods.


**Figure 5.7**: Screen shot of the live demonstrator for 'analytic' image thinning, followed by homogeneous diffusion inpainting. On the *Sintel* movie (CC-BY, Blender Foundation) with a size of 480×360 pixels, about 43 FPS are obtained.

### 5.6.4 Examples

Let us now see some examples of the demonstrator in action. Both the semantic and the analytic approach have been integrated into the same program. It expects the raw data input either from a web cam, or from standard input. By using the realtime video converter `mencoder` as a preprocessing step to the latter option, our application can process any video data including live web streams. Because the transcoding runs on the CPU while the compression framework runs on the GPU, this option is designed to support the main statement of our demonstrator: It *is* possible to compress and decompress video streams of sport events or live footages with PDEs in realtime.

#### Analytic approach

Figure 5.7 shows a screen shot of our demonstrator using the analytic approach. Its window is divided into quadrants. In the top left part, it shows the original input. As an example, we use the *Sintel* movie produced by the Blender Foundation and released under terms of the CC-BY Creative Commons License. The top right frame shows the computed magnitude of the Laplacian, which is then dithered to obtain the mask which is shown in the bottom left quadrant. Note that the regularity artefacts in dark mask regions are due to the parametrisation of the forced random dithering method [PTG94] and can be removed without significant overhead by choosing larger lookup tables. In the bottom right corner, we finally see the inpainted result. With the shown 10% of pixels being preserved, it can hardly be distinguished from the original.

Moreover, note that this application obtains 43 frames per second on an input of size 480×360 pixels, as is indicated in the window title bar. This measurement includes the time required to upload the original and to visualise the solution. This means, there is only a latency of about 23 milliseconds between the deployment of raw data by the CPU and the presentation of the thinned and inpainted results on the screen. On larger frames of size 640×480 pixels, our algorithm still obtains 29 FPS. Hence, even frames in VGA resolution can easily be processed in realtime.

#### Semantic approach

Let us now have a look on our demonstrator running the semantic approach. This is shown in Figure 5.8. Again, the screen is divided into four parts, where the top left again presents the original and the bottom right shows the result. However, the top right now visualises the detected edges. In the bottom left frame, the contours of these edges are coloured with the pixel values at the respective points. This is the image that is inpainted to obtain the result. Again, our algorithm obtains almost 45 FPS on this sequence of size 480×360 pixels including memory transfer and visualisation. Again, this good performance carries over to larger frames. Streams with a spatial resolution of 640×480 pixels are processed with 29 FPS.

### 5.6.5 Efficiency

An interesting question which arises from the specialisation to linear diffusion inpainting is its efficiency compared to other GPU-based algorithms from the literature. While it seems that there are no non-


**Figure 5.8**: Screen shot of the live demonstrator for 'semantic' image thinning, followed by homogeneous diffusion inpainting. On the *Sintel* movie (CC-BY, Blender Foundation) with a size of 480×360 pixels, about 45 FPS are obtained.

linear PDE-based inpainting algorithms on the GPU so far, several frameworks were proposed to solve the Poisson equation on graphics processors [GWL+03, OBW+08, MP08, JCW09]. A fundamental problem for the comparison of these techniques is the lack of implementation details in the original publications, as well as a missing description of the conditions under which the original experiments were conducted. Often, the number of known points in the image or the convergence state of the system are not mentioned, although both affect the runtime.

Hence, the purpose of this brief overview shall be to compare the runtimes given in the original papers to 'equivalent' runs of our novel approach. While performing these comparisons, we must keep in mind that our approach is not specifically optimised to this much more simple problem, but uses the same generic setup as the anisotropic variant. However, the linear case and the fact that we approach the problem in a scale space allow us to use very fast parameter settings.

In order to optimise the parameters, we generate a ground truth for linear diffusion inpainting on the *Trui* image from Figure 5.2. To this end, we use an explicit scheme with 10 million time steps of size $1/8$. We find that by using CFED, already $d = 2$ iterations and a process stopping time $T = 110$ suffice to obtain a solution that differs by less than an MSE of $10^{-1}$ from this ground truth. If we now assume that the point density and distribution almost remains constant within a large range of possible input images, it is justified to use the same parameters for arbitrary images.

Under these conditions, our algorithm requires 8.24 milliseconds to reconstruct an image of size $512 \times 512$ pixels, and 22.43 milliseconds for an image of size $1\,024 \times 1\,024$. Compared to the runtimes of other approaches from the literature, it seems that our algorithm belongs to the fastest available methods.

The approach presented in [GWL+03] sets up on a classical GPGPU approach. Because it was evaluated on hardware that is much slower than modern commodity products, it is hard to perform a fair comparison. In 2003, the authors obtained a runtime of 400 milliseconds on $513 \times 513$ pixels, and 1.45 seconds on images of size $1\,025 \times 1\,025$. Considerations based on the performance gain reported by NVidia [NVi11b, Figure 1-1] suggest that this method would require about 22 and 80 milliseconds on modern hardware, respectively. This makes it about a factor 3 slower than the novel CFED approach.

A more recent approach described in [OBW+08] reports 'realtime performance' on images of size $512 \times 512$ pixels. If we assume that the authors refer to a runtime of 40 milliseconds per frame, it might require about 20–25 milliseconds on a modern graphics card. Again, CFED is 2–3 times faster.

It seems that the performance of CFED is most similar to those techniques proposed in [MP08] and [JCW09]. The first one reports runtimes of about 10 milliseconds 'per cycle' on an megapixel image. It is not said how many cycles are required to let the method sufficiently converge, but a comparison in [JCW09] suggests that about four cycles should be sufficient. Carried over to the performance of modern graphics cards, we obtain about 20–25 milliseconds, which is about the same performance as for CFED. The approach from [JCW09] even decreases this runtime at the cost of a higher approximation error. If an application does not require an optimal convergence state of the underlying method, this approach can even be up to two times more efficient as CFED, while still providing a more accurate solution.

Nevertheless, we can state that CFED is a justified alternative to all other GPU-based image inpainting algorithms from the literature. Besides its good runtime constraints, it also benefits from a clear and intuitive

algorithmic structure. In general, the linear variant of our algorithm seems to scale much better than the anisotropic version, which is both due to a more optimal memory bandwidth and the linear structure of the problem.

## 5.7 Conclusion

In this chapter, we have designed the first GPU-based algorithm for anisotropic PDE-based image inpainting. Our algorithm is based on the cascadic fast explicit diffusion scheme suggested by Grewenig *et al.* [GWB10]. This allows to use the FED-implementation from the last chapter, and to extend it by an efficient texture-based resampling strategy. The resulting algorithm scales very well on GPUs, and obtains up to a factor 48 on the corresponding CPU-based variant. This performance seems to be particularly good for the specialisation to homogeneous diffusion inpainting, for which CFED obtains similar or smaller runtimes than other GPU-based approaches from the literature. By using this homogeneous diffusion approach, images of size $1024 \times 1024$ pixels can in general still be inpainted in real-time. As a result, the thinning and reconstruction of video frames using either the method from [BBBW09] or [MW09] obtain real-time performance on frames with $640 \times 480$ pixels.

The good scaling behaviour despite the high memory throughput is a consequence of the careful algorithmic design. Both the FED scheme from the previous chapter, as well as the novel resampling strategy are tailored to the characteristics of GPUs. Nevertheless, both parts are very easy to implement. While the FED algorithm consists of little more than an explicit solver, restriction and prolongation operators are mapped to primitives of the hardware. This design makes our approach a perfect framework for quick but efficient implementations of arbitrary elliptic algorithms. Moreover, because these algorithms are still transparent to the developer, they are well suited for rapid prototyping.

Despite its simplicity, however, CFED does not seem to be *the optimal* solution for PDE-based image inpainting. This becomes obvious if we compare its runtime to solvers which directly approach the elliptic problem instead of approximating it with large parabolic steps. One prominent example is the CSOR method from [SWB09] which runs 4 times faster than CFED on the CPU, but which cannot be parallelised to the GPU due to its inherently sequential structure. A promising solution to this drawback might be a cascadic application of the Fast Jacobi scheme proposed by Weickert [Wei11b]. It uses Jacobi steps [Mei05] to solve the elliptic problem on each level, but over-relaxes them with a variant series of parameters that

are coupled to the series of FED time steps. This scheme would be fully data-parallel and can be realised on the GPU by using an anisotropic extension of the algorithm from [OBW+08]. However, it remains as future work to show the applicability to PDE-based image inpainting. In this context, it should also be investigated what the optimal relation between the FED time steps and the resulting over-relaxation weights is. In the next chapter, we see an application of a Fast Jacobi scheme to optic flow computations where it helps to speed up this process by more than a factor 2. This raises hopes that a similar strategy will also work for inpainting, and that it even performs significantly better than CFED.

# Chapter 6

# Optic Flow

*Do not go where the path may lead.*
*Go instead where there is no path and leave a trail.*
Ralph Waldo Emerson

## 6.1 Introduction

An important task in computer vision is the accurate computation of motion in a captured scene. If not more than an image sequence of this scene is given, we can still measure the apparent motion of brightness patterns in this sequence. This so-called *optic flow* is frequently used to track objects in a scene [LK81, TK91], or to register different views of the same scene onto each other [LK81, SC94]. Often, these patterns allow for even more exciting applications. Accurately registered frames of an exposure series can be combined to a single image with a higher dynamic range [MP95], to an image with a higher spatial resolution [MPS+09], or both [ZBW11a]. The deduction of motion information reoccurs in related models for stereo vision [ADSW02, BAS10] and scene flow estimation [HD07, VBZ+10].

The resulting quality in most of these applications depends directly on the accuracy of the underlying optic flow estimation. As a consequence, optic flow approaches enjoyed a tremendous increase in quality over the last three decades. This is impressively observable in the *Middlebury* benchmark [BSL+11][1]. Such a progress became possible by recent developments in the field of energy-based methods. Modern approaches still set up on the same energy minimisation ideas as early

[1] Available online at `http://vision.middlebury.edu/flow/eval/`

methods [HS81, NE86, BA96, WS01a]. However, they extend them by a variety of new ingredients which lead to more and more accurate results [BBPW04, ZPB07, SRLB08, WTP+09, ZBW+09, XJM10, ZBW11b].

Energy-based optic flow models find the optic flow as the minimiser of an energy, which typically consists of two different terms: The *data term* penalises deviations from constancy assumptions such as the preservation of intensity values or image gradients over time. This is complemented by the *smoothness term* (or *regulariser*), which enforces the solution to be sufficiently smooth.

The high quality of modern approaches is related to their careful design of these two terms. Most methods weight the local influence of the data term by robust sub-quadratic penaliser functions which reduce the impact of outliers [BA96, BBPW04, ZPB07, ZBW+09]. Moreover, they frequently apply higher-order constancy assumptions such as the constancy of the image gradient to be more robust under varying illumination [TP84, BBPW04, ZBW+09]. An additional normalisation prevents overweighting at image gradients [SAH91, ZBW+09].

Sub-quadratic penalisers also find application in the smoothness term, where they allow to preserve dominant discontinuities in the flow field, while small fluctuations are still reliably smoothed out [BA96, BBPW04, ZPB07]. Anisotropic regularisers further support this behaviour by allowing to smooth parallel to image or flow edges, but preventing smoothing across these edges [NE86, WS01a, SRLB08, WTP+09, ZBW+09]. In [ZBW+09, ZBW11b], this concept is even optimised by designing the smoothness term complementary to the data term. This allows to employ smoothing only in directions where the data term cannot give reliable results.

As a side effect of the growing accuracy of optic flow approaches, the accompanying energy functionals also become more complex. Modern optic flow energies are often highly non-convex and nonlinear, such that their minimisation is a numerically challenging task. Even highly efficient multigrid methods, which are well-known for their good performance on CPUs, cannot achieve even near-realtime performance on typical image sequences. Such methods can also be ported to the GPU, where they obtain real-time performance for relatively basic optic flow models [GT08]. However, their algorithmic structure requires many computations on coarse, i.e. small, representations of the problem. This low degree of parallelism contradicts the massively parallel layout of GPUs, and causes these approaches to waste significant amounts of the available resources. This might be a reason why this option was not pursued by a large community so far.

A class of algorithms which is better suited for GPU-based implementations are primal-dual approaches [ZPB07, WTP+09]. These methods intro-

duce a coupling term which allows to perform separate minimisations with respect to the data term and the smoothness term. While the latter minimisation comes down to a gradient descent algorithm similar to [Cha04], the new energy for the data term can easily be performed by thresholding. This strategy allows to solve modern models very efficiently, even on GPUs. However, despite these promising properties, primal-dual approaches are restricted with respect to the models they can handle. This is because the thresholding technique can only be efficiently implemented for a limited number of data terms, and because the gradient descent algorithm is particularly challenging if the desired regulariser is anisotropic [WTP+09].

On the CPU, the most popular and versatile strategy for minimising continuous energy-based methods is given by the Euler-Lagrange framework [HS81, NE86, WS01b, BBPW04, ZBW+09, ZBW11b]. According to the calculus of variations, the minimiser of such methods is given by a system of coupled partial differential equations which constitute necessary conditions for a minimiser [Els61]. Energy minimisation via the Euler-Lagrange framework has several advantages:

- **Flexibility**. Euler-Lagrange equations can directly be derived from arbitrary models. Even if non-differentiable penalisers are applied, these can easily be handled by introducing small regularisation parameters [Vog95].

- **Generality**. Euler-Lagrange equations are all of diffusion-reaction type. Hence, the same solution strategies work for Euler-Lagrange equations arising for arbitrary energies. The diffusion part can be minimised in a way that is closely related to elliptic diffusion systems such as the one from Chapter 5.

Nevertheless, the solution of Euler-Lagrange equations is often algorithmically demanding. Although bi-directional multigrid methods promise an efficient solution of such equations, they are in general not sufficient to obtain near-realtime performance on modern optic flow models [BW05]. Moreover, this problem seems to be even worse on GPUs, where such schemes are restricted to the solution of fairly basic models [GT08].

In this chapter, we develop a new GPU-based algorithm for variational optic flow based on a simple cascadic implementation of parallel iterative solvers. Our design adopts concepts from the elliptic diffusion scheme discussed in the previous chapter. Besides performing a gradient descent with a cascadic FED solver as in Chapter 5, we also apply the recent Fast Jacobi (FJ) solver [Wei11b]. Our novel parallel algorithm for this solver yields optic flow results in even shorter time.

In order to demonstrate the strength of our algorithms, we apply them to the modern optic flow method of Zimmer *et al.* [ZBW$^+$09] which gives very accurate results. In the same spirit as for the previous chapters, this computationally expensive method also offers the opportunity to design our algorithm very general, such that many other modern techniques can easily derived by an appropriate adaptation. Often, the arising scheme for such techniques is computationally less expensive than the method described here.

Our experiments show speedups of more than 40 over a bi-directional multigrid solver on the CPU. This performance also allows a better quality to runtime ratio than many other methods from the literature. Compared to the anisotropic primal-dual method of Werlberger *et al.* [WTP$^+$09], which is the top-ranking GPU-based anisotropic optic flow method apart from this work, our algorithm obtains better results in an equivalent runtime.

The work described in this chapter was also subject of a publication [GZG$^+$10]. At the time of publication, the algorithm presented in this chapter ranked 6th in the Middlebury benchmark, and obtained these results in the fastest runtime among the ten most accurate methods. Because of the high amount of newly presented techniques in the last months, our algorithm lost several places in the Middlebury benchmark. Today, it claims the 18th rank, but still represents the fastest algorithm among the best 30 techniques.

## 6.2 Variational Optic Flow

We begin with a detailed review of the complementary optic flow model by Zimmer *et al.* [ZBW$^+$09]. A deep understanding of this model allows us not only to find a suitable GPU-based algorithm for this purpose, but also gives many insights about general concepts which are similarly used in other modern optic flow techniques. Because our algorithm is tailored to these concepts rather than the concrete model, it will be easy to adapt it to other — potentially more complex — models later.

Let $\Omega \subset \mathbb{R}$ be a rectangular image domain. An image sequence is then given by $\boldsymbol{f}(\boldsymbol{x}) := (f^1(\boldsymbol{x}), \dots, f^n(\boldsymbol{x}))$, where $\boldsymbol{x} := (x, y, t)^\top \in \Omega \times \mathbb{N}$, and $n \in \mathbb{N}^+$. Each $f^i : \Omega \times \mathbb{N} \to \mathbb{R}$ denotes one channel of the image sequence at location $(x, y) \in \Omega$ and at time $t \geqslant 0$. In this chapter, we specialise on grey images and on RGB-valued colour images, such that $n$ only takes the values 1 and 3, respectively. In both cases, we assume the image sequence to be spatially presmoothed by a Gaussian with standard deviation $\sigma$.

For any given time $t \in \mathbb{N}^+$, we aim at computing the optic flow $\boldsymbol{w} := (u, v, 1)$ as the dense displacement field from $\boldsymbol{f}(x, y, t)$ to $\boldsymbol{f}(x, y, t+1)$. In a variational framework, it is found as the minimiser of an *energy functional* of the form

$$E(u, v) = \int_\Omega \Big( M(u, v) + \alpha\, V(\boldsymbol{\nabla} u, \boldsymbol{\nabla} v) \Big)\, dx\, dy \ . \tag{6.1}$$

In this context, $\boldsymbol{\nabla} := (\partial_x, \partial_y)^\top$ denotes the spatial gradient operator, and $\alpha > 0$ is a smoothness weight. While $M$ denotes the data term which penalises deviations in the consistency between the image sequence and the computed flow, $V$ penalises deviations from the smoothness in the result. The concrete shape of the data term and the smoothness term depends on the underlying optic flow model.

### 6.2.1 Complementary Optic Flow

We instantiate this framework with the recent complementary optic flow (COF) method by Zimmer *et al.* [ZBW+09]. In several perspectives, it represents one of the most accurate and versatile variational optic flow models in the literature. Its data term enforces the constancy of both image grey values and gradients, and weights contributions non-quadratically. Such configuration is frequently applied in the literature and can easily be extended by other constancy assumptions. Moreover, it features an anisotropic smoothness term which is related to the diffusion scheme from Chapter 4, and can thus easily be adapted to simpler isotropic terms where necessary.

The combination of these two powerful ingredients makes COF a very powerful method, as it is witnessed by the Middlebury benchmark. In June 2010 when the work described in this chapter was first published, COF was listed on the fourth place with respect to the endpoint error. In the meantime, it has been passed on to the 18th position, which is an amazing indicator for the rapid progress in this field. We detail more on these facts in Section 6.5.

#### Data Term

For the data term $M$, we use the same model as described in [ZBW+09], but interpret the input as RGB or grey channels instead of converting them

to the HSV colour model from the original work. This leads to

$$M(u,v) := \Psi_M \left( \sum_{i=1}^{n} \theta_0^i \left( f^i(\boldsymbol{x}+\boldsymbol{w}) - f^i(\boldsymbol{x}) \right)^2 \right) \tag{6.2}$$
$$+\gamma \; \Psi_M \left( \sum_{i=1}^{n} \left( \theta_x^i \left( f_x^i(\boldsymbol{x}+\boldsymbol{w}) - f_x^i(\boldsymbol{x}) \right)^2 + \theta_y^i \left( f_y^i(\boldsymbol{x}+\boldsymbol{w}) - f_y^i(\boldsymbol{x}) \right)^2 \right) \right)$$

with subscripts denoting partial derivatives. The weight $\gamma$ steers the influence of the two contributions in the data term:

1. The first line in (6.2) penalises deviations from the constancy of grey values under displacement [HS81]. To this end, it steers the solution towards a state in which

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{w}) \approx \boldsymbol{f}(\boldsymbol{x}) \,. \tag{6.3}$$

   As a measure against an overweighting of the data term at large image gradients, we perform a normalisation in the spirit of [SAH91]. This comes down to the multiplication with a normalisation factor

$$\theta_0^i := \frac{1}{|\boldsymbol{\nabla} f^i|^2 + \zeta^2} \,, \tag{6.4}$$

   where $0 < \zeta < 1$ is a small positive constant to regularise $\theta_0^i$ as $|\boldsymbol{\nabla} f^i|$ approaches 0. Moreover, we robustify the data term against noise and occlusions by weighting its contribution by a sub-quadratic penalisation function

$$\Psi_M(s^2) := \sqrt{s^2 + \varepsilon^2} \,. \tag{6.5}$$

   In this context, $0 < \varepsilon \ll 1$ denotes a small regularisation parameter similar to the one in [BBPW04].

2. The second line in (6.2) models the constancy of the image gradient under displacement [BBPW04]. This robustifies process under additive illumination changes, and yields a solution $\boldsymbol{w}$ which fulfils

$$\boldsymbol{\nabla} \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{w}) \approx \boldsymbol{\nabla} \boldsymbol{f}(\boldsymbol{x}) \,. \tag{6.6}$$

   Again, we normalise this contribution by a factor

$$\theta_{\{x,y\}}^i := \frac{1}{|\boldsymbol{\nabla} f_{\{x,y\}}^i|^2 + \zeta^2} \,, \tag{6.7}$$

and apply the same sub-quadratic penaliser as above. However, note that we perform a separate penalisation of the brightness constancy assumption and the gradient constancy assumption. This follows the suggestion from [BW05], and proves to be more stable under outliers produced by only one of the two constancy assumptions.

### Smoothness Term

Because motion can be ambiguous, the data term is only capable of supplying flow vectors along the direction of incident, the so-called *data constraint direction*. Orthogonal to this direction, no information is provided by the data term. This caveat is also known as the *aperture problem* [Mar82], and is one of the reasons for the ill-posedness of the problem [BPT88]. A popular way to circumvent this problem is the introduction of an additional constraint that requires a global smoothness of the solution [HS81].

Following the arguments from above, it makes sense to design this smoothness term in a way that it *complements* the data term [ZBW+09]. Along the data constraint direction, we want a reduced smoothing behaviour to preserve the information that is given by the data term. Perpendicular to it, however, a strong smoothing allows to fill in missing information from the neighbourhood.

As a first step towards this goal, it is necessary to determine the data constraint direction. We obtain it as the direction of the largest eigenvector of the regularisation tensor

$$\boldsymbol{R}_\rho := \sum_{i=1}^{n} K_\rho * \left( \theta_0^i \boldsymbol{\nabla} f^i \left( \boldsymbol{\nabla} f^i \right)^\top + \gamma \left( \theta_x^i \boldsymbol{\nabla} f_x^i \left( \boldsymbol{\nabla} f_x^i \right)^\top + \theta_y^i \boldsymbol{\nabla} f_y^i \left( \boldsymbol{\nabla} f_y^i \right)^\top \right) \right), \tag{6.8}$$

where $K_\rho * \cdot$ denotes a Gaussian filter as in Definition 3.1 with standard deviation $\rho$. Note that for $\rho = 0$, $\boldsymbol{R}_\rho$ is reduced to a spatial motion tensor as it occurs in linearised data terms. A more detailed explanation of this relation can be found in [ZBW+09] and [ZBW11b].

Let $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ be the larger and smaller orthonormal eigenvectors of $\boldsymbol{R}_\rho$. The complementary smoothness term is then given by

$$V(\boldsymbol{\nabla} u, \boldsymbol{\nabla} v) = \Psi_V \Big( \left( \boldsymbol{r}_1^\top \boldsymbol{\nabla} u \right)^2 + \left( \boldsymbol{r}_1^\top \boldsymbol{\nabla} v \right)^2 \Big) + \left( \boldsymbol{r}_2^\top \boldsymbol{\nabla} u \right)^2 + \left( \boldsymbol{r}_2^\top \boldsymbol{\nabla} v \right)^2 . \tag{6.9}$$

Since $\boldsymbol{r}_1$ is parallel to the data constraint direction, we reduce smoothing in this direction by applying a sub-quadratic Perona-Malik penaliser [PM87, BA96]:

$$\Psi_V(s^2) := \lambda^2 \ln \left( 1 + \frac{s^2}{\lambda^2} \right), \tag{6.10}$$

where $\lambda > 0$ takes the role of a contrast parameter. Along the perpendicular direction of $\boldsymbol{r}_2$, we perform a strong quadratic penalisation to allow missing information to be filled in from the neighbourhood.

### 6.2.2 Energy Minimisation via the Euler-Lagrange Framework

Let us now discuss the minimisation of the energy from (6.1) with the data and smoothness terms from (6.2) and (6.9), respectively. By the calculus of variations, we know that a minimiser $(u, v)$ must necessarily fulfil the Euler-Lagrange equations [Els61]:

$$\partial_u M - \alpha \operatorname{div}\big(\boldsymbol{D}\,(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{\nabla} u, \boldsymbol{\nabla} v)\ \boldsymbol{\nabla} u\big) = 0\,, \tag{6.11}$$

$$\partial_v M - \alpha \operatorname{div}\big(\boldsymbol{D}\,(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{\nabla} u, \boldsymbol{\nabla} v)\ \boldsymbol{\nabla} v\big) = 0\,, \tag{6.12}$$

with reflecting boundary conditions. These equation take the form of a diffusion-reaction system. The minuends $\partial_u M$ and $\partial_v M$ in (6.11) and (6.12) stem from the data term, and play the role of the reaction part. They are complemented by the subtrahends which arise from the smoothness term $V(\boldsymbol{\nabla} u, \boldsymbol{\nabla} v)$. We underline their role as the diffusion part of the system by denoting them in divergence form.

In order to deduce these terms in more detail, let us introduce the following abbreviations. For any $\lozenge \in \{x, y\}$ and $\blacklozenge \in \{x, y, xx, xy, yy\}$, we define derivatives on the image sequence as

$$f^i_{\blacklozenge} := \partial_{\blacklozenge} f^i(\boldsymbol{x}+\boldsymbol{w})\,, \tag{6.13}$$

$$f^i_z := f^i(\boldsymbol{x}+\boldsymbol{w}) - f^i(\boldsymbol{x})\,, \tag{6.14}$$

and

$$f^i_{\lozenge z} := \partial_{\lozenge} f^i(\boldsymbol{x}+\boldsymbol{w}) - \partial_{\lozenge} f^i(\boldsymbol{x})\,. \tag{6.15}$$

Moreover, we refer to $\Psi'_M(s^2)$ as the derivative of $\Psi_M(s^2)$ with respect to its argument $s^2$. For the reaction term, this leads to:

$$\begin{aligned}\partial_u M = \quad & \Psi'_M\left(\sum_{i=1}^n \theta_0^i \big(f_z^i\big)^2\right) \cdot \left(\sum_{i=1}^n \theta_0^i f_z^i\, f_x^i\right) \qquad (6.16)\\ + \ \gamma\ & \Psi'_M\left(\sum_{i=1}^n \Big(\theta_x^i \big(f_{xz}^i\big)^2 + \theta_y^i \big(f_{yz}^i\big)^2\Big)\right) \cdot \left(\sum_{i=1}^n \big(\theta_x^i f_{xz}^i\, f_{xx}^i + \theta_y^i f_{yz}^i\, f_{xy}^i\big)\right),\\ \partial_v M = \quad & \Psi'_M\left(\sum_{i=1}^n \theta_0^i \big(f_z^i\big)^2\right) \cdot \left(\sum_{i=1}^n \theta_0^i\, f_z^i\, f_y^i\right) \qquad (6.17)\end{aligned}$$

$$+\ \gamma\ \Psi'_M\left(\sum_{i=1}^{n}\left(\theta_x^i\left(f_{xz}^i\right)^2+\theta_y^i\left(f_{yz}^i\right)^2\right)\right)\cdot\left(\sum_{i=1}^{n}\left(\theta_x^i f_{xz}^i\, f_{xy}^i+\theta_y^i f_{yz}^i\, f_{yy}^i\right)\right)\,.$$

By using the same notation for the diffusion term, we obtain for the diffusion tensor $\boldsymbol{D}$:

$$\boldsymbol{D}\left(\boldsymbol{r}_1,\boldsymbol{r}_2,\boldsymbol{\nabla}u,\boldsymbol{\nabla}v\right):=\Psi'_V\Big(\left(\boldsymbol{r}_1^\top\boldsymbol{\nabla}u\right)^2+\left(\boldsymbol{r}_1^\top\boldsymbol{\nabla}v\right)^2\Big)\,\boldsymbol{r}_1\boldsymbol{r}_1^\top+\boldsymbol{r}_2\boldsymbol{r}_2^\top\,. \tag{6.18}$$

By construction, this diffusion tensor acts complementary to the data term. However, we also see that it is both image and flow driven. Its direction stems from the regularisation tensor in (6.8), which adapts to image structures. In contrast, its magnitude depends on the spatial gradients of the flow $\boldsymbol{\nabla}u$ and $\boldsymbol{\nabla}v$. A combination of these two components allows to obtain the sharp flow edges such as in image-driven methods, but reduces artefacts arising from over-segmentation [SRLB08].

### 6.2.3 Warping

As it turns out, the Euler-Lagrange equations from (6.11) and (6.12) are very challenging to solve. This is because the unknown flow $\boldsymbol{w}$ appears implicitly in the derivatives $f^i(\boldsymbol{x}+\boldsymbol{w})$ of the image sequence. In the literature, problems of this kind are commonly solved by embedding the solution into a coarse-to-fine multi-scale warping strategy [BBPW04]. In such an approach, the input sequence is gradually modified by the computed flow, until these increments vanish. The total correction performed throughout this process is the sought optic flow. Moreover, the application of this strategy on multiple scales enables the process to handle large displacements.

As a first step in this process, we require the input images to be restricted to a coarser scale. On this level $k$, we then split the flow field into the partition $\boldsymbol{w}^k=(u^k,v^k,1)^\top$ which we obtained from the next coarser level, and into the increment $d\boldsymbol{w}^k=(du^k,dv^k,0)^\top$ that is being computed on the current level. Together, these two parts yield the flow that is passed on to the next finer level:

$$\boldsymbol{w}^k+d\boldsymbol{w}^k=:\boldsymbol{w}^{k+1} \tag{6.19}$$

The increment $d\boldsymbol{w}^k$ is computed by a linearised approach [BBPW04]. As a first step to derive this approach, we perform a Taylor linearisation

$$f_z^{i,k+1}:=f^i(\boldsymbol{x}+\boldsymbol{w}^{k+1})-f^i(\boldsymbol{x})\ \approx\ f_x^{i,k}du^k+f_y^{i,k}dv^k+f_z^{i,k}\,, \tag{6.20}$$

where the superscripts $k$ in the linearisation indicate that the arguments are all taken from level $k$.

If we replace each occurrence of $f_z^{i,k+1}$ in (6.11) by the linearisation from (6.20), we obtain the *linearised Euler-Lagrange equations*:

$$\begin{aligned}
0 = \quad & \Psi'_M\left(\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^k+f_y^{i,k}dv^k+f_z^{i,k}\right)^2\right)\\
& \cdot\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^k+f_y^{i,k}dv^k+f_z^{i,k}\right)f_x^{i,k}\\
& +\gamma\cdot\Psi'_M\Bigg(\sum_{i=1}^{n}\Big(\quad\theta_x^{i,k}\left(f_{xx}^{i,k}du^k+f_{xy}^{i,k}dv^k+f_{xz}^{i,k}\right)^2\\
& \qquad\qquad+\theta_y^{i,k}\left(f_{xy}^{i,k}du^k+f_{yy}^{i,k}dv^k+f_{yz}^{i,k}\right)^2\Big)\Bigg)\\
& \cdot\sum_{i=1}^{n}\Big(\quad\theta_x^{i,k}\left(f_{xx}^{i,k}du^k+f_{xy}^{i,k}dv^k+f_{xz}^{i,k}\right)f_{xx}^{i,k}\\
& \qquad+\theta_y^{i,k}\left(f_{xy}^{i,k}du^k+f_{yy}^{i,k}dv^k+f_{yz}^{i,k}\right)f_{xy}^{i,k}\Big)\\
& -\alpha\,\mathrm{div}\left(\boldsymbol{D}\left(\boldsymbol{r}_1^k,\boldsymbol{r}_2^k,\boldsymbol{\nabla}\left(u^k+du^k\right),\boldsymbol{\nabla}\left(v^k+dv^k\right)\right)\ \boldsymbol{\nabla}\left(u^k+du^k\right)\right), \qquad (6.21)
\end{aligned}$$

and

$$\begin{aligned}
0 = \quad & \Psi'_M\left(\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^k+f_y^{i,k}dv^k+f_z^{i,k}\right)^2\right)\\
& \cdot\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^k+f_y^{i,k}dv^k+f_z^{i,k}\right)f_y^{i,k}\\
& +\gamma\cdot\Psi'_M\Bigg(\sum_{i=1}^{n}\Big(\quad\theta_x^{i,k}\left(f_{xx}^{i,k}du^k+f_{xy}^{i,k}dv^k+f_{xz}^{i,k}\right)^2\\
& \qquad\qquad+\theta_y^{i,k}\left(f_{xy}^{i,k}du^k+f_{yy}^{i,k}dv^k+f_{yz}^{i,k}\right)^2\Big)\Bigg)\\
& \cdot\sum_{i=1}^{n}\Big(\quad\theta_x^{i,k}\left(f_{xx}^{i,k}du^k+f_{xy}^{i,k}dv^k+f_{xz}^{i,k}\right)f_{xy}^{i,k}\\
& \qquad+\theta_y^{i,k}\left(f_{xy}^{i,k}du^k+f_{yy}^{i,k}dv^k+f_{yz}^{i,k}\right)f_{yy}^{i,k}\Big)\\
& -\alpha\,\mathrm{div}\left(\boldsymbol{D}\left(\boldsymbol{r}_1^k,\boldsymbol{r}_2^k,\boldsymbol{\nabla}\left(u^k+du^k\right),\boldsymbol{\nabla}\left(v^k+dv^k\right)\right)\ \boldsymbol{\nabla}\left(v^k+dv^k\right)\right). \qquad (6.22)
\end{aligned}$$

## 6.3 Numerical Solution

Let us now discuss the numerical solution of equation (6.21) and the corresponding equation for $dv$. In this context, we derive and evaluate two different approaches. On the following pages, we first adapt the FED scheme from Section 4.3 to this problem. Secondly, we use the recent Fast Jacobi technique by Weickert [Wei11b] as an alternative to this scheme.

### 6.3.1 Fast Explicit Diffusion

Equations such as (6.21) have similar numerical properties as the ones appearing in context of image inpainting: They are of elliptic kind. If we want to solve them by a gradient descent, this means that we must solve a diffusion-like process with a stopping time that approaches infinity. With the same arguments as before, FED seems to be a very good technique to perform this task. It offers large time steps at a low computational cost, and can easily be parallelised.

As a first step in the development of an FED update equation for complementary optic flow, we perform a gradient descent on (6.21) and discretise the arising equation in time. This leads to the following *stabilised* explicit scheme [WS01b]:

$$\begin{aligned}
\frac{du^{k,l+1}-du^{k,l}}{\tau} = \operatorname{div}\Big(D\left(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k, \boldsymbol{\nabla}\left(u^k+du^{k,l}\right), \boldsymbol{\nabla}\left(v^k+dv^{k,l}\right)\right) \ \boldsymbol{\nabla}\left(u^k+du^{k,l}\right)\Big) \\
- \frac{1}{\alpha}\Bigg( \quad \Psi'_M\Bigg(\sum_{i=1}^{n} \theta_0^{i,k}\left(f_x^{i,k}du^{k,l}+f_y^{i,k}dv^{k,l}+f_z^{i,k}\right)^2\Bigg) \\
\cdot \sum_{i=1}^{n} \theta_0^{i,k}\left(f_x^{i,k}du^{k,l+1}+f_y^{i,k}dv^{k,l}+f_z^{i,k}\right) f_x^{i,k} \\
+ \gamma \cdot \Psi'_M\Bigg(\sum_{i=1}^{n}\Bigg( \quad \theta_x^{i,k}\left(f_{xx}^{i,k}du^{k,l}+f_{xy}^{i,k}dv^{k,l}+f_{xz}^{i,k}\right)^2 \\
+ \theta_y^{i,k}\left(f_{xy}^{i,k}du^{k,l}+f_{yy}^{i,k}dv^{k,l}+f_{yz}^{i,k}\right)^2\Bigg)\Bigg) \\
\cdot \sum_{i=1}^{n}\Bigg( \quad \theta_x^{i,k}\left(f_{xx}^{i,k}du^{k,l+1}+f_{xy}^{i,k}dv^{k,l}+f_{xz}^{i,k}\right) f_{xx}^{i,k} \\
+ \theta_y^{i,k}\left(f_{xy}^{i,k}du^{k,l+1}+f_{yy}^{i,k}dv^{k,l}+f_{yz}^{i,k}\right) f_{xy}^{i,k}\Bigg)\Bigg)\Bigg],
\end{aligned} \tag{6.23}$$

and

$$\begin{aligned}\frac{dv^{k,l+1}-dv^{k,l}}{\tau} = \operatorname{div}\Big(D\left(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k, \boldsymbol{\nabla}\left(u^k+du^{k,l}\right), \boldsymbol{\nabla}\left(v^k+dv^{k,l}\right)\right) \ \boldsymbol{\nabla}\left(v^k+dv^{k,l}\right)\Big) \\ - \frac{1}{\alpha}\Bigg( \quad \Psi'_M\Bigg(\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^{k,l}+f_y^{i,k}dv^{k,l}+f_z^{i,k}\right)^2\Bigg) \\ \cdot\sum_{i=1}^{n}\theta_0^{i,k}\left(f_x^{i,k}du^{k,l}+f_y^{i,k}dv^{k,l+1}+f_z^{i,k}\right)f_y^{i,k} \\ + \gamma\cdot\Psi'_M\Bigg(\sum_{i=1}^{n}\Bigg(\quad \theta_x^{i,k}\left(f_{xx}^{i,k}du^{k,l}+f_{xy}^{i,k}dv^{k,l}+f_{xz}^{i,k}\right)^2 \\ + \theta_y^{i,k}\left(f_{xy}^{i,k}du^{k,l}+f_{yy}^{i,k}dv^{k,l}+f_{yz}^{i,k}\right)^2\Bigg)\Bigg) \\ \cdot\sum_{i=1}^{n}\Bigg(\quad \theta_x^{i,k}\left(f_{xx}^{i,k}du^{k,l}+f_{xy}^{i,k}dv^{k,l+1}+f_{xz}^{i,k}\right)f_{xy}^{i,k} \\ + \theta_y^{i,k}\left(f_{xy}^{i,k}du^{k,l}+f_{yy}^{i,k}dv^{k,l+1}+f_{yz}^{i,k}\right)f_{yy}^{i,k}\Bigg)\Bigg)\Bigg],\end{aligned} \tag{6.24}$$

The new variable $l$ in these equations refers to the iteration number within the explicit scheme, and $\tau$ denotes a small time step with. Note that the values for $du^k$ and $dv^k$ in the reaction term are already taken from the new time step $l+1$ to improve the stability.

It remains to discretise the process spatially on a grid with spacings $h_x$ and $h_y$, and to embed it into an FED context. While the input images and output flow fields can simply be sampled at this grid, we obtain derivatives and the regularisation tensor $\boldsymbol{R}_\rho$ by second-order finite difference discretisations [Smi04, ZBW+09]. In order to ease notation, we denote the discrete representations of all involved variables with the same name in bold face letters. In this context, we also keep the notation $\boldsymbol{du}$ and $\boldsymbol{dv}$, where each construct refers to a single vector, and interpret products between vectors component-wise unless stated differently. Moreover, we denote the discretisation of $\operatorname{div}(D(..)\boldsymbol{\nabla}(u^k{+}du^{k,l}))$ and $\operatorname{div}(D(..)\boldsymbol{\nabla}(v^k{+}dv^{k,l}))$ by matrix-vector products:

$$\boldsymbol{A}(\boldsymbol{u}^k+\boldsymbol{du}^{k,l}, \boldsymbol{v}^k+\boldsymbol{dv}^{k,l})(\boldsymbol{u}^k+\boldsymbol{du}^{k,l}) =: \boldsymbol{A}^{k+1,l}\boldsymbol{u}^{k+1,l}\,, \tag{6.25}$$

$$\boldsymbol{A}(\boldsymbol{u}^k+\boldsymbol{du}^{k,l}, \boldsymbol{v}^k+\boldsymbol{dv}^{k,l})(\boldsymbol{v}^k+\boldsymbol{dv}^{k,l}) =: \boldsymbol{A}^{k+1,l}\boldsymbol{v}^{k+1,l}\,. \tag{6.26}$$

With this notation, we finally obtain the (stabilised) FED update equations:

$$\begin{aligned}
\boldsymbol{du}^{k,l+1} = \Bigg[ & \boldsymbol{du}^{k,l} + \tau_l \boldsymbol{A}^{k+1,l} \boldsymbol{u}^{k+1,l} \\
& - \frac{\tau_l}{\alpha} \Bigg( \Psi'^{\,k,l}_{M,*}\ (...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_y^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_z^{i,k} \right) \boldsymbol{f}_x^{i,k} \\
& \quad + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{xz}^{i,k} \right) \boldsymbol{f}_{xx}^{i,k} \\
& \quad + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{yy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{yz}^{i,k} \right) \boldsymbol{f}_{xy}^{i,k} \Bigg) \Bigg] \\
& \cdot \Bigg[ 1 + \frac{\tau_l}{\alpha} \cdot \Psi'^{\,k,l}_{M,*}\ (...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_x^{i,k} \right)^2 \\
& \quad + \frac{\tau_l}{\alpha} \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \left( \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xx}^{i,k} \right)^2 + \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \right)^2 \right) \Bigg]^{-1}, \quad (6.27)
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{dv}^{k,l+1} = \Bigg[ & \boldsymbol{dv}^{k,l} + \tau_l \boldsymbol{A}^{k+1,l} \boldsymbol{v}^{k+1,l} \\
& - \frac{\tau_l}{\alpha} \Bigg( \Psi'^{\,k,l}_{M,*}\ (...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_x^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_z^{i,k} \right) \boldsymbol{f}_y^{i,k} \\
& \quad + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xx}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{xz}^{i,k} \right) \boldsymbol{f}_{xy}^{i,k} \\
& \quad + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{yz}^{i,k} \right) \boldsymbol{f}_{yy}^{i,k} \Bigg) \Bigg] \\
& \cdot \Bigg[ 1 + \frac{\tau_l}{\alpha} \cdot \Psi'^{\,k,l}_{M,*}\ (...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_y^{i,k} \right)^2 \\
& \quad + \frac{\tau_l}{\alpha} \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \left( \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \right)^2 + \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{yy}^{i,k} \right)^2 \right) \Bigg]^{-1}, \quad (6.28)
\end{aligned}$$

where $\Psi'^{\,k,l}_{M,*}(...)$ and $\Psi'^{\,k,l}_{M,**}(...)$ refer to the terms $\Psi'$ from (6.23)–(6.24) with discretised arguments. Furthermore, note that the previously constant time step $\tau$ is replaced by a step-variant reordered FED time step $\tau_l$ as in (4.15).

Doing so requires to keep the operator $\boldsymbol{A}^{k+1,l}$ constant over the period of one FED cycle, which is conceptually very similar to time-lagged diffusivity schemes. The extension to several FED cycles follows straightforwardly by re-using a series of FED time steps multiple times.

### 6.3.2 Fast Jacobi

A second possibility to solve linearised Euler-Lagrange equations such as (6.21) is to apply an iterative solver directly to this equation. A promising candidate for this purpose which can again easily be parallelised for GPUs is the recently developed *Fast Jacobi* (FJ) technique [Wei11b]. It is based on a classical relaxed Jacobi solver [Mei05], but uses special iteration-varying over-relaxation parameters $\omega_l$ to accelerate the process. We discuss their selection at the end of this section.

We begin with a spatial discretisation of the process as in the previous section, end derive a standard relaxed Jacobi scheme. To this end, let $\boldsymbol{P}^{k+1,l}$ be the matrix that contains the main diagonal from $\boldsymbol{A}^{k+1,l}$:

$$\boldsymbol{P}_{i,j}^{k+1,l} := \begin{cases} \boldsymbol{A}_{i,j}^{k+1,l}, & i = j \\ 0, & \text{else} \end{cases} . \tag{6.29}$$

By calling this matrix $\boldsymbol{P}$ instead of using the canonical name $\boldsymbol{D}$, we avoid confusion with the diffusion tensor from above. With the same notation as for FED, we obtain one Jacobi relaxation step by the equations:

$$\begin{aligned} \boldsymbol{du}^{k,l+1} = \boldsymbol{du}^{k,l} & \\ + \omega_l \Bigg[ & \Psi'^{\,k,l}_{M,*}\,(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_x^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_y^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_z^{i,k} \right) \boldsymbol{f}_x^{i,k} \\ & + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xx}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{xy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{xz}^{i,k} \right) \boldsymbol{f}_{xx}^{i,k} \\ & + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{yy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{yz}^{i,k} \right) \boldsymbol{f}_{xy}^{i,k} \\ & - \alpha \boldsymbol{A}^{k+1,l} \boldsymbol{u}^{k+1,l} \Bigg] \\ \cdot \Bigg[ & \boldsymbol{P}^{k+1,l} + \Psi'^{\,k,l}_{M,*}\,(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_x^{i,k} \right)^2 \\ & + \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \left( \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xx}^{i,k} \right)^2 + \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \right)^2 \right) \Bigg]^{-1} , \end{aligned} \tag{6.30}$$

and

$$\begin{aligned}
\boldsymbol{dv}^{k,l+1} = \boldsymbol{dv}^{k,l} & \\
+ \omega_l \Bigg[ & \Psi'^{\,k,l}_{M,*}\,(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_x^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_y^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_z^{i,k} \right) \boldsymbol{f}_y^{i,k} \\
& + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xx}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{xy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{xz}^{i,k} \right) \boldsymbol{f}_{xy}^{i,k} \\
& + \gamma \cdot \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \boldsymbol{du}^{k,l} + \boldsymbol{f}_{yy}^{i,k} \boldsymbol{dv}^{k,l} + \boldsymbol{f}_{yz}^{i,k} \right) \boldsymbol{f}_{yy}^{i,k} \\
& - \alpha \boldsymbol{A}^{k+1,l} \boldsymbol{u}^{k+1,l} \Bigg] \\
\cdot \Bigg[ & \boldsymbol{P}^{k+1,l} + \Psi'^{\,k,l}_{M,*}\,(...) \cdot \sum_{i=1}^{n} \boldsymbol{\theta}_0^{i,k} \left( \boldsymbol{f}_y^{i,k} \right)^2 \\
& + \Psi'^{\,k,l}_{M,**}(...) \cdot \sum_{i=1}^{n} \left( \boldsymbol{\theta}_x^{i,k} \left( \boldsymbol{f}_{xy}^{i,k} \right)^2 + \boldsymbol{\theta}_y^{i,k} \left( \boldsymbol{f}_{yy}^{i,k} \right)^2 \right) \Bigg]^{-1} .
\end{aligned} \tag{6.31}$$

Note the occurrence of the reaction terms in the inverses. It stems from a similar stabilisation as for the stabilised explicit scheme, in which $du$ for the data term is taken already from the new time step $k+1$.

In this context, $\omega_l$ is a relaxation parameter. Setting $\omega_l = 1$ for all $l$ yields a standard Jacobi method. Besides, $\omega_l$ can also take different values to improve its convergence speed or error dampening behaviours [Mei05]. In these cases, however, $\omega_l$ is bounded by a small constant to guarantee the stability of the arising system.

The idea of the *Fast Jacobi* technique is to relate $\omega_l$ to a series of reordered FED time steps such as the ones from (4.15) [Wei11b]. This concept is motivated by the elliptic homogeneous diffusion case in which a Jacobi step turns out to be equivalent to an explicit step with limiting time step size. Hence, it is likely that the same concepts which preserve the stability over large time steps also work in context of the Jacobi solver, and that these concepts carry over to the non-linear case.

However, it seems that the optimal relation between the series of FED time steps $(\tau_l)_{0<l\leqslant l_{\max}}$ and the series of over-relaxation parameters $(\omega_l)_{0<l\leqslant l_{\max}}$ is non-trivial in general. Although it would stand to reason that these variables should be in the linear dependency

$$\omega_l = c \cdot \tau_l \,, \tag{6.32}$$

approaches to determine the linear weight $c$ by stability considerations yielded overcautious configurations. While these estimations of a small upper bound for $c$ guarantee the process to be stable, they cause a much longer runtime than necessary.

Experiments suggest that it is possible to chose $c$ much larger than these estimates. On all tested inputs, complementary optic flow remains stable up to a bound of about 3.4. Because the stability of the process depends on numerical inaccuracies of the architecture which change with the input, however, it is advisable to be slightly more conservative. In the following, we use the configuration $c = 3$, which proved to create stable results in all experiments performed so far.

As a final remark, note that a similar over-relaxation does not work in the case of FED. There, even small increases of the time step size suffice to let the process diverge.

### 6.3.3 Cascadic Application

Independent of whether we decide to use the FED or the FJ scheme, in both cases the runtime of the process can significantly be reduced if we embed this scheme into a coarse-to-fine hierarchy. In the previous chapter, we have already discussed such a hierarchic process in the context of PDE-based image inpainting. Because a coarser scale can be sampled with larger grid spacings, information can travel faster over the image domain. This comes only at a low cost, because computations on a more compact representation of the problem require less time. Similar to image inpainting, we choose the smallest restriction factors $\nu = {}^{h}/_{H}$ with $\nu \geqslant 0.5$ in both dimensions for which the coarser image can again be represented on a uniform grid. In this context, we use $h$ and $H$ to refer to the spatial sampling rate on the fine and on the large grid, respectively.

This hierarchic algorithm for the cascadic FED or FJ solvers must not be confused with the hierarchic warping strategy, which is interwoven with this technique. For warping, we also restrict data to coarser grids, but the resampling factors $\eta \in [0.5, 1)$ used in this case are in general chosen much larger than for the cascadic solver. A larger $\eta$ causes the hierarchy to be *steeper*, which means that many computations are performed on relatively similar grids. While such choice allows to find a good minimum for the given energy, it also takes considerably longer to compute.

Figure 6.1 shows this double-hierarchic algorithm consisting of non-linear solver steps, denoted by $\mathsf{S}$, and warping steps, visualised by $\mathsf{W}$. Depending on the chosen numerical scheme, the role of $\mathsf{S}$ is either played by FED or by FJ. Arrows represent the data flow between either of these steps.

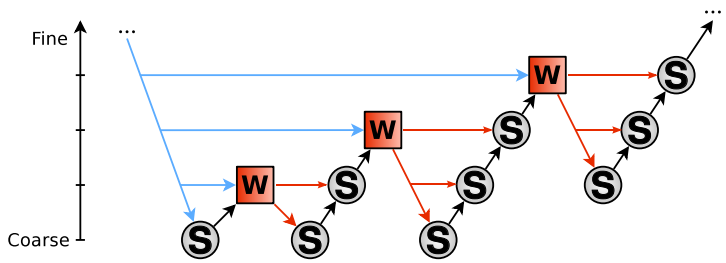



**Figure 6.1**: Visualisation of the double hierarchic algorithm for optic flow, consisting of solver (S) and warping (W) steps. Data flow is drawn in blue to denote original images, in red to represent image pairs of which one partner is warped, and in black to identify solutions.

Where they are running in the vertical direction, a resampling is required. In this context, blue represents a pair of original images which is restricted to all warping levels. At either level, it is received by the warping routine and compensated by the motion from the previous stage. This modified image pair consisting of one original and one warped image is then passed on to the hierarchic solver for this stage, which is shown in red. The intermediate solution is then gradually improved, and is used for the next warping level until this process finds the sought result on the finest level.

## 6.4 Implementation on the GPU

The complementary optic flow model as it is described above is mathematically complex and algorithmically challenging. However, thanks to our preparatory work from the previous chapters we can easily construct an efficient GPU-based algorithm. Most operators have already been optimised in context of the previous PDE-based image processing tasks. Hence, we only discuss the main similarities and differences for those kernels which already appeared in other contexts. This list is extended by a brief description for the new CUDA kernels for Fast Jacobi, warping, and the weights from the data term.

- The **FED** and **FJ kernels** have the same memory layout and interface, and are structurally similar to the FED solver from Chapter 4. Again, they are highly memory bound, and reduce their memory bandwidth by using textures, and by exploiting symmetries in the stencil weights.

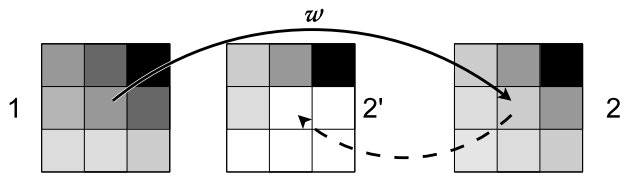



**Figure 6.2**: 'Backwards' warping as performed in [BBPW04, ZBW+09].

- Among these weights, those stemming from the **smoothness term** are computed by 3 subsequent CUDA kernels which compute the derivatives, the diffusion tensor, and finally the arising weights. Because all of these kernels involve the convolution of a signal with a stencil, collapsing them into a single kernel would increase (instead of decrease) their memory bandwidth, such that it seems this step cannot be optimised further.

- Weights arising from the **data term** are spatially localised, and can thus be computed by a single data-parallel kernel. All memory operations arising in this context are efficiently solved by linear loads and stores from and to global device memory.

- **Restriction and prolongation** operators are similar to those from Chapter 5, but do not perform the normalised convolution step from (5.6). Because our algorithm is tailored to both grey-valued and RGB-valued image sequences, we supply different kernels for these two cases which may be called at request. This allows to mask additional memory latencies in the case of RGB-valued image sequences.

- For the **Gaussian filter** required to presmooth the input and the regularisation tensor, we again apply our efficient algorithm from Chapter 3 with a truncation $c\sigma = 3\sigma$. It includes a cache-based convolution with a Gaussian kernel for small standard deviations, and a recursive filter for larger standard deviations.

- As it turns out, the **warping** step required on each level is not more challenging, either. Different to the work in [RFSB10], we do not warp images along the direction of the flow, but against it. This 'backwards' warping step is much simpler and computationally inexpensive than forward warping. As it is shown in Figure 6.2, we evaluate expressions of type $\boldsymbol{f}^i(\boldsymbol{x}+\boldsymbol{w}^k)$ by using the texturing unit of graphics cards. To this end, we simply store the image channel $i$ that is to be warped in a texture, compute the target location by adding flow field and pixel

coordinates, and fetch the texture at the respective point. Albeit incoherent memory access is often considered a major performance problem on massively parallel hardware, this operation turns out to be highly efficient: Optic flow is often piecewise laminar and sufficiently smooth, such that the missing data locality is largely compensated by the 2-D texture cache.

## 6.5 Experiments

Let us now evaluate the quality and the runtime of our new algorithm with some experiments. Besides general impressions for these properties, we are in particular interested in a direct comparison between FED and FJ, and in the benefit over CPU-based methods from the literature.

### 6.5.1 Quality

#### Visual Comparison and Quantitative Evaluation

As a first experiment, we compare results produced by our approach visually, and compare them with respect to their approximation error. In this context, we pick four sequences *Dimetrodon*, *Grove2*, *RubberWhale*, and *Urban2* from the Middlebury optic flow database [2]. Since we are interested in the optimal quality that our algorithm can produce, we tune its parameters individually for each sequence. Besides the generic model parameters $\sigma = 0.3$ and $\rho = 1.3$ which yield optimal results in all cases, we further use

- $\alpha = \phantom{1\,}400$, $\gamma = \phantom{1\,}8$, $\zeta = 1.0$, and $\lambda = 0.05$ for *Dimetrodon*,
- $\alpha = \phantom{1\,}50$, $\gamma = \phantom{1\,}1$, $\zeta = 1.0$, and $\lambda = 0.05$ for *Grove2*,
- $\alpha = 1\,000$, $\gamma = 20$, $\zeta = 1.0$, and $\lambda = 0.05$ for *RubberWhale*,
- $\alpha = 1\,500$, $\gamma = 25$, $\zeta = 0.01$, and $\lambda = 0.1$ for *Urban2*.

The arising energy functional is numerically minimised on a warping hierarchy with $\eta = 0.91$, with 1 cascadic solver cycle and 1 linear update per warping level. As an interesting observation, we find that FED obtains the minimum for a stopping time $T = 150$, while FJ already converges to a very similar solution if the stopping time of the generating FED process is chosen as $T = 40$. As a consequence, the runtimes for the FJ algorithm are always smaller than those for the FED algorithm. We discuss this observation in more detail later in this chapter.

[2] Available at `http://vision.middlebury.edu/flow/data/`

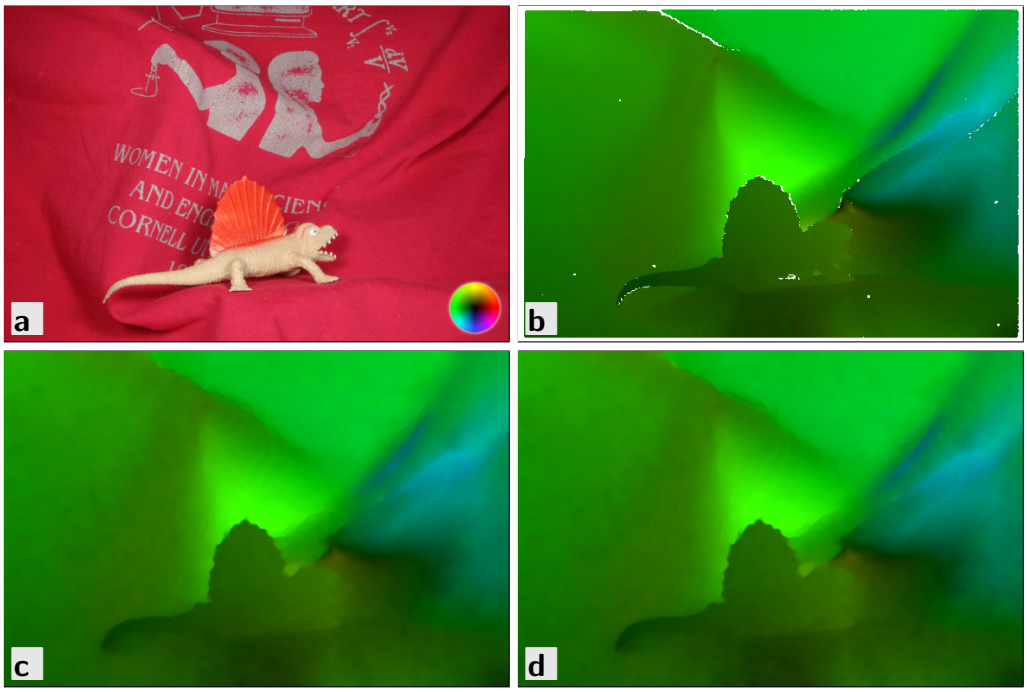



**Figure 6.3**: **a.** *Dimetrodon* from the Middlebury dataset, 584×388 pixels, with flow key. **b.** Ground truth. **c.** FED result, AAE = 1.49°, AEE = 0.08, Runtime: 226 ms. **d.** FJ result, AAE = 1.49°, AEE = 0.08, Runtime: 153 ms.

Figures 6.3–6.6 show the results of this experiment. Part a. of all figures shows the reference frame of the underlying sequence. In their bottom right corners, these figures include the colour visualisation key for the arising flow fields. Part b. shows the provided ground truth coded in this colour scheme, where unknown partitions are denoted with white pixels. Parts c. and d. finally depict the solutions of our FED and FJ algorithms on the GPU, respectively.

In all cases, the solutions yielded by the different numerics cannot be distinguished from each other. The similarity of the results obtained by the cascadic FED and FJ solvers is also reflected in the approximation errors of both solutions to the ground truth. These are defined as follows. Let a flow field $\boldsymbol{w} := (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)^\top$ with $\boldsymbol{w}_i := (\boldsymbol{u}_i, \boldsymbol{v}_i, 1)^\top$ be given. We distinguish the *Average Angular Error (AAE)* [FJ90, BFB94]:

$$\mathrm{AAE}(\boldsymbol{w}_1, \boldsymbol{w}_2) = \sum_{i=1}^{n} \arccos \left\langle \frac{\boldsymbol{w}_{1,i}}{\|\boldsymbol{w}_{1,i}\|}, \frac{\boldsymbol{w}_{2,i}}{\|\boldsymbol{w}_{2,i}\|} \right\rangle, \tag{6.33}$$

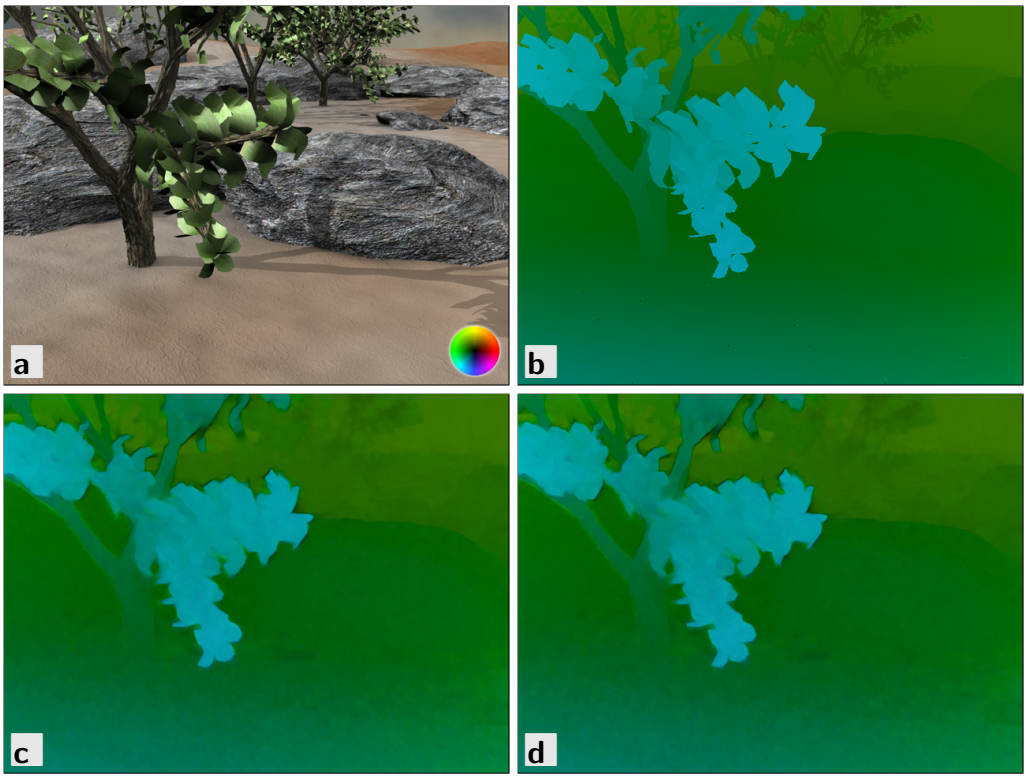



**Figure 6.4**: **a.** *Grove2* from the Middlebury dataset, 640×480 pixels, with flow key. **b.** Ground truth. **c.** FED result, AAE = 2.32°, AEE = 0.16. Runtime: 366 ms. **d.** FJ result, AAE = 2.32°, AEE = 0.16, Runtime: 246 ms.

where $\langle\cdot\rangle$ denotes the inner product, and the *Average Endpoint Error (AEE)* [ON95]:

$$\mathrm{AEE}(\boldsymbol{w}_1, \boldsymbol{w}_2) = \sum_{i=1}^{n} \sqrt{(\boldsymbol{u}_{1,i} - \boldsymbol{u}_{2,i})^2 + (\boldsymbol{v}_{1,i} - \boldsymbol{v}_{2,i})^2}\,. \tag{6.34}$$

Table 6.1 summarises the approximation errors for the shown examples, and compares them to the corresponding errors obtained by the original method from [ZBW+09]. With respect to both error measures, the results obtained by FED and FJ are almost identical. This is remarkable if we consider that both approaches use a numerical parameter set that is optimised for a low runtime, but FJ only requires about 2/3 of the time FED consumes.

Moreover, the errors deviate only marginally from those obtained by the original method, although the implementation of the algorithms differs

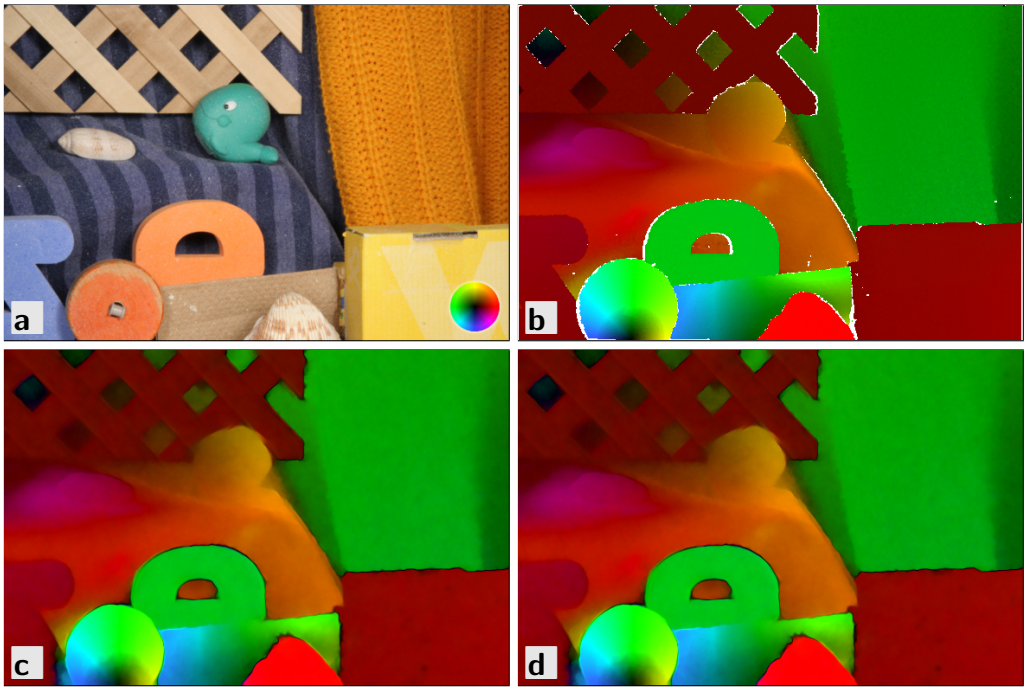



**Figure 6.5**: **a.** *RubberWhale* from the Middlebury dataset, 584×388 pixels, with flow key. **b.** Ground truth. **c.** FED result, AAE = 2.93°, AEE = 0.09. Runtime: 322 ms. **d.** FJ result, AAE = 2.90°, AEE = 0.09, Runtime: 219 ms.

**Table 6.1**: Comparison of FED and FJ against the original method (FAS) from [ZBW+09], using 4 Middlebury sequences with known ground truth and the optimal parameter sets from Section 6.5.1.

| | **FAS (CPU)** | | **FED (GPU)** | | **FJ (GPU)** | |
|---|---|---|---|---|---|---|
| | **AEE** | **AAE** | **AEE** | **AAE** | **AEE** | **AAE** |
| **Dimetrodon** | 0.08 | 1.48 | 0.08 | 1.49 | 0.08 | 1.49 |
| **Grove2** | 0.16 | 2.31 | 0.16 | 2.32 | 0.16 | 2.32 |
| **RubberWhale** | 0.09 | 2.82 | 0.09 | 2.93 | 0.09 | 2.90 |
| **Urban2** | 0.31 | 2.77 | 0.29 | 2.75 | 0.29 | 2.75 |

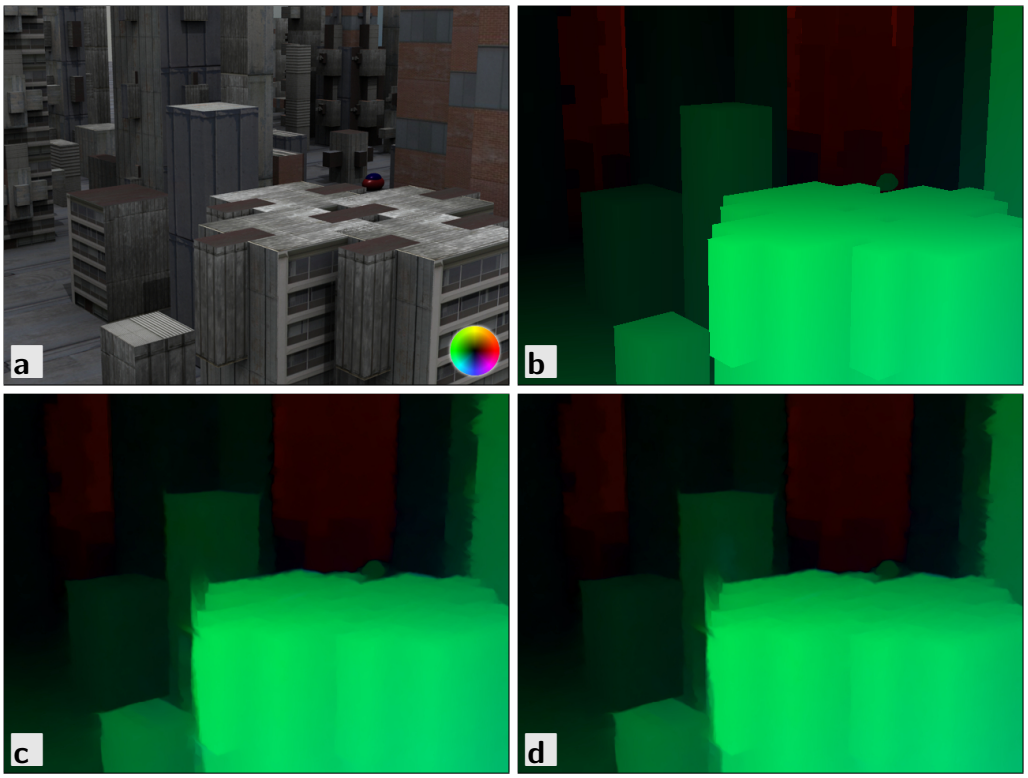



**Figure 6.6**: **a.** *Urban2* from the Middlebury dataset, 640×480 pixels, with flow key. **b.** Ground truth. **c.** FED result, AAE = 2.75°, AEE = 0.29. Runtime: 725 ms. **d.** FJ result, AAE = 2.75°, AEE = 0.29, Runtime: 577 ms.

in significant parts: Besides different numerics and new prolongation and restriction schemes, our method particularly uses runtime-optimised numerical settings, such as the decision to call only one FED or FJ cycle per level. The latter simplification seems to benefit from the warping hierarchy, in which several FED or FJ cycles are subsequently applied over a series of images sequences in very similar scales.

### Optimised vs. Generic Parameters

We would like to confirm these observations on the quality by more quantitative experiments. A good resource for a comparison of our algorithm with other methods in this field is the Middlebury Optical Flow Benchmark [BSL+11]. However, the submission rules to this benchmark demand results to be computed with a fixed set of parameters. As a first step towards this goal, we thus search a generic parameter set for the sequences

**Table 6.2**: Error measures for 4 Middlebury sequences with known ground truth using the optimal parameter sets from Section 6.5.1, and the fixed parameter set $\alpha = 300$, $\gamma = 20$, $\zeta = 0.01$, and $\lambda = 0.1$. Results are obtained with FED using $T = 150$, and with FJ using relaxation parameters according to $T = 40$.

| | **Optimised** | | | | **Fixed** | | | |
|---|---|---|---|---|---|---|---|---|
| | **FED** | | **FJ** | | **FED** | | **FJ** | |
| | **AEE** | **AAE** | **AEE** | **AAE** | **AEE** | **AAE** | **AEE** | **AAE** |
| **Dimetrodon** | 0.08 | 1.49 | 0.08 | 1.49 | 0.11 | 2.20 | 0.11 | 2.20 |
| **Grove2** | 0.16 | 2.32 | 0.16 | 2.32 | 0.19 | 2.69 | 0.19 | 2.69 |
| **RubberWhale** | 0.09 | 2.93 | 0.09 | 2.90 | 0.11 | 3.76 | 0.11 | 3.75 |
| **Urban2** | 0.29 | 2.75 | 0.29 | 2.75 | 0.36 | 3.53 | 0.36 | 3.54 |

from above. The results obtained with this new parametrisation are then compared to those from above, where we again apply the AAE and the AEE as measures for the dissimilarity.

As optimised parameter sets, we again use the same settings as for the examples from Figures 6.3–6.6. To those, we compare the parameter set $\alpha = 300$, $\gamma = 20$, $\zeta = 0.01$, and $\lambda = 0.1$, which was optimised using all Middlebury sequences with known ground truth. This procedure gives confidence that a similar configuration might also give good results on the actual test suite with hidden ground truth. Again, we run our FED algorithm with $T = 150$, and use a relaxation parameters for our FJ that are based on the time steps of an FED process for $T = 40$.

Table 6.2 shows this comparison. The generic setting performs worse than the optimised configuration, but our algorithms still yield very accurate results. Their quality is comparable to those reported by Werlberger *et al.* [WTP+09] which is the top-ranked GPU-based method in the Middlebury benchmark if we exclude our own work. However, our algorithm can again slightly be improved over this method if we also optimise the parameters such as done in the previous experiment. Moreover, we see that our two numerical schemes perform equally well, which indicates that both schemes seem to be sufficiently converged.

### Middlebury Benchmark

In context of the original publication of our algorithm using the FED solver [GZG+10], results were submitted to the Middlebury benchmark site to compare its performance to other methods in the field. Besides the ap-

| | Average endpoint error | avg. rank | Army (Hidden texture) GT all | Army (Hidden texture) GT disc | Army (Hidden texture) GT untext | Army (Hidden texture) im0 all | Army (Hidden texture) im0 disc | Army (Hidden texture) im0 untext | Army (Hidden texture) im1 all | Army (Hidden texture) im1 disc | Army (Hidden texture) im1 untext | Mequon (Hidden texture) GT all | Mequon (Hidden texture) GT disc | Mequon (Hidden texture) GT untext | Mequon (Hidden texture) im0 all | Mequon (Hidden texture) im0 disc | Mequon (Hidden texture) im0 untext | Mequon (Hidden texture) im1 all | Mequon (Hidden texture) im1 disc | Mequon (Hidden texture) im1 untext | Schefflera (Hidden texture) GT all | Schefflera (Hidden texture) GT disc | Schefflera (Hidden texture) GT untext | Schefflera (Hidden texture) im0 all | Schefflera (Hidden texture) im0 disc | Schefflera (Hidden texture) im0 untext | Schefflera (Hidden texture) im1 all | Schefflera (Hidden texture) im1 disc | Schefflera (Hidden texture) im1 untext | Teddy (Stereo) GT all | Teddy (Stereo) GT disc | Teddy (Stereo) GT untext | Teddy (Stereo) im0 all | Teddy (Stereo) im0 disc | Teddy (Stereo) im0 untext | Teddy (Stereo) im1 all | Teddy (Stereo) im1 disc | Teddy (Stereo) im1 untext | Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Classic+NL [31] | 4.8 | **0.08** 1 | **0.23** 1 | 0.07 2 | 0.22 7 | 0.74 7 | 0.18 9 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 972 |
| 2 | MDP-Flow [26] | 5.5 | 0.09 2 | 0.25 2 | 0.08 5 | 0.19 2 | **0.54** 1 | 0.18 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 188 |
| 3 | NL-TV-NCC [25] | 6.3 | 0.10 6 | 0.26 5 | 0.08 5 | 0.22 7 | 0.72 5 | 0.15 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 20 |
| 4 | Complementary OF [21] | 7.6 | 0.10 6 | 0.26 5 | 0.09 8 | 0.21 6 | 0.73 6 | 0.15 3 | 0.34 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 600 |
| 5 | Adaptive [20] | 8.6 | 0.09 2 | 0.26 5 | **0.06** 1 | 0.23 10 | 0.78 9 | 0.18 9 | 0.54 17 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 9.2 |
| 6 | ComplOF-FED-GPU [36] | 10.3 | 0.11 10 | 0.27 9 | 0.09 8 | 0.20 4 | 0.77 8 | 0.14 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0.96 |
| 10 | Aniso. Huber-L1 [22] | 11.0 | 0.10 6 | 0.28 10 | 0.08 5 | 0.31 19 | 0.88 16 | 0.28 22 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | TV-L1-improved [17] | 13.2 | 0.09 2 | 0.26 5 | 0.07 2 | 0.20 4 | 0.71 4 | 0.16 5 | 0.53 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2.9 |
| 15 | Rannacher [23] | 15.1 | 0.11 10 | 0.31 13 | 0.09 8 | 0.25 12 | 0.84 15 | 0.21 15 | 0.57 21 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | F-TV-L1 [15] | 15.2 | 0.14 23 | 0.35 19 | 0.14 25 | 0.34 22 | 0.98 22 | 0.26 20 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 8 |
| 25 | Bregman-TV [35] | 23.3 | 0.12 18 | 0.36 23 | 0.12 23 | 0.31 19 | 0.97 21 | 0.26 20 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2.9 |
| 35 | FOLKI [16] | 34.4 | 0.29 34 | 0.73 36 | 0.33 34 | 1.52 35 | 1.96 36 | 1.80 35 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1.4 |

**Figure 6.7**: Excerpt of the Middlebury ranking at 2010-06-21, sorted by the AEE. Annotated are the rank (left) and the runtime in seconds for one computation on 640×480 pixels (right).

proximation quality in which we are interested at first place, we also use the opportunity to compare the runtimes of the benchmarked methods against our approach.

An excerpt of the original snapshot of the Middlebury ranking at the time of submission is shown in Figure 6.7. Listed are all methods which perform better than our method, as well as all methods in the ranking which process an image pair of size 640×480 pixels in less than 10 seconds. Note that these times are supplied by the authors, and are thus not normalised to a specific architecture or processor clock. The given 0.96 seconds for our algorithm were computed with FED on an NVidia GeForce GTX 285.

Both in this line-up which is based on the AEE, as well as in the list that is sorted by the AAE, our algorithm ranked sixth among 36 methods. Moreover, it represented the fastest technique in the top 10. All other methods among the 10 most accurate were 10 times or more slower.

The much different quality of complementary optic flow on the CPU and on the GPU relates to a misleading entry for the CPU method: The benchmarked state of the method by Zimmer *et al.* [ZBW+09] was already an improved version which finally led to the method from [ZBW11b]. Today, the Middlebury benchmark lists this improved version as *OFH*, and correctly assigns the label *Complementary OF* to the original approach from [ZBW+09].

| | Average endpoint error | avg. rank | Army (Hidden texture) GT all | Army (Hidden texture) GT disc | Army (Hidden texture) GT untext | Army (Hidden texture) im0 all | Army (Hidden texture) im0 disc | Army (Hidden texture) im0 untext | Army (Hidden texture) im1 all | Army (Hidden texture) im1 disc | Army (Hidden texture) im1 untext | Mequon (Hidden texture) GT all | Mequon (Hidden texture) GT disc | Mequon (Hidden texture) GT untext | Mequon (Hidden texture) im0 all | Mequon (Hidden texture) im0 disc | Mequon (Hidden texture) im0 untext | Mequon (Hidden texture) im1 all | Mequon (Hidden texture) im1 disc | Mequon (Hidden texture) im1 untext | S (Hidden texture) | eddy (Stereo) GT all | eddy (Stereo) GT disc | eddy (Stereo) GT untext | eddy (Stereo) im0 all | eddy (Stereo) im0 disc | eddy (Stereo) im0 untext | eddy (Stereo) im1 all | eddy (Stereo) im1 disc | eddy (Stereo) im1 untext | Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MDP-Flow 2 [40] | 5.2 | 0.09 6 | 0.23 3 | 0.07 3 | 0.16 2 | **0.52** 1 | 0.13 3 | 0 | | | | | | | | | | | | | | 1.11 9 | 0.72 7 | | | | | | | 420 |
| 2 | Layers++ [38] | 6.1 | 0.08 2 | **0.21** 1 | 0.07 3 | 0.19 5 | 0.56 3 | 0.17 13 | | | | | | | | | | | | | | | **0.88** 1 | 0.72 7 | | | | | | | 18206 |
| 3 | TC-Flow [48] | 9.0 | **0.07** 1 | **0.21** 1 | **0.06** 1 | **0.15** 1 | 0.59 4 | **0.11** 1 | 0.3 | | | | | | | | | | | | | | 1.35 14 | 0.93 25 | | | | | | | 2500 |
| 4 | LSM [41] | 9.0 | 0.08 2 | 0.23 3 | 0.07 3 | 0.22 13 | 0.73 15 | 0.18 15 | 0.28 8 | | | | | | | | | | | | | | 99 3 | 0.73 10 | | | | | | | 1615 |
| 5 | Classic+NL [31] | 9.9 | 0.08 2 | 0.23 3 | 0.07 3 | 0.22 13 | 0.74 16 | 0.18 15 | 0.29 10 | | | | | | | | | | | | | | .98 2 | 0.74 12 | | | | | | | 972 |
| 6 | IROF-TV [57] | 10.8 | 0.09 6 | 0.25 10 | 0.08 10 | 0.22 13 | 0.77 17 | 0.19 19 | | | | | | | | | | | | | | | 1.08 7 | 0.73 10 | | | | | | | 261 |
| 7 | MDP-Flow [26] | 12.1 | 0.09 6 | 0.25 10 | 0.08 10 | 0.19 5 | 0.54 2 | 0.18 | | | | | | | | | | | | | | | 1.68 32 | 0.97 28 | | | | | | | 188 |
| 8 | OFH [39] | 13.2 | 0.10 14 | 0.25 10 | 0.09 19 | 0.19 5 | 0.69 11 | 0.14 5 | 0. | | | | | | | | | | | | | | 1.40 17 | 0.74 12 | | | | | | | 620 |
| 9 | OF-Mol [49] | 13.2 | 0.08 2 | 0.23 3 | 0.07 3 | 0.28 29 | 0.99 33 | 0.20 24 | 0.28 | | | | | | | | | | | | | | 1.08 7 | 0.78 14 | | | | | | | 1160 |
| 10 | NL-TV-NCC [25] | 13.7 | 0.10 14 | 0.26 15 | 0.08 10 | 0.22 13 | 0.72 14 | 0.15 7 | 0.35 16 | | | | | | | | | | | | | | 11 | 0.55 2 | | | | | | | 20 |
| 11 | Sparse Occlusion [58] | 13.8 | 0.09 6 | 0.24 7 | 0.08 10 | 0.22 13 | 0.63 7 | 0.19 19 | 0.38 | | | | | | | | | | | | | | 1.13 10 | 0.67 5 | | | | | | | 2312 |
| 12 | SimpleFlow [52] | 14.5 | 0.09 6 | 0.24 7 | 0.08 10 | 0.24 22 | 0.78 18 | 0.20 24 | | | | | | | | | | | | | | | 1.04 6 | 0.72 7 | | | | | | | 1.7 |
| 13 | CostFilter [42] | 14.8 | 0.10 14 | 0.27 18 | 0.08 10 | 0.20 10 | 0.63 7 | 0.15 7 | | | | | | | | | | | | | | | 1.02 4 | 0.62 3 | | | | | | | 90 |
| 14 | Adaptive [20] | 17.2 | 0.09 6 | 0.26 15 | **0.06** 1 | 0.23 20 | 0.78 18 | 0.18 15 | | | | | | | | | | | | | | | 1.37 16 | 0.79 15 | | | | | | | 9.2 |
| 15 | DPOF [18] | 18.9 | 0.12 30 | 0.33 30 | 0.08 10 | 0.26 25 | 0.80 22 | 0.20 24 | | | | | | | | | | | | | | | 1.02 4 | **0.54** 1 | | | | | | | 287 |
| 16 | Adapt-Window [34] | 19.1 | 0.10 14 | 0.24 7 | 0.09 19 | 0.19 5 | 0.59 4 | 0.15 7 | 0.27 6 | | | | | | | | | | | | | | 0.88 21 | | | | | | | | 1935 |
| 17 | Complementary OF [21] | 20.2 | 0.11 20 | 0.28 19 | 0.10 28 | 0.18 3 | 0.63 7 | 0.12 2 | 0.3 | | | | | | | | | | | | | | 1.48 19 | 0.95 26 | | | | | | | 44 |
| 18 | ComplOF-FED-GPU [36] | 20.2 | 0.11 20 | 0.29 22 | 0.10 28 | 0.21 12 | 0.78 18 | 0.14 5 | | | | | | | | | | | | | | 4 15 | 1.50 21 | 0.83 17 | | | | | | | 0.73 |
| 21 | Aniso. Huber-L1 [22] | 21.0 | 0.10 14 | 0.28 19 | 0.08 10 | 0.31 32 | 0.88 27 | 0.28 35 | 0.5 | | | | | | | | | | | | | | 1.36 15 | 0.79 15 | | | | | | | 2 |
| 23 | TV-L1-improved [17] | 24.8 | 0.09 6 | 0.26 15 | 0.07 3 | 0.20 10 | 0.71 13 | 0.16 10 | 0.53 27 | | | | | | | | | | | | | | 2 27 | 0.87 20 | | | | | | | 2.9 |
| 27 | F-TV-L1 [15] | 27.0 | 0.14 37 | 0.35 33 | 0.14 41 | 0.34 36 | 0.98 32 | 0.26 34 | 0.59 | | | | | | | | | | | | | | 1.56 23 | 0.66 4 | | | | | | | 8 |
| 33 | Bartels [43] | 30.8 | 0.12 30 | 0.30 25 | 0.11 34 | 0.22 13 | 0.65 10 | 0.19 19 | 0.35 16 | | | | | | | | | | | | | | .08 36 | 1.22 37 | | | | | | | 0.15 |
| 38 | Shiralkar [44] | 35.5 | 0.13 36 | 0.39 40 | 0.10 28 | 0.28 29 | 1.08 37 | 0.1 | | | | | | | | | | | | | | 1.04 38 | 2.13 47 | 1.10 32 | | | | | | | 600 |
| 48 | HBpMotionGpu [45] | 41.0 | 0.17 41 | 0.41 45 | 0.13 39 | 0.61 46 | 1.34 48 | 0.59 47 | 0. | | | | | | | | | | | | | | 43 2.04 40 | 1.67 45 | | | | | | | 1000 |
| 57 | FOLKI [16] | 55.4 | 0.29 53 | 0.73 57 | 0.33 53 | 1.52 57 | 1.96 57 | 1.80 57 | 1.23 54 | 2. | | | | | | | | | | | | | 57 | 4.32 57 | | | | | | | 1.4 |

**Figure 6.8**: Excerpt of the Middlebury ranking at 2011-09-17, sorted by the AEE. Annotated are the rank (left) and the runtime in seconds for one computation on 640×480 pixels (right).

If we compare these measurements to a recent snapshot of the same ranking as shown in Figure 6.8, we observe the impressive progress in the area of optic flow among the last few months. Within 15 months, the list grew from 36 entries to 58. Some of the new techniques are more accurate than our algorithm, such that its position dropped to the 18th rank in the AEE-based line-up, and even to place 19 in the AAE-based ranking.

Most of these methods are clearly optimised to a good approximation quality rather than a low runtime, which is reflected in extraordinarily high runtimes of several minutes to hours. This is also visible for the methods ranking on 38th and 48th place. They are also included in our excerpt because they set up on a GPU-based algorithm, although their runtime lies in the order of several minutes. However, there are also new interesting competitors which possess similar runtimes as our approach: The anonymous method on rank 12 requires about twice the time of our approach, but is also more accurate.

Nevertheless, complementary optic flow on the GPU still represents the fastest method among the top 30 of the most accurate methods. This shows that even today, our algorithm offers a very good trade-off between a high quality on the one hand, and a very low runtime on the other hand.

Note that the timing for our approach has additionally been replaced by a new benchmark obtained on the more recent NVidia GeForce GTX 480, and can even be improved if we go from the FED numerics to the new FJ solver. Section 6.5.2 goes into more detail about the runtime of our algorithms.

### Convergence: FED vs. FJ

In this last quality-related experiment, we are interested in the convergence behaviour of our two parallel solvers. From the first experiment, we already know that FJ seems to converge with much less iterations per level than FED. To prove this hypothesis, we choose different numbers of iterations for FJ and FED, and execute the arising algorithm with the generic parameter set on the *RubberWhale* sequence from Figure 6.5.

Since complementary optic flow is a highly non-convex model, the valid determination of a convergence state is not trivial. Even if the algorithm is fully converged on the finest level, a numerical inaccuracy on a coarser level could steer the process into different minima with a similar energy. Hence, we choose the official ground truth as a neutral basis for comparison, and relate the difference between a given solution and this ground truth to the convergence state of the minimisation process. As a measure for dissimilarity, we use the AAE.

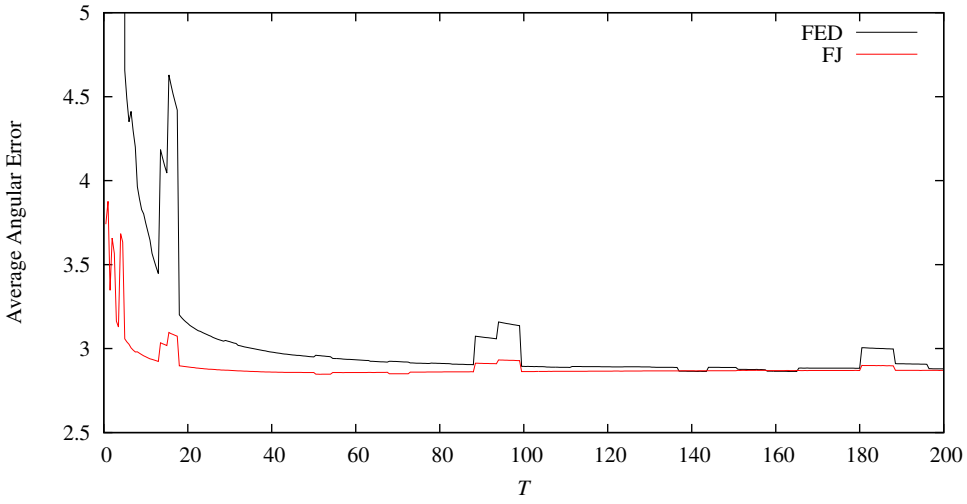



**Figure 6.9**: Convergence comparison for FED and FJ on *RubberWhale* with optimal parameters, normalised to the step generating stopping time $T$ per cycle.

By (4.13), the number of iterations is related to the stopping time of the FED process. A similar relation can be established for the FJ solver, although the notion of a stopping time for an elliptic process is formally wrong. Instead, we interpret $T$ in this context as the stopping time of the *step size generating* FED process, and thus as an intuitive continuous analogon to the discrete number of iterations.

Figure 6.9 shows the result of this experiment. It confirms our previous observation that FJ converges faster than FED: While FED requires a stopping time of about $T \approx 150$ per level to reach a state in which the solution approaches a constant error level, a similar state is already reached for $T \approx 40$ if we use FJ. This corresponds to about 30 iterations for FED, but only 15 iterations for FJ, and explains the fundamentally different runtime observed in both cases.

Besides this insight, we also see that the error does not decay by a monotonous function as we increase $T$. This is because the aforementioned non-convexity of the problem: Even small numerical errors or dissipative effects during the minimisation can steer the process to a different solution. This seems to occur for certain numbers of iterations, where we obtain piecewise linear segments in the graph. However, again the FJ scheme appears to be more robust than FED by reducing these effects. This dampening behaviour could be related to the fact that we couple $\omega_l$ by a factor of $c = 3$ to $\tau_l$, although larger factors are experimentally possible. It is likely that this effective under-relaxation further stabilises the process.

### 6.5.2 Runtime

#### Scaling on Image Size

Let us now evaluate the runtime and scaling behaviour of our algorithm on image sequences in different sizes. Because the Middlebury benchmark largely features images in the size ratio 4:3 and our algorithm is optimised to this measure, we use the same size ratio in this experiment. Following the insights obtained in the previous sections, we use the generic parameter set, and use a stopping time $T = 150$ for FED. In context of FJ, time steps are generated based on an FED process with $T = 40$.

Besides the pure time required for the computations, we also measure the time required to transfer the problem and the solution from CPU to GPU RAM and vice versa. For optic flow, this measure is interesting for two reasons: While the previous chapters were primarily concerned with concise sub-problems of complex visual computing applications, optic flow can already be seen as one of such self-contained frameworks. Moreover, the bandwidth between CPU and GPU is a very important factor for optic flow computations. For RGB-valued image pairs in single precision floating point arithmetics, 24 bytes per pixel must be uploaded to the device, and a solution with 8 bytes per pixel must be retrieved once the computations are finished.

The outcome of this experiment is shown in Figure 6.10. For small images, the GPU is not fully occupied, and the overhead required for expensive operations such as the management of the hierarchy require additional runtime. Both effects cause the problem of sub-optimal scaling under these conditions. However, the full potential of the graphics device is clearly unveiled once the size of one image exceeds about 1 megapixels. Then, the process scales almost linear in the number of pixels.

This leads to remarkable runtimes when the image sequence is relatively large. RGB-valued image sequences with 3.5 megapixels per image can be processed in less than 2 seconds, and the optic flow in typical screen resolutions such as $1\,280\times960$ pixels is computed in about 1 second or less.

In a direct comparison of the computations on grey-valued and RGB-valued image sequences, we see that the latter setting can only process images up to a size of about 3.5 megapixels, while the grey-valued case is even able to yield results for images that are larger than 5.0 megapixels. This limitation is only due to the restricted GPU RAM, and will more and more vanish with the development of new graphics cards with larger main memory. Nevertheless, we also see that the runtimes do not differ much between these two cases. This is because the number of input channels

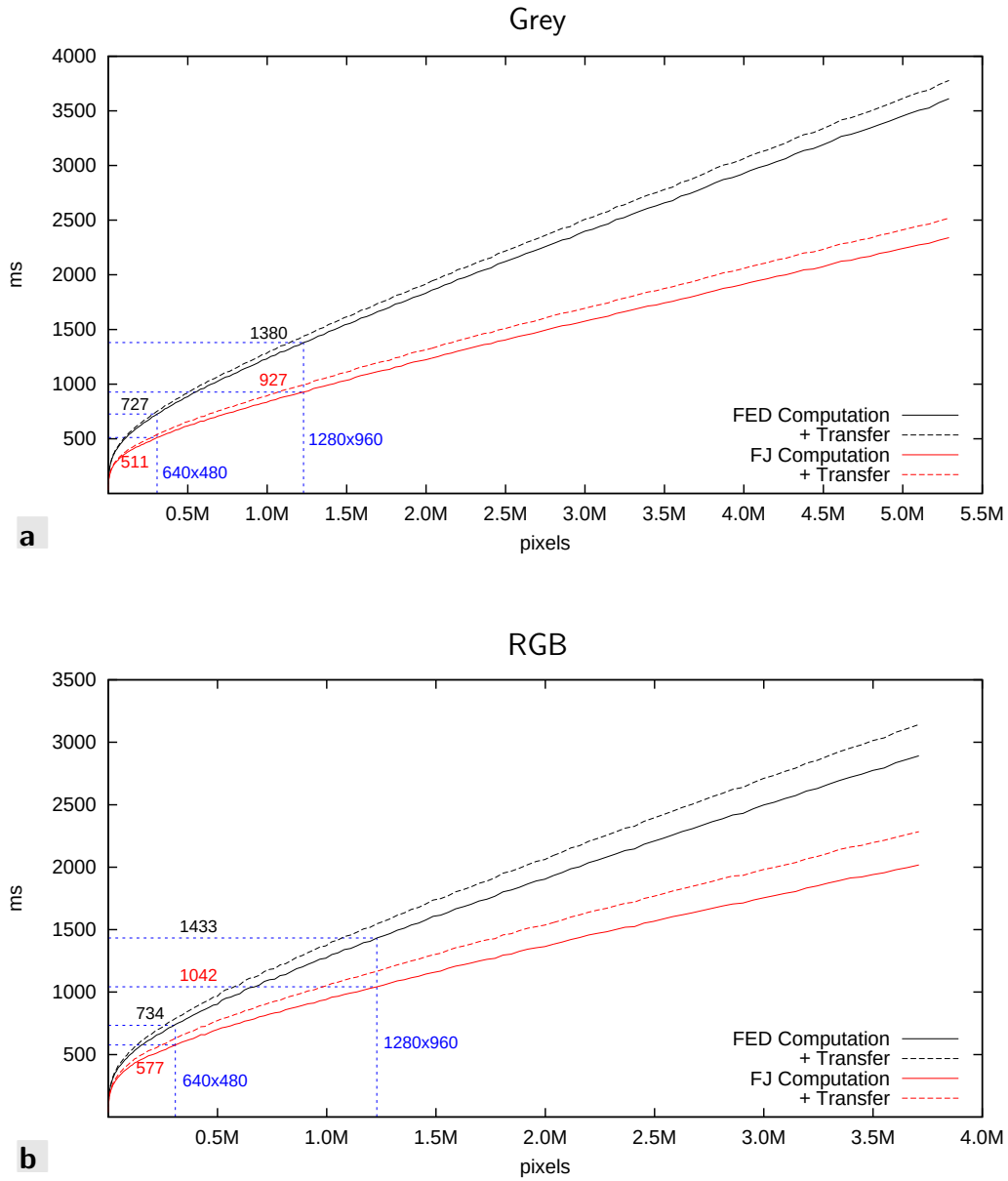



**Figure 6.10**: Runtime of GPU-based complementary optic flow for **a.** grey-valued input images, and **b.** RGB-valued input images, both in the size ratio 4:3. We use the generic parameter set from Section 6.5.1, choose $T = 150$ for FED, and generate FJ time steps based on $T = 40$.

**Table 6.3**: Runtimes in milliseconds for GPU-based FED and FJ, for their CPU-based variants, and the original FAS technique used in [ZBW+09]. Speedups refer to the parallelisation gain (FED, FJ), or to CPU FAS vs. GPU FJ (Total).

| | | **320 × 240** | | **640 × 480** | | **1 280 × 960** | |
|---|---|---|---|---|---|---|---|
| | | **Grey** | **RGB** | **Grey** | **RGB** | **Grey** | **RGB** |
| **CPU** | **FAS** | 2 537 | 3 007 | 9 630 | 11 575 | 38 658 | 46 681 |
| | **FED** | 3 475 | 3 792 | 15 054 | 16 336 | 63 516 | 68 812 |
| | **FJ** | 2 639 | 2 948 | 11 535 | 12 786 | 48 191 | 53 538 |
| **GPU** | **FED** | 453 | 460 | 727 | 734 | 1 380 | 1 433 |
| | **FJ** | 328 | 374 | 511 | 577 | 927 | 1 042 |
| **Speedup** | **FED** | 7.67 | 8.25 | 20.71 | 22.27 | 46.02 | 48.03 |
| | **FJ** | 8.05 | 7.89 | 22.55 | 22.15 | 52.01 | 51.36 |
| | **Total** | **7.74** | **8.05** | **18.83** | **20.05** | **41.72** | **44.78** |

only has an influence on the setup phase of the diffusion tensor, but not on the actual solver iterations. Since the latter consume a vast majority of the runtime, the overheads for more than one image channel do not deteriorate the overall runtime significantly.

If we are interested to use our algorithm within a CPU-based vision application, the memory upload and download costs are another important quantity. As it turns out, they are not negligible for optic flow. While they take up to about 10% of the runtime on computations on grey-valued image sequences, this partition even grows to about 15% for RGB-valued sequences.

### Comparison to CPU

Let us now have a more detailed look on the runtime of our process, and compare it against the runtime of the respective variants on the CPU, as well as against the original non-linear multigrid (FAS) method from [ZBW+09]. In this experiment, we assume data to reside already in the memory of the target architecture. To this end, we abstract from the times required to upload or download the problem.

Table 6.3 lists the runtimes in milliseconds for all analysed algorithms, as well as the speedups obtained by the GPU-based algorithm[3]. On the CPU,

[3] My thanks go to Henning Zimmer and Andrés Bruhn for providing a sample implementation of the FAS scheme from [ZBW+09]

both FED and FJ run slightly slower than the original FAS approach. This matches our expectation, since FAS offloads even more computational effort to hierarchic representations. Such coarse-grid computations are very efficient on sequential architectures such as CPUs, while they are very expensive on massively parallel architectures due to their low hardware occupancy. Nevertheless, FJ seems to be a serious competitor even on this architecture, since it performs only slightly worse than FAS.

Our GPU algorithm clearly profits from the high data-parallelism of FED and FJ, and obtains in both cases speedups from about 8 on image sequences of size 320×240 to about 50 for 1 280×960. From the almost linear behaviour observed in the last experiment, we know that this speedup factor will also roughly hold for larger inputs. This speedup is even higher than for inpainting, which is related to the different numerical structure, but also to a slightly less computationally demanding discretisation scheme for the derivatives.

Finally, we compare FAS as the fastest method on the CPU to FJ as the most efficient method on the GPU. This is indicated by the last row in Table 6.3. Again, we obtain speedups of about 8 for image sequences of size 320×240 pixels, up to more than 40 for frames of size 1 280×960 pixels. This is due to the high performance and data-parallelism of FJ, and substantiate our hypothesis that this scheme might be one of the best parallel solvers for elliptic problems in visual computing. Future work should be concerned with a detailed evaluation of this scheme, also with respect to other elliptic problems such as PDE-based image inpainting (see Chapter 5).

### Profiling

In a last short experiment, we analyse the time consumption of the different parts of our algorithm in more detail. Sample outputs of the CUDA profiler `cudaprof` for FED and FJ are shown in Figure 6.11. They were obtained on a run on the *Urban2* sequence from Figure 6.6 using the generic parameter set. Note that the overall runtime does not fully sum up to the overall runtime from Figure 6.10. This is because the profiling plot does not include CPU-side calls to GPU kernels and the latencies involved therein, but only covers the plain runtime of the kernels.

To ease presentation, the 22 involved kernels (including internals such as for memory copy operations) are clustered to 10 meaningful groups, each. For both numerical schemes, the solver kernel consumes the most significant partition of the runtime. In this contribution, we also spot the main differences between the runtimes of FED and FJ, which is related to the different numbers of iterations needed in either case. The second most expensive op-

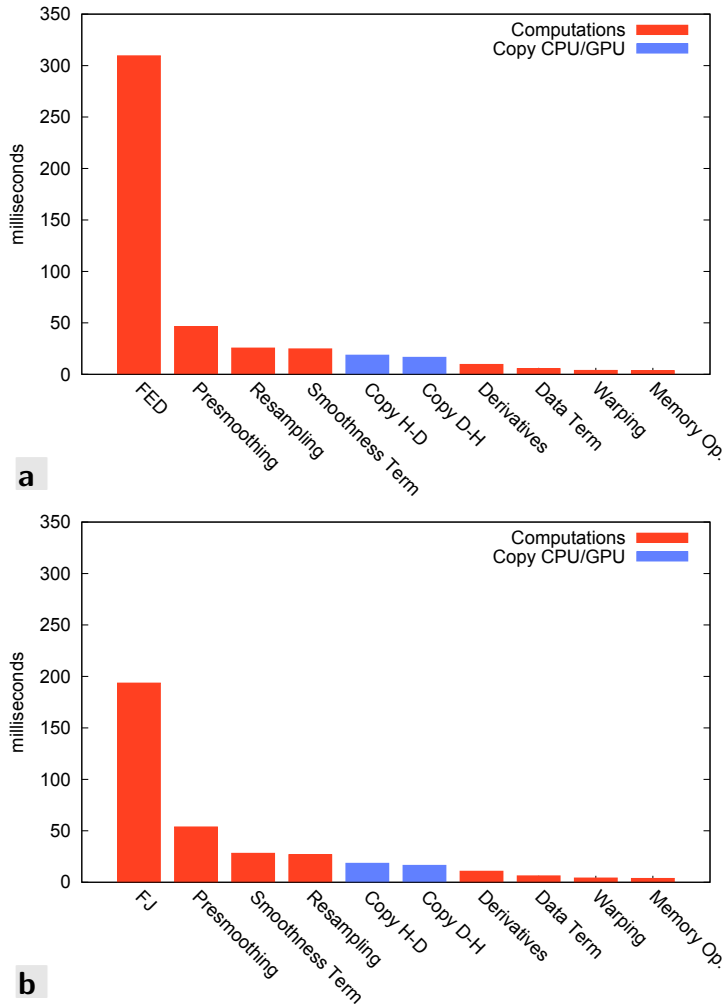



**Figure 6.11**: Profiling of the GPU-based algorithm with **a.** FED, and **b.** FJ, on *Urban2* (640×480 pixels, RGB).

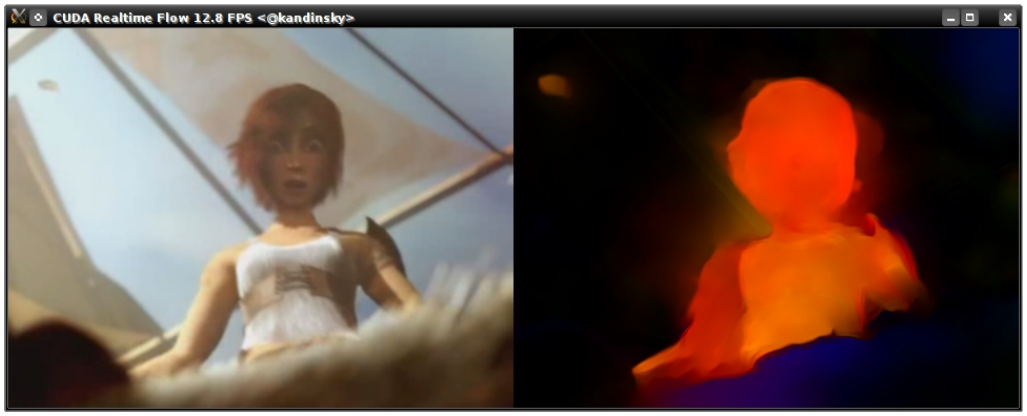



**Figure 6.12**: Screen shot of the live demonstrator for complementary optic flow. The image shows a frame of the *Sintel* movie (CC-BY, Blender Foundation). With a high-speed parameter setting, about 13 FPS are obtained on input frames of size 480×360 pixels.

erator is the Gaussian filter involved at local scale and at integration scale. Following our analysis from Chapter 3, it seems this part cannot be significantly improved further. The same might hold for prolongation and restriction, as well as for the set up of the diffusion tensor, which share a common third place. Both operations are highly memory-intensive, such that it is likely that these operations also cannot be significantly optimised further. Compared to them, the remaining kernels for the image derivatives, data term weights, warping, and memory operations such as `memset` require only a negligible partition of the runtime. This is particularly remarkable for the data-intensive warping step, and indicates that our assumption of a piecewise laminarity of the flow field (see Section 6.4) seems to be valid. This causes a good cache hit rate in this step.

## 6.6 Interactive Real-Time Application

Similar to the application of PDE-based image inpainting, the optic flow framework described above can easily be integrated into an interactive video application. This development follows closely our demonstrator from Section 5.6, and uses similar optimisation strategies.

Even more importantly than for inpainting, optic flow requires a high bandwidth between the CPU and the GPU, i.e. via the PCI bus. Since this bandwidth can hardly be provided by hardware available today, we exploit the fact that our algorithm is given a continuous video stream. This allows

to upload frames one by one to the device, where we leave them in the planar YV12 format as provided by the camera or video converter. This reduces the payload for each frame to 1.5 bytes per pixel, as compared to 24 bytes per pixel for a full RGB sequence in single precision arithmetics.

On the device, these frames are fed into a ring buffer, decoded, and pre-smoothed by convolution with a Gaussian of standard deviation $\sigma$. Following this preparation phase, our algorithm computes the optic flow between the previous and the current frame, visualises the resulting flow field, and discards the oldest input frame. Similar to the inpainting demonstrator from Section 5.6, we compute the necessary visualisation steps such as colour coding and frame buffer assembly on the CUDA device. This allows to hand finished frames over to the OpenGL pipeline without requiring to transfer them back to the CPU first.

Figure 6.12 shows a screen shot of our demonstrator on the *Sintel* movie. It is directly streamed into our algorithm by using `mencoder` as a realtime video transcoder which is attached to the `stdin` interface of our program. For this screen shot, a similar setting as in the generic parameter set was applied, where time-critical parameters were reduced in favour of a faster runtime. Although the algorithm is not fully converged in this case, the outcome still convinces with crisp flow edges and a smooth filling-in of intermediate regions. In this particular case, we obtain 13 FPS on an input of size 480×360 pixels. If we use the same high-quality parameters in the Middlebury benchmark, our algorithm still yields about 2 FPS on this size.

## 6.7 Summary and Conclusion

In this chapter, we have designed a highly efficient method for the minimisation of modern variational optic flow approaches. Following the example of the complementary optic flow model from [ZBW+09], we have developed a general framework for the parallel solution of complicated Euler-Lagrange equations and similar elliptic processes arising in the field of visual computing.

Our algorithm scales very well over inputs in different sizes, and obtains speedups of more than 40 over the original FAS method on the CPU. This allows to provide highly accurate flow fields of coloured RGB image sequences in the size 640×480 pixels within little more than half a second. At the time of the original publication [GZG+10], our GPU-based approach ranked 6th in the Middlebury benchmark with respect to the approximation quality, and constituted the fastest algorithm among the top 10 of most accurate methods. Due to rapid developments over the last months,

it dropped to the 18th place since, where it still claims the position of the fastest method among the top 30.

Large parts of this new algorithm incorporate the work and insights from previous chapters. While presmoothing is done by linear diffusion approaches from Chapter 3, the core of our algorithm is given by the fast explicit diffusion (FED) algorithm from [GWB10] whose application to the GPU has been discussed in Chapter 4. From Chapter 5, we use the efficient hierarchic application of this numerical scheme on GPUs. This collection of thoroughly optimised operators has been complemented by a new GPU-based warping strategy, as well as by novel GPU kernels for the evaluation of the data term and smoothness term contributions.

Moreover, we designed a parallel solver which sets up on the recent Fast Jacobi (FJ) technique [Wei11b]. This new numerical scheme has been shown to converge much faster than FED, while providing results of similar quality. This gives rise to evaluate and use this scheme in GPU-based approaches for a large variety of tasks in visual computing, even far beyond optic flow. The question for an *optimal* relation between FED time steps and FJ relaxation weights seems to be one of the most interesting questions arising in this context. Experiments suggest that mathematical derivations of bounds usually lead to overcautious settings which are not able to exploit the full potential of the algorithm.

The framework we have designed in this chapter shares the high versatility of this solver. Our anisotropic smoothness term constitutes one of the most general formulations for this purpose. In the context of optic flow, it can easily be specialised to obtain other popular smoothness terms such as TV regularisation [ZPB07]. Furthermore, because our minimisation strategy sets up on the Euler-Lagrange framework, the same ideas that we used in this chapter can also be applied to many other variational approaches. In all of these cases, it is likely that the general concepts developed in this and the previous chapters help to tangibly speed up the solution of many challenging problems in image processing and computer vision.

In view of this variety of possible applications, it seems that the development of GPU-based algorithms for PDE-based approaches remains not only an exciting but also a challenging field of research. Our work shows that a handful of carefully optimised, but fully transparent parallel operations is able to solve complex problems faster than any other algorithm proposed in the literature so far. The wide applicability of numerical schemes such as FED or FJ motivates to search for other schemes with similar characteristics. In this context, a major key to astonishing runtimes seems to be the reduction of the memory bandwidth and the total number of operations, as well as a fully data-parallel and simple algorithmic structure. Although

these techniques are usually not optimal on sequential architectures, they easily unveil their full performance on massively parallel systems.

In context of hierarchic algorithms, it turns out that efforts which aim at a *reduction* of coarse-grid operations are in general able to lower the runtime on massively parallel hardware significantly. This is in contradiction to the general paradigm on sequential CPUs, where one typically achieves speedups by offloading *more* operations to coarser grids.

With the end of this chapter, we also leave the area of PDE-based methods. The remainder of this thesis will be concerned with a different class of tasks in the field of visual computing. Although they are also based on physical principles, they are fundamentally different to the problems we have discussed and solved so far. As a consequence, we must approach them with different solution strategies. Nevertheless, we are going to see that these techniques also play an important role in image processing and computer vision, and can even be used to enhance other PDE-based approaches such as PDE-based image compression.

# Chapter 7

# Halftoning

*Some say they see poetry in my paintings.*
*I see only science.*
Georges-Pierre Seurat

## 7.1 Motivation

Let us talk about inkblots. Towards the end of the 19th century, painters established *pointillism* as a new genre and novel way to represent objects on a canvas. Their work influenced both art and technology over more than a century, and is still of great importance in our modern life.

Artists such as Georges-Pierre Seurat were fascinated by the impressionistic idea to reduce images to a few simple primitives such as strokes, and to focus on the expression of light, emotions, and feelings. Nevertheless, they realised that this level of abstraction is yet unable to transport the many facets of light and colour which they wanted to communicate in their paintings. The reason for this limitation was soon identified: A classical mixture of colours on a palette possesses a luminosity which is limited by the used pigments. The more pigments are blended, the darker becomes the overall impression of the outcome.

The idea neo-impressionist painters came up with to solve this problem revolutionised art. In pointillist paintings, whole images are assembled from tiny blurs of colour chosen from an extremely reduced palette. If the observer steps back and regards the painting from a larger distance, such blobs seem to merge to homogeneous areas. Depending on how many blurs are contained, these regions can have any arbitrary nuance of the colour

space spanned by the used palette. In particular, the variety of virtual colours created by this technique exceeds the initial palette significantly up to a continuous scale of nuances. Although pointillism received rather controversial reviews of contemporary critics, works by Seurat, Signac, or Cross are today seen as milestones of modern art. These early works inspired many artists in the following century, and laid the foundation for a scientific approach to basic concepts such as colour theory or human visual perception.

Maybe even more importantly, very similar ideas found their way into technology. Towards the end of the 19th century, printing was significantly accelerated by the invention of automatic line printing machines that replaced manual typesetting by means of lead sorts [Mer40]. While text could now be produced much faster than before, the reproduction of images with high quality still required time and manpower. Whenever photographs were to be produced in more detail than by a simple thresholding, these figures had to be created manually by means of lithography, stippling, or hatching [Sen11, SW87]. This changed with the invention of halftoning as a class of techniques that allowed to reproduce images with a continuous grey scale in black-and-white printing processes.

Halftoning goes back to early reproductions of photographs in *The Daily Graphic* towards the end of the 19th century. Some sources identify an 1 873 photograph of the *Steinway Hall* in Manhattan to be the first halftone [Sul03]. Others argue that this print was supposedly obtained by photoengraving techniques, and call *A Scene in Shantytown*, published by the same newspaper in 1 880, the first 'full-tonal range' halftone [Les06]. However, there is a broad consensus in the literature that either of these works laid the foundations for halftoning to enjoy a broad popularity in the following years [Hol26, Uli87].

Traditional halftone printing plates were obtained by exposing a photolithographic plate with a negative of the original image that is superposed by a glass plate with an etched regular grid [Mur36]. After the development of computers with binary screens and digital printers in the mid of the last century, this idea has been carried over into the digital age. In this chapter, we focus on such digital halftoning methods and their results.

In the literature, many different types of digital halftones are known. Some of them are designed with a purely technical motivation in mind, and aim at a preparation of digital images for reproduction on screen or paper. Thus, they are often working in a purely discrete setup, i.e. they output images in a regular raster of white or black pixels. The foundations for digital image halftoning go back to the mid of the 20th century. Goodall and Roberts show how the impression of a continuous tone can be created by

intentionally perturbing an image with Gaussian white noise prior to thresholding [Goo51, Rob62]. Such white-noise *dithering* algorithm allows to fill regions with a ratio of black and white pixels that correspond to the grey value of the original, and lent its name to a whole class of algorithms which produce similar results. One of these approaches, *clustered-dot ordered dither*, mimics the behaviour of old photographic screens [Uli87]. Different to white noise dithering, it thus describes a deterministic algorithm which guarantees the exact average grey value in a region. Due to its simplicity, this method is very popular today, and is even included in the PostScript standard [Ado99]. However, because the smallest dot reproducible on a device steers the resolution of the whole screen, these *amplitude-modulated (AM) screens* appear much more coarse than the printing technology allows.

*Dispersed-dot ordered dithering* tries to remedy this problem by controlling the grey value of a region by the spacing of dots rather than by their size [Lim69, LK73, Bay73]. This allows to represent arbitrary grey values despite the fact that all dots have the same size. In order to distinguish this method from classical screens, it is often referred to as *frequency-modulated (FM) screening*, or *threshold screening* [JLR07]. The latter is a description of the process. A precomputed thresholding mask with continuous, stochastically well-distributed values is used to threshold continuous images. If the image is brighter than the mask, the respective pixel is rendered in white, otherwise in black. Because of their simplicity and optimality under special conditions [Bay73], such threshold screening approaches are widely used in real printing systems. Often, the used masks are fairly small, such as $16 \times 16$ pixels, and are tiled to threshold larger images. Modern threshold screening approaches still use the same idea, but optimise the masks such that the results look less regular and less artificial without comprising the quality of the method [PTG94, VO08]. However, a fundamental problem remains. Since the mask and the input image are uncorrelated, one cannot give optimality guarantees for arbitrary inputs.

This is different for *blue noise dithering* methods [Uli88]. They create halftones only based on the input image. Distances between single black pixels are optimised such that the halftone represents all grey values as good as possible, with the general appearance being kept as smooth as possible. The quality of such halftones is reflected in their spectral properties, i.e. in the presence and quantity of certain frequencies. *Error diffusion* is the first class of algorithms designed to create such results [FS76]. Such dithering algorithms process the image once in a predefined order, and threshold pixels one after the other. The error, i.e. the excessive or insufficient grey value, is immediately added to the unprocessed neighbouring pixels. After the famous work of Floyd and Steinberg [FS76], many modifications

were proposed to improve these results. These include better masks or grids [JJN76, SA85], image-adaptation and optimisations with respect to dot-gain [Stu81], a grey-value adaptive mask [Ost01], and threshold modulation to avoid artefacts in mid-tone regions [Kno89, ZF03]. As a consequence, error diffusion is not only frequently applied for a high number of tasks, but it also remains an interesting field of research. Several extensions and modification were proposed in the literature. A combination with ordered dither called *dot diffusion* was shown to give more pleasing, but blurred results [Knu87]. Other extensions aim at the enhancement of structures, contrast or fine details [JJN76, JR76, Kno89]. Modern approaches obtain such results by considering many image features, such as frequency, gradient orientation, or contrast [CAO09]. All these methods have in common that they sacrifice theoretical optimality properties to turn halftones perceptually more similar to the original, or to make them more pleasing to the eye.

Over the last decade, a new class of algorithms for blue-noise dithering became popular that did not set up on the old idea of error diffusion. These algorithms based on *direct binary search* (DBS) find a halftone by minimising an error measure on random swaps of pixels [AL96, PQW+08]. Thus, they can be tuned to preserve arbitrary additional features of the image, such as fine structures. However, these algorithms are also computationally expensive, because many pixels must be exchanged until the optimum is reached. In order to obtain this minimum faster, DBS halftoning methods are thus often initialised with an error diffusion result obtained with a classical method [PQW+08]. A related method performs halftoning using a Markovian framework [GRS93]. As it is for many DBS algorithms, this approach obtains results which pronounce both contrast and structure, rather than approximating the original image well. Besides the methods mentioned so far, there are a number of hybrid or rather unconventional halftoning techniques which cannot be assigned to a specific class of algorithms. Most of them possess blue noise properties, but aim at specific applications such as nonphotorealistic screening [OH95].

In the last few years, *continuous* halftoning methods were presented which distribute dots or other primitives freely in the image domain. These grid-less approaches are important for different reasons. On the one hand, they can be used in context of high addressability halftoning [Kan99]. Such methods are required when a printer supports a dynamic dot placement beyond the geometry of a single inkblot. On the other hand, continuous halftoning methods enjoy many applications in the field of artistic screening [Sec02, BSD09, Fat11], object placement [RB85, DHL+98, CSHD03, SAG03, PH10], or sampling [Coo86, MF92, KK03]. The boundaries be-

tween these individual applications are often blurred, because they can technically be realised in a very similar fashion. We see some examples later in this chapter. Continuous dot-placement techniques first arose in the area of sampling, and in particular for importance sampling [Coo86, MF92]. A popular class of techniques in this field of research is Poisson disk sampling, i.e. placement strategies in which dots can be freely arranged as long as all points keep a given minimal distance to their neighbours. For such techniques, many evaluation measures were developed which judge the stochastic properties and homogeneity of point distributions [MF92]. An early approach to continuous image halftoning approximates images with either dots or non-intersecting level lines [PB94]. Either of these visualisation elements is replicated onto equipotential lines of a function that solves the Eikonal equation for the underlying image. In the last years, many new techniques were proposed that obtain continuous point distributions of non-trivial densities. Besides variants of established Monte-Carlo methods [Hal60, Coo86, Han05] and a regular tiling with good stochastic properties [KCDL06], a series of approaches uses Voronoi tessellations [Sec02, BSD09]. Recently, these approaches became very popular due to their high approximation quality, and produced the best halftones of this time [BSD09].

In the following, we discuss a novel approach to continuous digital halftoning that outperforms these previous methods. In its basic form, it represents a point-based halftoning method with uniform dot sizes, and is thus visually most similar to works such as [Sec02, BSD09], and [Fat11]. By this, it represents one instance of FM screening. Its results are obtained by electrostatic laws. This idea was first formulated by Ilbery [Ilb00], although the concept to use potentials for halftoning is much older [PB94]. However, neither of these techniques thought this conception through to the end, but used electrostatic forces only to optimise stencils [Ilb00], or applied other potentials to find halftones [PB94].

## 7.2 Point-Based Halftoning

Before we go into detail about electrostatic halftoning, let us first define some basic notation for halftoning. These considerations define the goal a 'good' halftoning method tries to reach, and form the basis for an objective qualitative comparison and evaluation.

In *point-based halftoning*, we search an image of black dots on a white plane in which points approximate a given (continuous) image $u$, and where points are optimally distributed:

**Definition 7.1 (Halftone, Halftoning Method)**

*Given a continuous domain* $\Omega \subset \mathbb{R}^2$*, and an image* $u : \Omega \to [0,1]$ *that is integrable on* $\Omega$*, the image*

$$h : \Omega \to \{0,1\} \tag{7.1}$$

*is called* halftone *of* $u$*, if there exists a* halftoning method

$$H : (\Omega \to [0,1]) \to (\Omega \to \{0,1\}) \tag{7.2}$$

*such that* $h = H(u)$*, and*

$$\int_\Omega h \, \mathrm{d}\boldsymbol{x} = \int_\Omega u \, \mathrm{d}\boldsymbol{x}. \tag{7.3}$$

□

Unless stated differently, we associate in the following the co-domain value 0 with black, and 1 with white. By (7.3) we require the halftoning method to be grey value preserving [PB94]. Although it is often formulated in a purely discrete way, halftoning can thus be seen as a shock-generating diffusion-reaction system with the halftone as its steady state.

In addition, one typically demands a certain similarity between $h$ and $u$ for a human observer. This assumption sounds intuitive, but bears a fundamental technical problem: By today, the human visual system (HVS) is still not fully understood and can thus not be accurately modelled in a mathematical sense. In the early halftoning literature, one often finds the actual printing or visualisation process and the human perception of the results combined in a *physical reconstruction function* [Uli87]. This function depends on a sampling at a periodic grid corresponding to potential dot locations, a convolution with a linear shift invariant response kernel, and a weighting by material properties of the paper or video screen. Optionally, the imperfectness of the printing process can be taken into account by applying an additional *dot noise* function. However, because modern printing processes are already optimised for a 'good' perception of the result by humans, this comprehensive vision model became less and less applicable. Today, one often tries to separate technological details and human perception.

Hence, the human visual system (HVS) is described by dedicated *HVS models* which are also frequently used in many other areas of visual computing. Experts today widely agree that the underlying filter has low-pass

characteristics [Näs84, KA02, GBAL06, LA08]. Moreover, there is good evidence that certain features are perceived differently than others. Examples are the adaptive stimulation of the retina based on the frequencies of the signal and the background [CCL69, vdBV96], deviations from regular patterns (saliency) [IKN98], virtual continuation of incomplete edge or pattern information [FHH93], or motion [WA85]. Still, there is no mutual consent about a universal HVS model that is able to mathematically explain all effects occurring in human vision. This can best be seen in extremal situations such as optical illusions, or in challenging machine vision tasks such as object and character recognition, or scene understanding. In these fields, computers are usually not able to mimic the human impressions to a satisfying degree.

However, let us assume there is a perfect HVS model $V : (\Omega \to [0,1]) \times \Omega \to \mathbb{R}^n$ which maps a real image to its neural perception in the human brain. The image $u$ and its halftone can then be called similar if

$$\exists\, 0 \leqslant \varepsilon \ll 1 : \int_\Omega \|V(h,\boldsymbol{x}) - V(u,\boldsymbol{x})\| \;\mathrm{d}\boldsymbol{x} \leqslant \varepsilon \tag{7.4}$$

where $\|\cdot\|$ denotes a 'suitable' norm. Although this idea cannot provide a final and comprehensive solution to the problem, it can at least serve as a rough model to provide objective measures for a quality comparison of halftoning methods. In particular, we can incorporate parameters such as viewing distance, lighting conditions, colour-dependent perception, and all kinds of other impairments of the visual system into $V$. In Section 7.7.2, some of these properties are used as error measures to judge the quality of halftones. Besides, there is still the option to perform a user study which is less prone to modelling errors, but could be biased by subjective preferences. For the time being, let us resort to an abstract notion of similarity, and come back to these ideas later.

In order to create a more profound basis of evaluation, we separate point-based halftoning into two operators which are subsequently applied: *Sampling* and *rendering*. The first term describes the placement of dots on the medium, i.e. accounts for the spacing between dots. The second term handles the visualisation of the dot in its desired size. In this particular case, the halftone is generated by sampling, and rendering has a purely technical character. The main computational effort thus lies on sampling.

We should recall that the concept of separating the halftoning process into two operators is frequently found in the literature, although it is not explicitly stated as such [Uli87, Kip01]. Since the times of photographic halftones, people address technical shortcomings such as dot gain or wrong ink application as the drawback of a halftoning method for a particular

device. In this context, rendering is often seen as part of the physical reconstruction function [Uli87], as a technical limitation that must be considered in the halftoning process [Kip01, ZN10], or is simply ignored [BSD09]. Because devices change more frequently than halftoning technologies, the separation into an abstract and a purely technical stage allows to evaluate halftoning methods more objectively. We can thus identify principle theoretical issues and distinguish them from the technical imperfectnesses of an actual printer. Moreover, this concept is very flexible. Most halftoning schemes known today can be separated in two operators, albeit they sometimes look essentially different than for point-based halftoning. For instance, consider an application of AM screening, in which the halftone is steered by the size of dots rather than by their distance. Here, sampling comes down to a trivial regular placement, while the essential grey-value generating methodology is included in the (fault-prone) rendering operator. In Section 7.4.7 we see examples for more complex rendering operators, and for their combination with state-of-the-art sampling techniques.

### 7.2.1 Rendering

Let us first consider the rendering operator, as it is easier to understand and lays the foundations for the sampling step. In the following, we use $\|\cdot\|$ to denote the Euclidean norm: $\|\boldsymbol{x}\| = \sqrt{\sum_i \boldsymbol{x}_i^2}$.

**Definition 7.2 (Point-Based Rendering of a Point Set)**

*Given a finite set of 2-D points $P \in \mathcal{P}(\Omega)$, and a reference image $u \in \Omega \to [0,1]$, a functional $R : \mathcal{P}(\Omega) \times (\Omega \to [0,1]) \times \Omega \to \{0,1\}$,*

$$R(P, u, \boldsymbol{x}) = \begin{cases} 0, & \exists\, \boldsymbol{p} \in P : \|\boldsymbol{x} - \boldsymbol{p}\| < r \\ 1, & \text{else} \end{cases} \tag{7.5}$$

*with the radius $r$ such that*

$$|P|\pi r^2 = \int_\Omega u \, \mathrm{d}\boldsymbol{x} \tag{7.6}$$

*is called a* point-based rendering of $P$ *with respect to $u$.*

□

This *ideal* point-based rendering process only depends on the initial point set $P$ and the image it is about to approximate. However, real-world printing processes often also depend on a set of additional parameters, such as the properties of the output device. Depending on the application, the rendering operator can be implemented differently:

- Often, the number of points is chosen such that (7.6) yields points that possess the same size as single inkblots on the device. In Section 7.4.2, we see in detail how this can be accomplished. Rendering then simply comes down to physically printing a point set with one blot per point.
- For reproduction on a screen, one usually describes $P$ and $r$ by vector graphics. The actual rendering then comes down to resolution-dependent (dynamic) rasterisation, and is performed by a PostScript interpreter or similar engines.
- If we require a discrete representation of the halftoned result and if the points are relatively large (as it is for artistic screening purposes) we can sample the outcome of (7.5) at a regular grid.

Note that neither of these implementations can create a perfect rendering, either due to physical limitations, or due to discretisation artefacts. In particular when it comes to real printing processes, many undesired effects occur. In inkjet printers, dots of ink can be subject to different aerodynamic or electrostatic environmental conditions such that points are offset and vary in size [Uli87, KM79]. Laser printers use fine toner particles which differ in size due to their production process [SBD81]. Finally, the structure, apsorptivity, and reflectance properties of the paper have different impacts on the shape of inkblots and their appearance to an observer [Kip01]. Thus, the range of purely technical limitations even fades seamlessly into a set of virtual issues that arise from the human visual system and cannot fully be distinguished. However, for theoretical considerations, we assume in the following that a perfect rendering process exists. All concrete implementations can then be regarded as sufficiently good approximations of it.

If we consider purely discrete display devices such as computer screens, it can additionally be interesting to come up with a second, alternative rendering function. Such devices are often pixel-based, which means that they rasterise results at a regular grid. Hence, let us consider a second 'discrete version' of Definition 7.2:

**Definition 7.3 (Rasterisation of a Point Set)**

*Let $P \in \mathcal{P}(\Omega)$ be a finite set of 2-D points and $\Gamma = \Omega \cap \mathbb{N}^2$ be a discrete image domain. We call $D : (\mathcal{P}(\Omega)) \times \Gamma \to \{0, 1\}$ a* rasterisation *of $P$ if*

$$D(P,(i,j)) = \begin{cases} 0, & \exists\, \boldsymbol{p} \in P : (i-\frac{1}{2} \leqslant \boldsymbol{p}_x < i+\frac{1}{2}) \wedge (j-\frac{1}{2} \leqslant \boldsymbol{p}_y < j+\frac{1}{2}) \\ 1, & \text{else} \end{cases} . \tag{7.7}$$

□

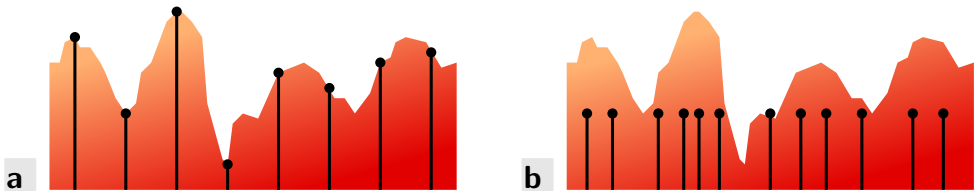



**Figure 7.1**: Sampling (black) of a signal (red). **a.** Regular sampling with equidistant samples of varying value. **b.** Importance sampling with irregularly spaced samples of constant value.

Note that this 'discrete rendering' is no longer performed with respect to a continuously-valued image $u$, but instead we assume unit pixel size and an appropriate number of points in $P$. This fundamental difference is inspired by the insight that rasterisation only depends on the actual output device with respect to which we scale and sample the original image. Point-based rendering, in contrast, shall also be applicable to artistic screening purposes such as stippling or pointillism where fairly large points can be of particular interest.

### 7.2.2 Sampling

Let us now go into detail about the sampling function. It represents the non-trivial part of the halftoning algorithm and has the main impact on the quality of the resulting halftone. Before giving a definition, let us briefly sketch the idea of sampling density functions.

The type of sampling we are looking for in this context is not the standard discretisation of signals we know from many fields such as image or audio processing. There, one tries to approximate a continuous signal with equidistant peaks of varying height. Instead, we use peaks of equal height and allow variations in the place. This difference is visualised in Figure 7.1. Both sampling variants have in common that a convolution of the sampled signal with an interpolation kernel gives a continuous signal that is perceptually close to the original. However, the latter variant is better suited for halftoning purposes, as it already incorporates the binary character of halftoning.

**Definition 7.4 (Irregular Uniform Sampling)**

*In the context of halftoning, an* irregular uniform sampling *is an operator*

$$S : (\Omega \to \mathbb{R}) \times \mathbb{N}^+ \to \mathcal{P}(\Omega), \quad S(u, M) \mapsto P \tag{7.8}$$

*with* $|P| = M$*, and for all* $\boldsymbol{y} \in \Omega$ *there exists a small* $\varepsilon > 0$ *such that* $\mathcal{N}_{\varepsilon,\boldsymbol{y}} = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{y}\| < \varepsilon\}$*:*

$$\int_{\mathcal{N}_{\varepsilon,\boldsymbol{y}}} u(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \sum_{\boldsymbol{q} \in (\mathcal{N}_{\varepsilon,\boldsymbol{y}} \cap P)} \left( \int_{\Omega} \frac{u(\boldsymbol{x})}{M} \, \mathrm{d}\boldsymbol{x} \right) . \tag{7.9}$$

□

In (7.9), the term in parentheses corresponds to the grey value that is represented by one particle. Hence, we require that within a 'suitably small' neighbourhood, the grey value of the sampled version equals the grey value of the continuous image. Our intuition tells us that we are primarily interested in a good locality of $S$. This can be achieved by choosing the sampling operator such that it provides a similarity for sufficiently small $\varepsilon$.

However, the lower bound for $\varepsilon$ cannot be immediately derived as in the case of uniform sampling. There, $\varepsilon$ only depends on the grid size. To this end, it is hard to tell whether an irregular sampling is optimal with respect to the number of points involved in the process.

The number of samples $M \in \mathbb{N}^+$ is a degree of freedom comparable to the grid spacing in uniform sampling. If we choose $M$ very small, we obtain a smaller point set at the cost of a higher approximation error. If we use $M \to \infty$ and integrate over infinitesimal neighbourhoods, we obtain the original image $u$.

In the context of Monte Carlo simulations, the type of sampling given by Definition 7.4 is often referred to as *importance sampling* [Cla61]. In these applications, it is used to analyse the characteristics of an unknown probability density function, and to improve and accelerate the estimation by focussing on 'interesting' areas of this function. To this end, one does not sample the unknown function uniformly, but spends more samples to areas where the values of the function are more likely to give characteristic insights. The distribution of these samples is given by a second probability distribution which describes the degree of importance. Hence, this *biasing density* can be seen as a normalisation relative to which other functions can be evaluated.

Importance sampling enjoys a broad coverage in the literature. A large field of application is traditionally the simulation and automatic evaluation of events in random processes. In signal processing, such models are required to compensate for high noise levels signals are exposed to. This is often the case for satellite communication or radar measurements [CLSY93, Sri02]. In finance and economics, importance sampling is used for a better risk estimation in asset management and stock market in-

vestments, as financial markets are often subject to a multitude of external influences [GHS99, EJ10].

In the past 20 years, importance sampling found its way into computer graphics and image processing. While it was first employed for antialiasing in raytracing [Coo86, Shi91], it soon became an integral part for many fields such as texturing and object placement [RB85, DHL+98, CSHD03, SAG03, PH10, Wei10], the layout of colour filters for digital cameras [Wei10], halftoning [Uli88], and non-photorealistic rendering including many types of artistic screening [Sec02, BSD09].

In the following sections, we focus on the latter applications. In their case, the classical concept of importance sampling for Monte-Carlo simulations is slightly abstracted. The interaction with the actual target density is no longer important, but usually a uniform distribution is assumed. Instead, the approximation quality of the biasing density becomes the major point of interest, and is often evaluated under a multitude of error measures.

### 7.2.3 Sampling + Rendering = Halftoning?

Before we consider a concrete implementation of the previously described operators, let us consider some theoretical aspects and limitations of the process. To this end, we consider the continuous case and construct a halftoning operator out of an abstract sampling operator $S$ and a point-based rendering functional $R$. Depending on the desired output device, we thus select a suitable number of points $n$ and set

$$h'(\boldsymbol{x}) := H'(u)(\boldsymbol{x}) = R(S(u,n),u,\boldsymbol{x}) \ . \tag{7.10}$$

Our goal is to obtain a halftone of $u$, i.e. we expect $h'$ to satisfy Definition 7.1. However, this goal can only be established under special conditions, as there are principle limitations in both rendering and sampling.

Let us have a close look on Equation (7.6). By construction, it seems to describe the grey value preservation of the halftone as specified by (7.3). However, this only true if points do not overlap, i.e. if it holds that

$$\forall\ \boldsymbol{p}_1,\boldsymbol{p}_2 \in P, \boldsymbol{p}_1 \neq \boldsymbol{p}_2 : \|\boldsymbol{p_2}-\boldsymbol{p_1}\| > 2r. \tag{7.11}$$

In very saturated regions, this property cannot be fulfilled, as there is no tiling with circles that could densely fill a rectangular area. As a consequence, circles overlap and leave the same area uncovered at another spot. However, we can find an upper bound of the error, given that $P$ is 'optimally distributed'. The latter condition means that points are placed in a way that the free space is minimised, i.e. they are not intentionally placed on top of each other.

**Theorem 7.1** → Proof 6

*Given a distribution $P$ of points which represents a perfect sampling of an image $u$, a rendering function given by Definition 7.2 approximates the average grey value to at least 96.28%:*

$$\left| \int_\Omega R(P, u, \cdot) \, \mathrm{d}\boldsymbol{x} - |P|\pi r^2 \right| < 0.0372, \tag{7.12}$$

*where $r$ is given by (7.6). Perfect distributions representing a grey value $u(\boldsymbol{x}) > 0.0931$ are rendered in such a way that the average grey value is being preserved.*

□

Note that the optimality condition on the point set $P$ is a relatively strong constraint. For highly textured images, it can even happen that there is no distribution which fulfils the perfectness assumption in Theorem 7.1 . As a simple counterexample, consider thin structures in an image. Usually, the grey value in these regions cannot be represented by an integer number of points, such that two or more regions share the same point. Moreover, the particular shape of the area might not allow the points to arrange in the optimal pattern for this area, as there arise wrong densities of points along image edges.

We see later in this chapter that these limitations only occur on real-world images which contain very dark regions. However, in these cases they arise by construction for all point-based halftoning methods from the literature (see e.g. [Kip01, LA08, ZN10]). In industrial applications, these effects are handled by using the circumcircle of the desired pixel hexagon as the shape of an inkblot [Uli87, ZN10]. Since this setup significantly taints the average grey value, the input image is preprocessed by a tone scale adjustment operator based on the physical reconstruction function. It accounts for a better representation of bright areas. In this case, a coverage analysis similar to Theorem 7.1 is given by Ulichney [Uli87]. It states an efficiency of about 82.7% which means that inkblots intentionally overlap by 20.9%. This is a lot compared to about 3.7% assuming an ideal inkblot size as above and makes a dedicated handling of grey value shifts irreplaceable. Still, this option is frequently chosen in real printing processes, since the ability to print truly black regions is an often requested feature.

For a rendering as in Definition 7.2, we can learn that grey values below 0.0931 are still represented accurately, if we perform a preprocessing tone scale adjustment which over-pronounces black tones. In such case, we create an intermediate image from the original which can even have negative grey values (undershoots). The sampling of this image rendered by the standard

rendering function then yields a better approximation of the original than a standard halftone of the original image. To this end, we modify the sampling in order to correct errors in the rendering operator. This is the reason why we are going to regard this tonemapping operation as an optional extension which is extremely useful for halftoning, but wrong by construction for most sampling applications. We discuss this extension in detail in Section 7.4.3.

Besides theoretical limitations of the rendering function, we shall also note at this point that several technical impairments exist, and often even dominate. The most frequent application of halftones is printing, and distribution of ink or toner particles on paper is subject to a variety of physical effects. Dots are often not rendered with their correct sizes, but they join with nearby blots to one large homogeneous area [LA08]. Liquid ink tends to follow paper fibres and crinkles. In laser printers, small dots might only be printed by a certain probability that additionally depends on the toner coverage in the neighbourhood [Uli87].

To this end, we can regard $H'$ as an approximation of $H$ that fulfils our requirements well enough to be used for most practical applications. The observations made above just tell us that the quality of results yielded by a point-based halftoning method is limited by technical and theoretical issues in both the rendering and the sampling stage. For specific applications, we shall thus individually decide whether the general framework can be adopted to this application at all and be aware of potential sources of error. Moreover, for quality evaluations of halftoning algorithms, we should always distinguish between artefacts that are due to the sampling function, and those which are a principle consequence of the imperfectness of rendering.

## 7.3 Electrostatic Halftoning

Let us now introduce a novel flexible model for a multitude of applications including point-based halftoning, dithering, importance sampling, and many other fields in visual computing. We introduce a framework to design highly accurate sampling operators. The result these operators yield can be rendered using any of the two functions presented in Section 7.2.1, or used as an importance sampling operator within a framework for complex computer vision applications. The work we discuss on the following pages is also published in [SGBW10].

The key property we want to obtain is a uniform distribution of samples within 'flat' areas of the input. Such an arrangement implies that distances between individual inkblots are maximised. If we assume that the human visual system blurs out high frequent fluctuations under a certain viewing

distance, this distribution causes every inkblot to virtually dissolve in an area of equal size. As a consequence, the impression of a smooth greyscale image arises. This concept of maximising distances is a well-known concept in nature, and can be found in both magnetic and electrostatic repulsion laws. While magnetic forces are less suited for our application due to their inherent bi-polarity, the electrostatic framework forms a perfect foundation for our purpose.

Consequently, we regard the sampling points as actors in an electrostatic particle system. In this model, all particles are equally charged and float on a thin glass plate with finite extent. In the moment we release them onto this plane, they repel each other and drift apart. Driven by Coulomb's force, they aim to maximise the distances to all other particles, as this is the energetically optimal state. However, if we hinder them to leave the glass plate, they soon are uniformly distributed over the whole domain. By using this simple idea, we can already sample flat densities, such as uniformly grey images.

As a second step, let us extend this concept to nontrivial images. Without loss of generality, assume the particles have negative charges. Hence, if we attach positively charged objects to the bottom of the glass plate, their electric field influences and attracts the particles moving on top of the plate. As the challenge is to make the particles arrange in a distribution that imitates the original image, we use a simple trick. We charge objects with positive charges relative to the local grey value of the original image. Regions with a higher charge density naturally attract more particles than those with lower charge density, such that the desired state naturally arises after some time.

Let us now model this idea as an image processing system. For simplicity of notation, we first consider grey-valued images. This allows us to have only one density function to approximate at once. Later in Section 7.4.6, we extend this idea to colour images and multi-channel images in general. To ease notation, we introduce an index set $I_P := \{1, \dots, |P|\}$ which describes the set of all particles. Every particle $n \in I_P$ is annotated with a position $\boldsymbol{p}_n$ such that $\{\boldsymbol{p}_n \mid n \in I_P\} = P$, and with a charge $q_n$.

### 7.3.1 Repulsion

First, we consider two particles $1, 2 \in I_P$ and observe the electrostatic force $\boldsymbol{F}_{1,2}^{(R)}$ acting on the first particle. Since it is caused by the electric field $\boldsymbol{E}_2$ induced by the second particle, we first compute its magnitude at $\boldsymbol{p}_1$, and use this information to derive the force.

An important property of our system is the assumption that particles

are only allowed to 'roll' on the glass plate, but must not be lifted off this plane. We are tricked by an illusion if we expect this behaviour to hold naturally: Our mind tends to assume gravitational forces on all objects, but those are not present in our purely electrostatic model. We can handle this problem in two ways: We could design additional forces that pull particles back, such as a second glass plate on top. However, a much easier way of thinking is to embed the system into a 2-D world. In this case, forces are by definition only acting parallel to the plane, and the question of whether particles are 'lifted off' the plane does not arise.

To this end, let us derive the magnitude of $\boldsymbol{E}_2$ at $\boldsymbol{p}_1$. By the theorem of Gauß-Ostrogradski [Mes06], the electric flux around $\boldsymbol{p}_2$ is given by

$$\Phi = \frac{q_2}{\varepsilon_0} \ , \tag{7.13}$$

where $\varepsilon_0$ is the electric constant. Due to its radial nature, the magnitude $|\boldsymbol{E}|$ of the electric field on a circular curve with radius $r$ is given by

$$|\boldsymbol{E}_2| = \frac{\Phi}{2\pi r} = \frac{q_2}{2\pi\varepsilon_0 r} \ . \tag{7.14}$$

Because particle 1 resides at a distance of radius $r = \|\boldsymbol{p}_2 - \boldsymbol{p}_1\|$ from $\boldsymbol{p}_2$, we can compute the magnitude of the repulsive force as

$$|\boldsymbol{F}_{1,2}^{(R)}| = q_1|\boldsymbol{E}_2| = \frac{q_1 q_2}{2\pi\varepsilon_0 \|\boldsymbol{p_2} - \boldsymbol{p}_1\|} \tag{7.15}$$

It remains to determine the direction of $\boldsymbol{F}_{1,2}^{(R)}$. Let us thus denote the unit vector $\boldsymbol{e}_{1,2}$ from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ by

$$\boldsymbol{e}_{1,2} := \frac{\boldsymbol{p}_2 - \boldsymbol{p}_1}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|} \ . \tag{7.16}$$

We want $\boldsymbol{F}_{1,2}^{(R)}$ to act parallel to this vector, and in opposite direction, as it is a repulsive force. With the abbreviation $\frac{1}{2\pi\varepsilon_0} =: k$, i.e. $k$ taking twice the value of Coulomb's constant [Mes06], we finally obtain

$$\boldsymbol{F}_{1,2}^{(R)} = -\frac{k q_1 q_2}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|}\boldsymbol{e}_{1,2} \ . \tag{7.17}$$

In this equation, we can see the impact of our assumed 2-D world. Different to classical electrostatics in the 3-D world, forces decay by $1/r$ instead of by $1/r^2$. This difference is introduced by the flux being integrated over a circle instead of over a sphere.

Let us now compute the overall force acting on a particle $m \in I_P$. By the superposition principle, the electric field $\boldsymbol{E}$ equals the sum over the individual electric fields $\boldsymbol{E}_n$ induced by the point charges $\boldsymbol{p}_n$. Hence, we modify and extend (7.17) and obtain:

$$\boldsymbol{F}_m^{(R)} = -\sum_{\substack{n \in I_P \\ n \neq m}} \frac{k q_m q_n}{\|\boldsymbol{p}_n - \boldsymbol{p}_m\|} \boldsymbol{e}_{m,n} \ . \tag{7.18}$$

By the additional constraint $n \neq m$, we take care of the fact that each particle is only moving in the electric field induced by other particles, but is never subject to its own electric field. This means, a particle never repulses itself.

### 7.3.2 Attraction

Let us now extend this concept to the attraction of particles towards an image $u$. Although the underlying force can be derived in the same manner as for repulsion, there are additional challenges to handle. Different to the sparse set of points $P$, we assume a continuous charge distribution. This means we need to integrate over areas rather than summing over a set of points. Moreover, we want regions to attract particles proportional to their darkness. To this end, we assume a continuous charge density $q(1 - u(\boldsymbol{x}))$ with some constant $q$ over the image domain $\Omega$.

By the same motivation as for repulsion, we can use (7.18), and replace its discrete sum with an integral over the image domain:

$$\boldsymbol{F}_m^{(A)} = \int_\Omega \frac{k q_m q (1 - u(\boldsymbol{x}))}{\|\boldsymbol{x} - \boldsymbol{p}_m\|} \boldsymbol{e}_{m,\boldsymbol{x}} \, \mathrm{d}\boldsymbol{x} \ . \tag{7.19}$$

Note that the notation of the unit vector $\boldsymbol{e}_{m,\boldsymbol{x}}$ has been overloaded. In this context, it denotes the distance from position $\boldsymbol{p}_m$ to $\boldsymbol{x}$:

$$\boldsymbol{e}_{m,\boldsymbol{x}} := \frac{\boldsymbol{x} - \boldsymbol{p}_m}{\|\boldsymbol{x} - \boldsymbol{p}_m\|} \tag{7.20}$$

A fundamental difference arising from the continuous formulation is the behaviour around zero. While in the discrete case, one must exclude the case $\boldsymbol{x} = \boldsymbol{p}_m$, the continuous integral does not have any singularities:

**Theorem 7.2** → Proof 7

*For all particles $m$ residing at a point $\boldsymbol{p}_m \in \Omega$, the integral from (7.19) is absolutely convergent. The force $\boldsymbol{F}_m^{(A)}$ acting on a particle $m$ is bounded.*

□

Nevertheless, this special case can again be of importance for discrete implementations of the method, as we are going to see in Section 7.5.

### 7.3.3 Towards an Iterative Scheme

Let us now sum up the repulsive forces $\boldsymbol{F}_m^{(R)}$ and the attractive forces $\boldsymbol{F}_m^{(A)}$ acting on a particle $m$, and call this force $\boldsymbol{F}_m$:

$$\boldsymbol{F}_m = \int_\Omega \frac{kq_m q(1-u(\boldsymbol{x}))}{\|\boldsymbol{x}-\boldsymbol{p}_m\|}\boldsymbol{e}_{m,\boldsymbol{x}}\,\mathrm{d}\boldsymbol{x} - \sum_{\substack{n\in I_P\\ n\neq m}} \frac{kq_m q_n}{\|\boldsymbol{p}_n-\boldsymbol{p}_m\|}\boldsymbol{e}_{m,n}\ . \tag{7.21}$$

In terms of halftoning, we are interested to find a state in which the forces on all particles vanish. In the following, our goal is to use a particle system to find this state. As a first step in this direction, let us briefly analyse the objective, and refine (7.21). In halftoning, we are interested in the perfect point distribution. This is a setting in which all regions contain a number of particles proportional to their grey value. Thus, a desired property of $\boldsymbol{F}_m$ is that it should vanish if a region is already properly saturated. For simplicity, let us assume particles to be rendered in unit size. Then, we obtain this state of electrostatic neutrality if we assume unit charges, i.e. if we set all $q, q_m, q_n$ in (7.21) to 1 C (Coulomb):

$$\boldsymbol{F}_m = k'\left(\int_\Omega \frac{1-u(\boldsymbol{x})}{\|\boldsymbol{x}-\boldsymbol{p}_m\|}\boldsymbol{e}_{m,\boldsymbol{x}}\,\mathrm{d}\boldsymbol{x} - \sum_{\substack{n\in I_P\\ n\neq m}} \frac{1}{\|\boldsymbol{p}_n-\boldsymbol{p}_m\|}\boldsymbol{e}_{m,n}\right)\ . \tag{7.22}$$

The new constant $k' = \frac{k}{\mathrm{C}^2}$, $[k'] = 1\mathrm{N}$ (Newton), accounts for the fact that we removed the units from the charges.

**Theorem 7.3** → Proof 8

*The superposition of a circular homogeneous region with unit charge and a single particle at its centre are electrostatically neutral. The force $\boldsymbol{F}_m$ on an external particle $m$ induced by this setup vanishes.*

□

By this theorem, it follows that any region that is assembled from such circular primitives can perfectly be approximated by a point cloud such that the whole setup behaves neutral to external particles. This construction seems artificial and far from reality, because arbitrary regions can in general not be densely tessellated with circles. However, if we additionally take

Theorem 7.1 into account, we can give an tight upper bound for the maximal deviation from an ideal electrostatic neutrality state for any particular region. Moreover, experimental results suggest that the electric neutrality even carries over to inhomogeneous regions with arbitrary shape.

This is an important quality criterion for the arising halftoning process. In our particle system, it means that only 'imperfect' regions attract or repulse particles, depending on whether their grey value is under- or over-represented. Neutral regions can even support particles in passing them, by consuming a particle at one side, and emitting another one at the opposite side. This 'impulse' mechanism allows for a very efficient equilibration of unbalanced point densities, even over larger distances. The result of this process is a point distribution in which all regions are optimally represented by the right number of points.

Let us now use (7.22) to construct an iterative scheme that simultaneously minimises the forces acting on all particles. Such schemes are well-known from physical simulations, in particular for computer graphics [Ree83, Rey87]. In these cases, one assigns a mass to all particles, and uses the equality of force to the product of mass and acceleration to compute the accelerations to all particles. If this evaluation is performed in an iterative framework, one can update the velocity of all particles based on their current acceleration. This velocity vector weighted by a small artificial time step size can then be used to translate their position. To this end, one obtains a physically correct animation of the particles.

In the application for halftoning, we are not interested in the course of the evolution, but in a steady state in which an equilibrium of forces holds. Classical inertia-aware particle systems are in general unable to find such state, since they might end up with particles residing on circular of elliptic trajectories around the desired minimum. Such non-trivial states essential to our solar system and to celestial systems in general. As a modification to remedy these problems, we remove the association of velocity and mass to particles. Instead, we perform an artificial time evolution in which the particles are offset by the forces acting on them:

$$\boldsymbol{p}_m^{k+1} = \boldsymbol{p}_m^k + \tau \left( \int_\Omega \frac{1-u(\boldsymbol{x})}{\|\boldsymbol{x}-\boldsymbol{p}_m\|} \boldsymbol{e}_{m,\boldsymbol{x}} \, \mathrm{d}\boldsymbol{x} - \sum_{\substack{n \in I_P \\ n \neq m}} \frac{1}{\|\boldsymbol{p}_n - \boldsymbol{p}_m\|} \boldsymbol{e}_{m,n} \right) . \quad (7.23)$$

Here, superscripts denote the iteration number. Note that in comparison to (7.22), the constant $k'$ is integrated into the artificial time step size $\tau$.

This design follows our assumption that particles are massless and thus abstracts from the inertia of mass. A side-effect of this concept is that the

process approaches the desired solution much faster than a physically correct system: Since we expect the global error to decrease with the number of iterations, the force magnitudes are largest in the first iterations. In contrast to an inertia-aware model where particles must first be accelerated to reach a suitable velocity, they can freely move in this simplified model. Moreover, there is no need to store the velocity of a particle in addition to its position, such that memory requirements are diminished.

As it turns out, the process described by (7.23) needs one additional refinement, as it can get stuck in a local minimum. Although such situation seems to be rather theoretical, these states occur frequently in practice if we consider a 2-D optimisation process. As a simple example, consider a $2 \times 2$ checkerboard, where the two black regions have the charge of one particle, each. It is easy to see that the ideal halftone is given by two particles at the centres of the two squares. However, if we initially place these particles on the centres of the white tiles, they never find their way into the black regions: They are equally attracted to both black tiles and repulsed by the sum of both forces, such that their current position is a local (but unstable) minimum.

In our imagined setup of small particles moving on a glass plate, we can easily solve such situations by performing slight knocks on the experimental setup. These vibrations dislocate all particles by a small offset in a random direction. For our checkerboard thought experiment, this means that both particles have a preferred direction to pass the other particle, and finally end up in the desired state. In our particle model, we can use the same strategy to support a global convergence. To this end, the iteration is periodically interrupted after some iterations. Every particle is then offset by a small vector whose direction and magnitude are selected randomly from a uniform distribution. Since the magnitude shall not be too large, the interval out of which it is drawn is bounded by a small value that decays exponentially with the number of iterations. This ensures that larger displacements are performed when the process is still far from the solution, and still its convergence to a correct solution is not sacrificed. Experimentally, $(-0.6 + 0.1 \cdot \log_2(i_{\max})) \cdot \exp(\frac{-i}{1000})$ proved to be a good choice for the upper bound, where $i$ denotes the current iteration, and $i_{\max}$ is the desired maximal number of iterations. Its shape was determined by practical considerations on the shape of the curve, and the numerical parameters were empirically optimised on a test problem. Still, keep in mind that the only requirements of the desired function are a sufficient elimination of local minima in the first iterations, and a drop-off towards the end of the process. Many other functions with these properties can provide a similar quality.

Let us now discuss the initialisation of the process, i.e. $\{\boldsymbol{p}_m^0 \mid m \in I_P\}$. Because the process is stabilised with respect to local minima, all initialisations lead to similar minima. This allows to distribute particles equally over the image domain, or even to use the halftone of a different image with the same average grey value. However, we can accelerate the process if we place particles close to their prospective target locations. This is motivated by a simple observation: If large potential differences exist in the system, particles can travel large distances. Even though an optimisation of local interactions takes place right from the beginning, it does not show effect in these cases, since the large movements dominate. In turn, if regions are already close to being electrically neutral, particles are only optimised in a local neighbourhood. This leads to a fast convergence.

Clearly, the best initialisation would be given by the solution itself. However, since this configuration is unknown for obvious reasons, good choices are to use any other halftoning method, or to distribute particles in a stochastic way proportional to the grey value of the image. The latter is closely related to classical stochastic dithering approaches. In experiments, both approaches show a similar behaviour and converge to equivalent results. However, the runtime can slightly vary as different halftoning algorithms provide more or less qualitative initialisations. In order to provide a better comparability of electrostatic halftoning to other methods, we are thus going to use the stochastic approach in all experiments. By this, we also prohibit a biasing of the solution by a — potentially better — initialisation. The stochastic initialisation works as follows: First, we overlay the image with an equispaced grid. As the input is usually stored in a discrete manner, the separation into pixels already serves as a good candidate for this purpose. For all particles in $I_P$, we then randomly select a grid cell from a uniform distribution until we find a cell that does not contain a particle, yet. We randomly pick a number between 0 and 1. If this number is greater than the grey value of the image in this cell, we place the particle in this cell. This causes more particles to be put into dark areas of the image than in bright zones.

## 7.4 Modifications and Extensions

The model presented in the previous section can already be used for a multitude of applications, such as for dithering, stippling, screening, and sampling. Examples for this basic technique are shown in Section 7.7.1. In many fields, it is among the best methods for its purpose and yields very good results. However, since these applications differ in their details,

there are additional ways to tune and adapt the method towards these special requirements. In this section, we see some of these modifications and extensions.

### 7.4.1 Dithering

As the first modification, let us have a look on *dithering*. By historic and technical reasons, this technique on discrete locations is still very frequently used, but it is also very different to the basic approach of electrostatic halftoning which we have explored so far. This is also the reason why we handle this topic separately, as it can hardly be combined with any of the following extensions.

If we think back to the beginning of this chapter, the adaptation of electrostatic halftoning to grid-based dithering seems straightforward. Apparently, it suffices to replace the rendering operator by a rasterisation as in Definition 7.3, and the result yields discrete results. However, as it turns out this approach does not lead to results of a sufficient quality. The reason for this issue is that regular hexagonal patterns as they arise from the sampling process cannot be well represented on the rectangular grid. This is shown in Figure 7.2a. Even if we input a plain black image, some pixels of the result do not contain any particles. Consequently, they are rendered in white, although the solution should be entirely black. In the experimental section of this chapter, we see more examples for this issue, and observe in particular that such artefacts are even more severe in the midtone range (cf. Figure 7.21). To this end, we learn that exchanging the rendering operator is important, but does not suffice. A remedy for this problem can only be found if we additionally change the sampling operator. We now discuss two small changes which, jointly applied, lead to the desired result.

As a first step, we create an additional attractive force which pulls particles onto grid positions, i.e. into the centre of pixel cells. If $\boldsymbol{d}_{m\boldsymbol{x}}$ is the vector from a particle $m$ to the closest grid point $\boldsymbol{x}$, this additional force $\boldsymbol{F}_{g,m}$ is given by

$$\boldsymbol{F}_{g_m} = \alpha \cdot \frac{1}{1 + \frac{\|\boldsymbol{d}_{n\boldsymbol{x}}\|^8}{\lambda^8}} \cdot \frac{\boldsymbol{d}_{n\boldsymbol{x}}}{\|\boldsymbol{d}_{n\boldsymbol{x}}\|} \ , \tag{7.24}$$

where the parameter $\lambda = \frac{1}{\sqrt{10}}$ steers the drop-off of the potential depending on the distance of the particle to the grid point $\boldsymbol{x}$, and $\alpha = 3.5$ weights the influence of this new local force on the entire system. The magnitude of $\boldsymbol{F}_{g,m}$ is motivated by the Perona-Malik diffusivity [PM87] and is almost constant within a circle around the centre of a pixel. On the ridge between two pixels, the force almost vanishes. Because the directional contributions are

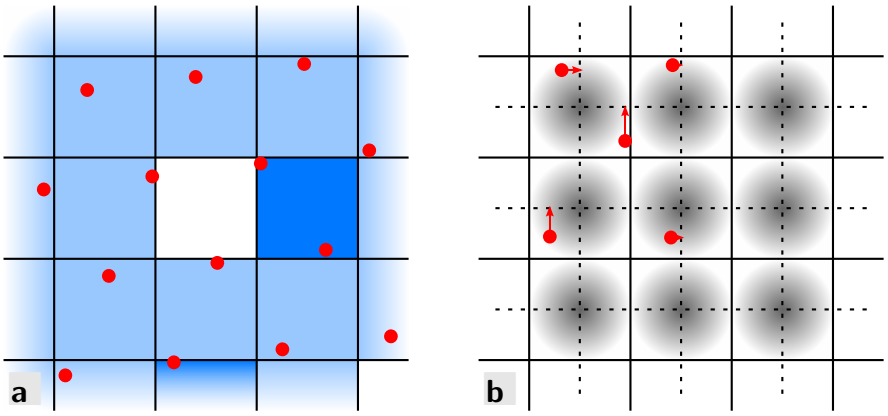



**Figure 7.2**: **a.** Wrong assignment of particles to pixels in the rasterisation of an unmodified sampling result for a black image. **b.** Additional forces and projection to pull particles to the grid.

equal in all directions, this force does not destroy properties such as electric neutrality of correctly filled regions. Instead, it can be seen as a neutral correction term to transfer hexagonal patterns into axis-aligned rectangular structures. Figure 7.2 visualises this 'egg crate shaped' potential as fuzzy black circles.

Although the new force $\boldsymbol{F}_{g_m}$ steers the solution towards the desired result, local minima still cause double assignments of particles to one pixel cell. This effect arises from a desired property of the result: Rotational invariance. Particles now move on an axis-aware force field, but they still behave as in the continuous case, without any axis-aligned bias. This helps the discrete result to carry over much of the quality from the continuous solution, but also supports the establishment of unintended local hexagonal structures. One example for this artefact is a particle residing at the meeting point of four pixels with the surrounding pixels being occupied as well. This is visualised in Figure 7.2b. As the diagonal repulsive forces do not suffice to resolve this state, a trick can be applied to still yield the desired result: Particles are only allowed to move along the grid lines that connect the centres of pixels. In the implementation, this side constraint comes down to one projection step to the closest point on the grid after every iteration. In the figure, grid lines and projections are depicted by dashed lines and red arrows, respectively. By tieing particles onto the grid lines, and thus supporting the constructive superposition of forces, local minima are successfully circumvented.

As we are going to see in the experiments (cf. Section 7.7.3), these two modifications make the model yield discrete dithering results of higher

quality than state-of-the-art methods. Because both changes act only on a local scale within the range of one pixel, global properties of the original model are still well preserved.

### 7.4.2 Point Size Adjustment

Let us now come back to a continuous distribution of points. When we derived the force $\boldsymbol{F}_m$ on a particle $m \in I_P$ in Equation (7.22), we made one important assumption: Particles were assumed to be rendered in unit size, i.e. with a radius $r$ such that $\pi r^2 = 1$. As it turns out, this assumption can be a problem in the printing process because the point size depends on technical parameters of the device. Moreover, a freely selectable radius does not conflict with our definitions for halftoning. In fact, Definition 7.2 explicitly allows an arbitrary number $|P|$ of points in $P$, and adapts the radius such that the average grey value is preserved. By a variation of the number of samples, it is thus possible to modify the radius. In terms of electrostatic halftoning, this change can be represented by a modification of (7.22) such that $r$ becomes a free parameter.

To this end, we consider (7.21), and in particular the charges $q_m, q_n$ of the particles $m, n$. So far, we wanted them to equal the charge $q$ of the image, as the area of one particle was equal to the area of one pixel. This changes if we consider larger particles. For example, let us consider particles of area 2. Intuitively, we want them to behave as if two particles of unit size are coinciding at one place. In other words, the charge of these new particles must be twice as high as before. Generalised to arbitrary factors $\lambda$, we obtain the dependency

$$\lambda q = q_m \ , \tag{7.25}$$

with $\lambda = \frac{r'^2}{r^2}$, where $r, r'$ describe the old and the new radius, respectively. In the standard model, we assume $\pi r^2 = 1$, thus

$$\lambda = \pi r'^2 \ . \tag{7.26}$$

From these considerations, we obtain the modified force $\boldsymbol{F}'_m$ acting on a particle $m$ as

$$\boldsymbol{F}'_m = \frac{1}{\lambda^2} k' \left( \lambda \int_\Omega \frac{1 - u(\boldsymbol{x})}{\|\boldsymbol{x} - \boldsymbol{p}_m\|} \boldsymbol{e}_{m,\boldsymbol{x}} \, \mathrm{d}\boldsymbol{x} - \sum_{\substack{n \leqslant \lambda |I_P| \\ n \neq m}} \frac{1}{\|\boldsymbol{p}_n - \boldsymbol{p}_m\|} \boldsymbol{e}_{m,n} \right) \ . \tag{7.27}$$

Compared to $\boldsymbol{F}_m$ from (7.22), two details change: After obtaining $\lambda$ from $r'$ according to (7.26), we adjust the amount of points $|I_P|$ such that the

average grey value of the new point set rendered with the new radius equals the one of the old point set rendered with the old radius. Secondly, the inverse image $1 - u(x)$ must also be rescaled by $\lambda$ to reestablish the charge equilibrium between the repulsing point set and the attractive force field.

Note that the transformation of $\boldsymbol{F}'_m$ into an iterative update equation as in (7.23) is straightforward. Moreover, since we are interested in the steady state $\boldsymbol{F}'_m = 0$, the constant factor $\frac{1}{\lambda^2}$ vanishes in the artificial time step size $\tau$. In Section 7.4.7, we develop this idea of different point sizes further, and design systems in which particles with different sizes are jointly optimised. While the need for a uniform adaptation of sizes has purely technical reasons, such *second order* halftones are used both for technical and for artistic purposes.

### 7.4.3 Grey Value Correction

To the beginning of this chapter, we saw that dark tones are not well represented in the final halftone. From Theorem 7.1, we learned that this problem is a principle limitation of the rendering operator itself. It occurs even if we consider a perfect sampling operator, and is due to using circles as a structure element. To this end, it is impossible to find a dense tiling of a large plane with circles of constant size. In this section, we remedy this problem by a rendering-dependent preprocessing step.

As is mentioned before, the classic halftoning literature approaches this problem by choosing circumcircles of regular hexagons [Uli87]. In a honeycomb pattern, each partition of space is covered by at least one circle. Since this action reduces, i.e. darkens, the average grey value of the whole result in bright regions, a tonemapping operator is applied to the input prior to halftoning. It artificially brightens up areas such that the 'wrong' halftone using larger circles again well approximates the original.

Assuming a perfect sampling operator, i.e. one which distributes circles in a perfect honeycomb pattern, we can apply a similar idea to correct grey values in the range $[0, 0.0931]$. To this end, we artificially darken very dark areas, sometimes even resulting in a negative grey value. Consequently, more samples are drawn in these regions. Since their relative distances must be reduced by the sampling operator to fit all circles in such areas, they overlap more than usual and cover the area much better than before.

**Theorem 7.4** → Proof 9

*Given a perfect sampling operator and a tonemapping operator*

$$T(x) = \begin{cases} 1 - \frac{\pi}{2\sqrt{3}\ c^2}, & x \leqslant 1 - \frac{\pi}{2\sqrt{3}} \\ x, & \text{else} \end{cases} \tag{7.28}$$

*with $c$ chosen such that*

$$\frac{\pi}{2\sqrt{3}\ c^2}\left(1-\frac{6}{\pi}\left(\arctan\left(\sqrt{\frac{1}{c^2}-1}\right)-c\ \sqrt{1-c^2}\right)\right)=x\ , \tag{7.29}$$

$$\frac{\sqrt{3}}{2}\leqslant c\leqslant 1\ , \tag{7.30}$$

*the halftone of $T(u(\boldsymbol{x}))$ is grey value preserving w.r.t. $u(\boldsymbol{x})$, i.e.*

$$\int_\Omega H(T\circ u,\boldsymbol{x})\mathrm{d}\boldsymbol{x}=\int_\Omega u(\boldsymbol{x})\ \mathrm{d}x\ . \tag{7.31}$$

□

As it turns out, the challenging part in the application of Theorem 7.4 is the evaluation of (7.29) which has to be solved for $c$ depending on $x$. Because of the multiple occurrence of $c$ in nontrivial functions such as the arc tangent, this relation can hardly be inverted. However, it is possible to evaluate the equation numerically for fixed $x$. This can easily be done by the use of a computer algebra system. Since the input is often discrete, we can thus compute a corresponding $T(u(\boldsymbol{x}))$ for all $u(\boldsymbol{x})$. Even more importantly, the input data is often quantised to a few levels, such that we can precompute all potential values in a lookup table. A sample table for grey values quantised to 8-bit depth is shown in Table 7.1.

Note that the values in this table perfectly agree with the theoretical framework given by Theorem 7.1: Below a grey value of 0.0931, there is no $c$ that fulfils both (7.29) and (7.30) which indicates that there is indeed no overlap that has to be corrected. In these cases, the tonemapping operator was thus designed to perform no modification on the image. Moreover, we find that $T(0.0372)=0$, i.e. a black tone renders as a homogeneous grey value of 0.0372 — which is just the statement of Theorem 7.1.

Finally, let us remark that this extension is fundamentally different to the approach of Zhuge and Nakano [ZN10]. Their method consists of a modification to classic error diffusion stencils on rectangular grids, and aims at a biasing of the halftone to form particle clusters on bright areas. By doing so, the authors try to artificially brighten areas by enforcing overlaps between particles such that the overall covered area is smaller. In contrast to this idea, the method proposed in this section solves the problem already at its root. It does so by locally adapting the number of particles according to the natural overlap that arises in the process. Hence, it allows the process to find the optimal solution without causing additional constraints about the dot distribution.

**Table 7.1**: Lookup table for the grey value correction on 8 bit images.

| $u(\boldsymbol{x})$ | $255 \cdot u(\boldsymbol{x})$ | $c$ | $255 \cdot T(u(\boldsymbol{x}))$ | $T(u(\boldsymbol{x}))$ |
|---|---|---|---|---|
| 0.000000 | 0.0000 | 0.866025 | -53.3459 | -0.209200 |
| 0.003922 | 1.0000 | 0.893535 | -34.6518 | -0.135890 |
| 0.007843 | 2.0000 | 0.905129 | -27.2789 | -0.106976 |
| 0.011765 | 3.0000 | 0.914085 | -21.7746 | -0.085391 |
| 0.015686 | 4.0000 | 0.921665 | -17.2408 | -0.067611 |
| 0.019608 | 5.0000 | 0.928358 | -13.3295 | -0.052273 |
| 0.023529 | 6.0000 | 0.934415 | -9.8621 | -0.038675 |
| 0.027451 | 7.0000 | 0.939984 | -6.7330 | -0.026404 |
| 0.031373 | 8.0000 | 0.945163 | -3.8726 | -0.015186 |
| 0.035294 | 9.0000 | 0.950017 | -1.2340 | -0.004839 |
| 0.039216 | 10.0000 | 0.954596 | 1.2184 | 0.004778 |
| 0.043137 | 11.0000 | 0.958935 | 3.5098 | 0.013764 |
| 0.047059 | 12.0000 | 0.963063 | 5.6611 | 0.022200 |
| 0.050980 | 13.0000 | 0.967000 | 7.6873 | 0.030146 |
| 0.054902 | 14.0000 | 0.970765 | 9.6019 | 0.037654 |
| 0.058824 | 15.0000 | 0.974370 | 11.4144 | 0.044762 |
| 0.062745 | 16.0000 | 0.977827 | 13.1337 | 0.051505 |
| 0.066667 | 17.0000 | 0.981144 | 14.7663 | 0.057907 |
| 0.070588 | 18.0000 | 0.984327 | 16.3175 | 0.063990 |
| 0.074510 | 19.0000 | 0.987380 | 17.7912 | 0.069769 |
| 0.078431 | 20.0000 | 0.990305 | 19.1904 | 0.075256 |
| 0.082353 | 21.0000 | 0.993102 | 20.5168 | 0.080458 |
| 0.086275 | 22.0000 | 0.995763 | 21.7684 | 0.085366 |
| 0.090196 | 23.0000 | 0.998274 | 22.9402 | 0.089962 |
| 0.094118 | 24.0000 | n/a | 24.0000 | 0.094118 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

### 7.4.4 Jittering for Stippling

One important application of electrostatic halftoning is stippling. Long before digital screening techniques were invented and could be technically realised, stippled graphics were frequently used in academic textbooks: Besides an accurate description of technical and biological objects with a high level of detail, they put an additional focus on an aesthetic and pleasant appearance of the results. Electrostatic halftoning clearly forms a good basis for stippling algorithms, as it distributes particles well on the image domain and still approximates the average grey value of the original very accurately. Still, halftones produced by this method frequently contain spots which are optimal with respect to their error to the original image, but which unintentionally appear salient for a human observer.

These visual artefacts often occur in regions with a low but non-zero image gradient. The human eye can hardly see these small fluctuations of the grey value but they have a significant effect on the halftone: Let us assume an edge between two regions that differ by a very small grey value. By Kepler's conjecture and Theorem 7.3, we know that each of these regions is best described by a number of particles being arranged in a regular hexagonal pattern. Since the size of the hexagonal patches each particle occupies is proportional to the darkness of the respective region, both hexagonal grids slightly differ in the pairwise distances of particles. If these two grids meet at the edge in consideration, it is in general impossible to connect both grids in the same absolute angle without introducing an error at this point. Consequently, electrostatic halftoning finds states in which these grids meet at a different angle, or even introduces irregular distances in a small region in order to keep the global error low. Both cases are striking to an observer, as the human eye is trained to spot deviations from regular structures as salient features of an image.

From these considerations it becomes clear that we cannot remedy these artefacts without allowing slight approximation errors. The challenge is to improve the visual quality by removing all artefacts without sacrificing too much of the algorithm's quality. A good concept to handle these issues can again be found in nature: Perturbations of the electric field. Apart from other causes, such effects occur because electric conductors are subject to thermal agitation. This electronic noise has first been described by Johnson, and was later explained by Nyquist [Joh28, Nyq28]. For the resulting electric field, this means that the electric field is perturbed over time by Gaussian noise with a small magnitude in an arbitrary direction.

We can use a very similar approach to avoid artefacts in stippled results. Since we are interested in a steady image rather than an oscillating state, we

modify this idea by taking a snapshot of the perturbation field to a random time, and by keeping this state constant over the evolution of the particles. By this change, the perturbation no longer counteracts the convergence of the process. Hence, even with small time steps $\tau$, the system can run into a minimum and is still perturbed to a sufficient degree. Moreover, we can improve the quality of the results if we only apply perturbation to relatively flat areas: Textured regions do not suffer from the aforementioned artefacts, and strong particle displacements would unnecessarily increase the error between the solution and the image that is to be approximated.

To this end, we create a 'smooth' white noise field by adding small displacement vectors at the grid locations and by interpolating bilinearly in between them. The additional smoothness assumption supports the convergence of the particle system without impairing the desired jittering of the result. In order to obtain suitable displacement vectors, we first draw two random variables $a, b$ from a 2-D zero-mean normal distribution with a small standard deviation $\psi$ that is truncated at 1 to avoid infinitely large displacements. The resulting displacement vector $\boldsymbol{v}$ is then obtained from an anisotropic weighting with the image gradient $\nabla \boldsymbol{u}$:

$$\boldsymbol{v}(\boldsymbol{x}) = \left| \left\langle \begin{pmatrix} a \\ b \end{pmatrix}, \nabla \boldsymbol{u}(\boldsymbol{x}) \right\rangle \right| \begin{pmatrix} a \\ b \end{pmatrix} , \tag{7.32}$$

where $\langle \cdot \rangle$ denotes the inner product. In the implementation, $a$ and $b$ can easily be obtained using a random generator for uniform distributions and the Box-Muller transform [BM58]: First, we choose a direction angle $\alpha$ from a uniform distribution over $[0, 2\pi)$. Then, we compute the length $l$ of the vector as $l = \psi \max(1, -2\ln(x))$, where $x$ is a second random variable from the uniform distribution over $(0, 1]$. After the length has been weighted by the projection of the image gradient, both values are transformed back into Cartesic coordinates. In experiments on a unit grid with unit area particles, all visible artefacts already disappear for $\psi \approx 2.5 \cdot 10^{-2}$. However, $\psi$ should also not be chosen too large: For $\psi > 5 \cdot 10^{-2}$, results reveal a noticeable graininess, in particular within dark regions. In Section 7.7.3, we see some example for this issue.

As it turns out, this extension only marginally affects the overall runtime of the process. The electric field $\boldsymbol{E}$ remains constant over the evolution of the particles, as we assume the image to be constant in time. At the other hand, the attractive force $\boldsymbol{F}_m^{(A)}$ from (7.19) depends linearly on $E$ (by $\boldsymbol{F} = q\boldsymbol{E}$ in (7.15)), such that $\boldsymbol{F}_m^{(A)}$ is constant over the evolution as well. Any vector field added to $\boldsymbol{E}$ can thus be interpreted as an additional zero-mean force term $\boldsymbol{F}_p$ which can be precomputed at the beginning of the program run and kept constant over the evolution. The dense vector field

can then be represented by a matrix which is sampled sufficiently fine and which is bilinearly interpolated to obtain perturbation values at arbitrary position.

Finally, let us note that other distribution functions for the choice of $\boldsymbol{v}$ are proposed in the literature. In [SGBW10], the length of the vector is chosen from a uniform distribution. Although this difference remedies the required cutoff of the probability density function, the faster drop-off of the probability density function causes more offsets to be chosen close to zero. Still, the results obtained with this less physically justified model are very similar to the ones yielded by the framework described above.

### 7.4.5 Edge Enhancement

In the definition of halftoning at the beginning of this chapter, we aim at an accurate representation of the original image. However, in particular when it comes to stippling, this property is often not desired. Instead, many halftoning techniques try to create perceptionally similar results by showing what people expect to see rather than what is actually in the image. While both approaches usually agree in flat image regions, they can differ significantly in textured areas: Halftones that approximate the true grey value are often perceived as blurred-out. This illusion comes from the fact that high frequencies can only be represented within the technical limitations of the rendering operator. For electrostatic halftoning, this means that the steepness of edges depends on the size of the rendered dots, as well as on the pairwise distance between particles. In the literature, these effects are remedied by an introduction of artificial high frequencies across edges. This is either done by a direct modification of the input image [JJN76, JR76, Stu81], or by additional structure enhancement in the halftoning method itself [EK91, GRS93, PQW$^+$08].

In the context of electrostatic halftoning, both approaches lead to a similar implementation. If we consider non-turbulent attractive force fields with classical potential sources and sinks, any additional attractive or repulsive force directly comes down to a modification of the input image. On the other hand, turbulences are bad for the convergence of the particle system anyway. They could 'trap' particles on a fixed trajectory of equal, but maybe not minimal, energy. To this end, we can safely assume edge-enhancement to be applied as an operator to the input image rather than an additional structure-enhancing force term.

The particular edge-enhancement operator included in the early dithering literature is known as *unsharp masking* or *high boost filtering* [JJN76, Stu81, GW08]. This method goes back to an old darkroom technique in

which the contrast of an image was virtually increased by exposing an photographic paper jointly with a negative and its blurred positive. The latter, called the mask, was obtained by a previous exposure step with a thin glass plate that blurred the image by distorting the trespassing light. By this technique, the contrast along an edge is locally increased, because grey values along the edge are shifted parallel to the image gradient. Thus, dark areas are darkened towards an edge to a bright area, and vice versa. In digital image processing, this technique comes down to creating a mask as the difference of an image and its blurred version, and by adding this mask to the original.

Given an image $u : \Omega \to \mathbb{R}$ and a Gaussian $K_\sigma$ with standard deviation $\sigma$, the mask $g : \Omega \to \mathbb{R}$ is created as

$$g(\boldsymbol{x}) = u(\boldsymbol{x}) - (K_\sigma * u)(\boldsymbol{x}) \;, \tag{7.33}$$

and is then added to the original using a weight $c$:

$$\hat{u}(\boldsymbol{x}) = u(\boldsymbol{x}) + c \cdot g(\boldsymbol{x}) \tag{7.34}$$

This process has two parameters $\sigma$ and $c$ as degrees of freedom. While the standard deviation $\sigma$ steers the radius of influence, $c$ steers the amount of grey values that are transported. In the literature, one often finds the term 'high boost filtering' for $c > 1$, and 'unsharp masking' for $c \leqslant 1$. During discretisation, both steps (7.33) and (7.34) can be unified into one operator. For dithering, the authors use small stencils with radius 2 [JJN76, JR76, Stu81]. However, since this choice was motivated by the fact that these methods return discrete results with unit pixels, the best values for $\sigma$ and $c$ depend on the problem. In experiments on a unit grid with circular points of unit area, $\sigma \approx 0.6$ and $c \approx 1.0$ give reasonable results on most images (cf. Section 7.7.3).

Finally, let us note that unsharp masking does not fulfil the minimum-maximum principle, i.e. values in the codomain of the filtered image can exceed the range of the codomain of the original image. Although the halftoning process still works on such images, extremal values can negatively affect the overall appearance of the result. This is closely related to the considerations we made in context of grey value correction (cf. Section 7.4.3). Depending on the application and the used rendering function, it can thus be advisable to clamp the preprocessed image prior to halftoning.

### 7.4.6 Colour Halftoning

Let us now lend some colour to the model. Many applications require the reproduction of images in their full range of colours. The problem arising

in this case is very similar to the black and white setup. Again, there are only a small set of different inks available which cannot be mixed before they are transferred onto the medium. Instead, each ink is applied in single dots, such that the overall appearance of all colours taken together shall best approximate the original image. The inks used in this process must be chosen such that all desired colours can be approximated using this technique. Since we do not know *a priori* which pigments are used in this process and what their physical and chemical mixture properties are, we cannot detail on the exact range of colours that can be reproduced. However, as a simple model, it often suffices to assume that colours mix linearly up to the maximal saturation of the used pigments. Let us represent each pigment by a vector in a multi-dimensional colour space. The range of printable colours is then given by the subset of linear combinations of these vectors which, projected to any pigment vector, do not exceed its length.

Multi-colour halftoning is simple if the supported colours are linearly independent, i.e. orthogonal. In this case, the problem decomposes into several one-dimensional problems. A trivial example is the grey valued case, which is by itself linear. For full-colour prints, both the cyan-magenta-yellow (CMY) and the red-green-blue (RGB) model fulfil this assumption. The subtractive CMY model is applied for reflective systems, such as paint on paper, while the additive one is used where the colour of light is to be mixed, such as it is for monitors. The latter one is less frequently used in the last years because modern displays usually support a broad colour range. Even the CMY colour model, which was frequently used in early desktop printers, lost much of its importance in recent times. It was replaced by more complex models such as CMYK. This model complements the traditional CMY approach by an additional 'black' channel. In the following, we consider an achromatic composition in which black replaces the common grey partition of the classic channels. This accounts both for technical and economical reasons in the printing process [Kip01]. When using liquid ink, paper is less soaked through in dark areas. Moreover, there are no ideal inks. This means that the mixture of cyan, magenta, and yellow often yields a brownish shade rather than a truly black colour. This is of particular importance as black is frequently appearing as a printing colour, such as for text. Finally, the application of one ink instead of three helps to reduce printing costs.

The main challenge of many-colour printing such as CMYK is that inks used in this model are no longer independent. This means we are no longer allowed to halftone channels independently of each other. To overcome this problem for electrostatic halftoning, we assign particles to *classes*. Each class contains the subset of all particles which share the same colour. All

particles of one class behave in the same way, such that they become indistinguishable. In order to model the repulsion between particles of different classes, we introduce the concept of *interaction matrices*. Interaction matrices are symmetric square matrices which contain one row and column per class. A zero entry indicates that two classes are independent of each other, while a non-zero entry shall be read as a weight to the force acting on a particle from one class when it is repulsed by a particle of the other class. For the moment, we design attractive and repulsive forces to be symmetric. This means the same matrix is applied to both repulsion and attraction. However, later in this chapter we also see other applications in which this requirement is intentionally dropped (cf. Section 7.4.7).

In an orthogonal colour space, the interaction matrix is simply given by the identity because all particles interact with other particles of the same class, but not with any other particle. For more complex systems such as CMYK, the off-diagonals become nontrivial to model interactions between particles from different classes. In such cases, we can decompose the interaction matrix into a linear combination of an identity matrix and a *non-overlap map*. While the first part covers interactions of particles of the same class with each other, the second part takes care that particles of different class only overlap where desired. For example, let us consider an achromatic CMYK colour space. It can be computed from the CMY model by making the black K channel take over the grey partition of all channels [Kip01]. Speaking of particles, this means that we introduce the new class of black (K) particles as a 'stack' of three different chromatic particles. To this end, a given colour vector $(c, m, y)^\top_{\text{CMY}}$ in the CMY model is transformed to CMYK by using an achromatic decomposition [Kip01]:

$$\begin{pmatrix} c \\ m \\ y \end{pmatrix}_{\text{CMY}} = \begin{pmatrix} c - \min(c,m,y) \\ m - \min(c,m,y) \\ y - \min(c,m,y) \\ \min(c,m,y)) \end{pmatrix}_{\text{CMYK}} . \tag{7.35}$$

For such a CMYK vector, the interaction matrix is set up as

$$\underbrace{\varphi \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{weighted identity}} + \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}}_{\text{non-overlap map}} = \underbrace{\begin{pmatrix} 1+\varphi & 0 & 0 & 1 \\ 0 & 1+\varphi & 0 & 1 \\ 0 & 0 & 1+\varphi & 1 \\ 1 & 1 & 1 & 1+\varphi \end{pmatrix}}_{\text{interaction matrix}} . \tag{7.36}$$

The non-overlap map arises from the fact that, by (7.35), a black particle represents the union of all chromatic colours. To this end, we neither want

**a**

| | C | M | Y | K |
|---|---|---|---|---|
| **C** | 2 | 0 | 0 | 1 |
| **M** | 0 | 2 | 0 | 1 |
| **Y** | 0 | 0 | 2 | 1 |
| **K** | 1 | 1 | 1 | 2 |

**b**

| | C | M | Y | R | G | B | K |
|---|---|---|---|---|---|---|---|
| **C** | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| **M** | 1 | 2 | 1 | 1 | 1 | 1 | 1 |
| **Y** | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| **R** | 1 | 1 | 1 | 2 | 1 | 1 | 1 |
| **G** | 1 | 1 | 1 | 1 | 2 | 1 | 1 |
| **B** | 1 | 1 | 1 | 1 | 1 | 2 | 1 |
| **K** | 1 | 1 | 1 | 1 | 1 | 1 | 2 |

**Figure 7.3**: Interaction matrices for **a.** CMYK and **b.** CMY-RGB-K.

a particle of a chromatic colour to overlap with a black particle, nor do we desire two particles of the same colour to overlap. In contrast, it is well desired that different chromatic particles overlap, thus the zeroes at the respective entries. Since a good distribution of particles of one colour and the non-overlap property are conflictive, the free parameter $\varphi$ acts as a weight to find a compromise. Experimentally, $\varphi = 1$ proves to be a good choice. In this case, the interaction matrix comes down to the one shown in Figure 7.3a. To simplify readability, also in respect to further extensions, it has been written in tabular form with the colour abbreviations as indices to both rows and columns. Still, it is to be interpreted as a matrix that can be multiplied to a column vector in the respective colour model.

Note that besides this *multiplicative* impact of the parameter $\varphi$, it can also be applied *additively* [Sch11]. This leads to weights $(1 - \tilde{\varphi})$ and $\tilde{\varphi}$ for the two matrices. While the case $\tilde{\varphi} = 1/2$ is equivalent to $\varphi = 1$ from above, the advantage of this choice lies in the better numerical reproducibility of the extremal case $\tilde{\varphi} \to 1$, which is given by $\varphi \to \infty$ in our case. However, since this case is usually not considered in real applications, we can stay with our simplified notation. Later in this chapter (cf. Section 7.4.8), this choice also helps us to write up more complicated interaction matrices in a relatively concise manner.

The implementation of a colour halftoning process is simple. First, the input image is converted to the CMY colour space and decomposed into CMYK using the achromatic split as in (7.35). Then, all channels are initialised with an appropriate number of particles, while the non-overlap map is used to avoid overlaps right from the beginning. Finally, the system is jointly optimised for all colours, where interactions between any two particles are weighted by the corresponding entry in the interaction matrix.

Interestingly, this multi-colour halftoning concept is much more versa-

tile as it seems at first glance, and can even be extended to an arbitrary number of colours. This is important for modern desktop inkjet printers, but also for artistic purposes such as digital pointillism. In both cases, the available inks or pigments are only able to reproduce colours that lie within a certain subspace of the full colour space, their *gamut*. Adding more pigments enlarges this region and provides the opportunity to tune the appearance of printed images to more closely resemble the human impression of the depicted scene. In advertisements for printers, this property is often announced as the 'brilliancy of colours' the printer is able to produce. However, experiments also show that the number of available printing colours is usually limited by a loss of spatial resolution in the result. In Section 7.7.3, we go into more detail about this issue.

As one example for a more involved model, let us consider a CMY-RGB-K colour space as it is supported by many large scale printing machines. Similar to the achromatic decomposition into CMYK, we can further split away R from M and Y, G from C and Y, and B from C and M. For CMY-RGB-K, this split is unique and commutative. For the non-overlap matrix, several choices are possible and valid. However, since all combinations of chromatic colours are already present as distinct particles, it is suitable to forbid any overlaps. This comes down to choosing the non-overlap matrix as $1^{7\times 7}$. With $\varphi = 1$, we eventually obtain the matrix depicted in Figure 7.3b. We see in Section 7.7.3 that this theory can easily be extended to many other colour models, such as 'asymmetric' CMY-RG-K or CMY-RB-K models as they are used in modern desktop printers.

Whenever a unique split into the required number of colour classes is impossible, we can assume additional constraints that resolve ambiguities. One toy example for a better understanding of this issue can be constructed by removing the K component from 7-colour printing, thus obtaining in a CMY-RGB model. A black region must then be expressed by a combination of other colours, but this combination cannot uniquely be computed by an achromatic decomposition as above. In fact, a red-cyan overlay perform as well as a green-magenta or a blue-yellow overlay. Even a linear combination of these pairs, such as 30% red-cyan and 70% green-magenta, still fulfil the concept of achromatic decomposition. Thus, we need additional constraints that steer the colour space transformation towards a unique solution. Usually, simple concepts such as a minimisation of the $L^2$ norm of the arising colour vector suffice to obtain a good solution. In terms of the simple CMY-RGB example, this strategy would cause black to be uniquely represented by $(1/3, 1/3, 1/3, 1/3, 1/3, 1/3)_{\mathrm{CMY-RGB}}$. Similar ideas can also be applied to more complex models, such as palettes with many different shades of red or green, but finding the solution is more tedious in these cases.

### 7.4.7 Second Order Screening

In the previous section, we laid the foundations for the application of electrostatic halftoning to printing processes. In order to enable this method for being used as a preprint stage in large-scale printing processes, we now address the technical problem of *dot loss*, and the minimisation of such effects by a tailored adaptation of electrostatic halftoning to this issue.

Dot loss occurs for different reasons, and is more likely the smaller the dot to be printed is. In laser printers, the inhomogeneous electrostatic attraction of toner particles is a typical reason for this issue [Kip01]. In bright regions, it can happen that the transfer unit attracts less toner particles than necessary to create a specific grey value, which causes small dots not to be printed at all. A similar effect can be observed for other printing techniques if paper with a higher graininess is used. In this case, small dots possess different reflectance properties depending on whether they are printed on top or on side of a paper fibre. Both scenarios have in common that only very small dots are affected. Large dots are printed in any case, and since their areas overlap whole paper fibres, they are also less subject to changes in reflective properties.

In the printing industry, people frequently address this problem by choosing different dot sizes and aim at a distribution of all types of dots over the whole image domain. In comparison to AM and FM screening, this technique is usually called *hybrid screening* or *second-order (frequency-modulated) screening* [Kip01]. Since large dots are perceived as clusters of smaller dots, these results look more grainy than first-order halftones. However, they have also fundamentally different spectral properties. High frequencies vanish, and the results obtained by such methods are often classified as *green-noise* halftones as opposed to standard halftones which possess blue-noise characteristics [LAG98, LA08]. The goal of the extension we are going to discuss in this section is to complement the idea of electrostatic halftoning by the additional option of different inkblot sizes to obtain such results. As a by-product of these considerations, we also obtain an interesting effect filter which can be used for many applications in the field of non-photorealistic rendering. For simplicity of notation, let us first consider two classes, each containing particles with one constant (but class-wise different) size. Later, we generalise this model to arbitrarily many classes.

From the electrostatic motivation in Section 7.3 and the optional point size adjustment in Section 7.4.2, it seems obvious to regard large particles as clusters of smaller ones. This comes down to a repulsion that is proportional to the areas of the participating particles. To this end, a particle

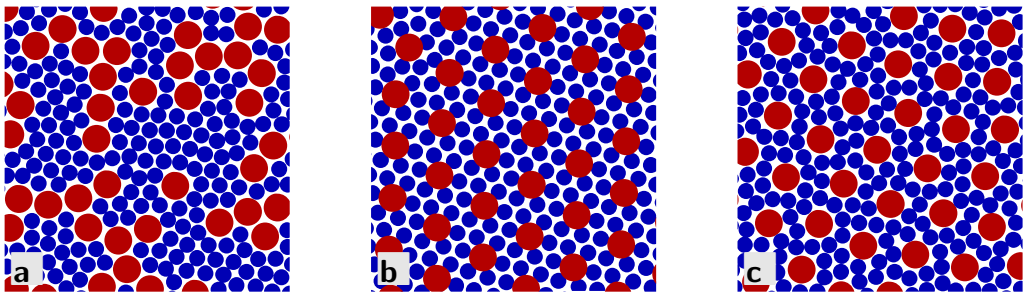



**Figure 7.4**: Comparison of different repulsion models for second order screening. **a.** No bias. **b.** Bias on both the small and large particles. **c.** Bias only on large particles.

with an area of size $x$ and a particle with area 1 repulse $x$ times stronger than two unit particles. Two large particles, each with area $x$, repulse $x^2$ times stronger than two unit particles. However, such a naive mass-based approach turns out to be insufficient to obtain the desired solution. The reason is that, if we observe the result from a larger distance, all that can be identified as a single object are the larger dots, while smaller dots are already blurred out. However, we do not require a certain distribution of large dots so far such that we can replace any set of $n$ small dots by one big dot and vice versa. This leaves the algorithm the freedom to form extremal cases, like assigning all big dots to one side of the image, and all small dots to the other. Figure 7.4a shows this issue for a constant grey value of $^{60}/_{255}$.

A natural remedy to this problem is to regard the particle system as a linear combination of all single-class interactions on the one hand (small with small, large with large), and the complete interaction on the other. This comes down to weighting the main diagonal of the interaction matrix with small constants that are larger than 1. The desired result would be one in which all sets of equally-sized particles, as well as the whole of all particles each provide a good approximation of the image. If we speak in terms of interaction matrices, such design is equivalent to multiplying the same constant factor to all entries on the main diagonal.

Surprisingly, also this approach does not yield an optimal solution. To understand the fundamental problem, let us consider the setup depicted in Figure 7.4b. Here, all three contributions were equally weighted, i.e. we expect the set of large particles, the set of small particles, and the complete set to give equally good approximations of the underlying density. However, the latter two constraints are in conflict. While the smaller particles must distribute equally in order to yield a good per-class approximation, they must at the same time leave gaps for the larger particles such that the

| | **S** | **L** |
|---|---|---|
| **S** | $A_\mathrm{S}^2$ | $A_\mathrm{S}A_\mathrm{L}$ |
| **L** | $A_\mathrm{S}A_\mathrm{L}$ | $CA_\mathrm{L}^2$ |
| $\mathbf{K}^+$ | $A_\mathrm{S}$ | $(1+w_\mathrm{L}(C-1))A_\mathrm{L}$ |

**Figure 7.5**: Interaction matrix for the interaction between small particles (S) with area $A_\mathrm{S}$, and large particles (L) with area $A_\mathrm{L}$.

overall particle cloud gives good results. Such a setup is impossible, and the establishing compromise contains overlapping small and large particles (cf. Figure 7.4b). These gear-like structures are certainly undesired for all applications, because they appear noisy and do not preserve the average grey value.

One solution to all of these problems is to drop the constraint of a good distribution of small particles in favour of the remaining constraints. As a consequence, only the whole set and its partition of large particles are required to distribute well. The small particles distribute such that these requirements are fulfilled, and have only a small impact on the placement of large particles. The sketch in Figure 7.4c shows such a setup which has a similar appearance to the unconstrained case in Figure 7.4a, but features a uniform distribution of large particles.

The interaction matrix required to obtain such results is shown in Figure 7.5. Different to colour halftoning, particles of all sizes are attracted by the same force field. This is the reason why we now distinguish a repulsive part (S, L vs. S, L), and an attractive part ($\mathrm{K}^+$ vs. S, L). The attractive part arises from the fact that the charge density of the image is normalised, i.e. the area of the (virtual) interaction partner $\mathrm{K}^+$ has unit size: $A_{\mathrm{K}^+} = 1$.

Note the constant $C$ as the important difference to the unconstrained model. It steers the influence that small particles have on large ones. The weight $w_\mathrm{L}$ describes the quotient of the area covered by large particles over the area covered by all particles, and is required in order to keep the system electrically neutral. It increases the attraction of large particles to compensate for the amount of additional repulsion caused to them by the other large particles:

$$(1-w_\mathrm{L})\cdot A_\mathrm{L} + Cw_\mathrm{L}A_\mathrm{L} \tag{7.37}$$
$$= (1+w_\mathrm{L}(C-1))A_\mathrm{L}\ . \tag{7.38}$$

If $C \approx 1$, large particles are pushed away if a cluster of small particles can yield a better approximation. For $C \approx 2$, we obtain a state as in Figure 7.4c,

where the structure of large particles can be slightly distorted in order to fit full clusters of small particles in the arising gaps. Finally, if we choose $C \gg 2$, large particles are forced to align in an optimal arrangement, regardless of whether small particles can fill the gaps or not. A more thorough evaluation of these issues is given in Section 7.7.3 (cf. Figure 7.39). We are going to see that the choice $C = 2$ can effectively suppress gear-like structures while still allowing a high quality of the results.

Finally, let us remark that this concept for second order screening can be generalised to an arbitrary number of different classes. To this end, we simply configure the system with respect to three sets of variables, and maintain different weights $C_M$, $C_L$ for the medium and the large particles, respectively:

1. **Sizes** of particles. In some applications, these dimensions are already prescribed by technical limitations of the device. Where it is not, a smart tradeoff must be found. If particles differ only marginally in size, second order screening does not have any benefits over classical electrostatic halftoning. On the other hand, the ratio between the smallest and the largest particle must also not be too large, as unpleasant 'halo' artefacts can occur. We see examples for this issue in in Section 7.7.3.

2. **Amounts** of particles. We must find a 'good' ratio between large and small particles, such that enough small particles are available to fill the gaps between the larger ones. Still, we must introduce as many large particles as possible, because the human eye easily spots single occurrences of large particles as salient points. Experimentally, $w_{\mathrm{S}} = 2/3$, $w_{\mathrm{L}} = 1/3$ yield good results. For three classes, this concept carries over to $w_{\mathrm{S}} = 2/3$, $w_{\mathrm{M}} = 2/9$, $w_{\mathrm{L}} = 1/9$, and so on.

3. **Priorities** of particles. $C_x$ should be chosen as an ascending series, with $C_{\mathrm{S}}$ for the smallest class being 1. The powers of 2 seem to be a good choice, i.e. for three classes $C_{\mathrm{S}} = 1$, $C_{\mathrm{M}} = 2$, $C_{\mathrm{L}} = 4$.

Figure 7.6a shows the arising interaction matrix for three classes, where $C_x$ was chosen as above. Part b shows the arising patterns on a flat image of grey value $100/255$, with areas $A_{\mathrm{S}} = 1$, $A_{\mathrm{M}} = 3$, and $A_{\mathrm{L}} = 5$. We see that the largest particles are best distributed, although they are not on a perfect grid. The other two classes fill up the spaces left by the more prioritised class, and establish regular grid patterns where possible. In Section 7.7.3, we perform some experiments on real-world images, and evaluate the choice of parameters in detail.

**a**

| | **S** | **M** | **L** |
|---|---|---|---|
| **S** | $A_S^2$ | $A_S A_M$ | $A_S A_L$ |
| **M** | $A_S A_M$ | $2A_M^2$ | $A_M A_L$ |
| **L** | $A_S A_L$ | $A_M A_L$ | $4A_L^2$ |
| $\mathbf{K}^+$ | $A_S$ | $(1+1w_M)A_M$ | $(1+3w_L)A_L$ |

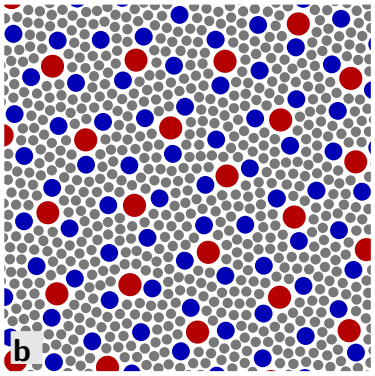



**Figure 7.6**: Second order screen with three classes ($A_S = 1$, $A_M = 3$, $A_L = 5$). **a.** Interaction matrix. **b.** Arising patterns for each of the classes.

### 7.4.8 Multi-Class Sampling

Both multi-colour halftoning and second order screening are based on a force matrix that weights the interplay between their actors. As it turns out, force matrices are a very flexible and powerful tool when it comes to the placement of heterogeneous objects on canvas. In such cases, one often considers much more complex rendering operators than one that only draws dots. We can use our generated samples as placeholders for people at a public place, cattle in a meadow, or trees in a forest. This class of techniques is usually referred to as *object placement* and is frequently used in the generation of photo-realistic scenes in computer graphics [RB85, DHL+98, CSHD03, SAG03, PH10]. If 2-D sprites are distributed, one often calls this process *procedural texture creation*. The resulting images can be arbitrarily large without any periodicity and are usually used for texturing large objects or ground planes in 3-D scenes [Per85, LD05]. Samples can also describe locations at which environment maps are sampled in order to simulate correct illumination and shadowing of scenes [CD01, ARBJ03, KK03]. The brighter the environment map at a specific point, the more light sources are allocated for this area such that the all objects are illuminated according to this virtual environment.

All these examples have in common that we are no longer interested in the rendering operator as it depends only on the application (and perhaps artistic considerations). Instead, we focus on the extended sampling operator that forms a uniform basis for all approaches. As it is for second order screening and colour halftoning, each dot is now assigned properties such as object type and size. This is the reason why all these approaches are comprisedly called *multi-class sampling*. By assigning and configuring classes

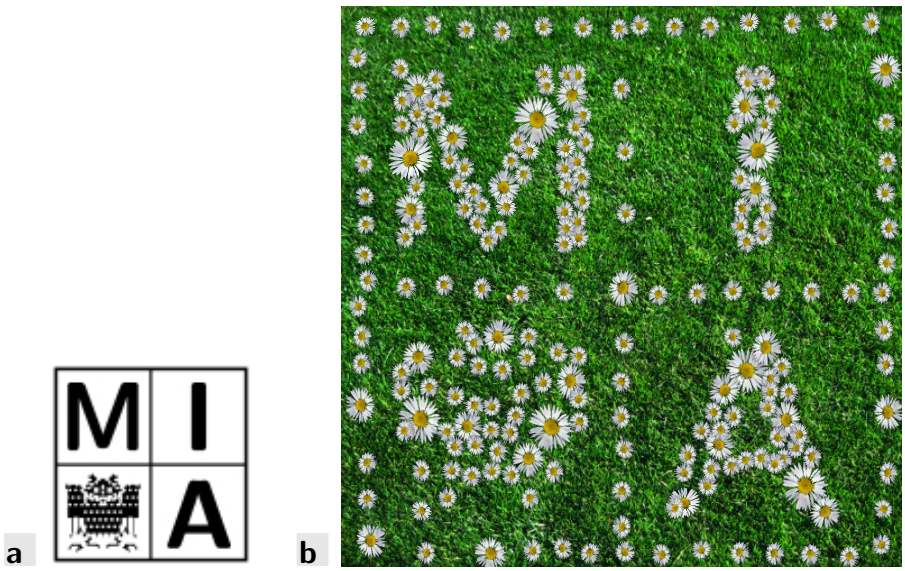



**Figure 7.7**: Sampling of **a.** a single-class density showing the logo of the MIA group with **b.** daisies in three sizes. Daisy sprite by André Karwath, CC-BY.

of objects and their interactions, we can model any setup that requires a locally adaptive distribution of heterogeneous samples on a plane. This is done by generalising different extensions such as multi-colour halftoning, second order screening, and jittering into one general and versatile concept.

Let us use a simple example from object placement to discuss the necessary modifications for a general multi-class sampling framework. Assume we want to distribute flowers on a flower field, and 'plant' them according to a predefined probability of presence. In order to make this scene look natural, we want these flowers to vary in size and shape, and to be well distributed. By the latter aspect, we require nearby plants not to overlap, as they would not do so in nature either, and to approximate the desired charge density best possible. A classic approach in the literature is to distribute objects independently from their size [LD05]. However, this approach is nothing more than a workaround, since it only works if the variation of sizes is small enough such that the wrong placement is not recognised by the observer.

If we only regard one class of objects, second order screening and a new rendering operator can already give reasonable results. One example of how such image can look like is given in Figure 7.7. It shows a binary image rendered with daisy sprites on a grass texture. In order to make the appearance more natural, we use sprites in three different sizes and rotate each individual sprite by a random angle to avoid regular patterns.

Let us now extend this idea to an arbitrary number of different classes

of objects. In such mixed setups, we want 'good' distributions of

1. **each individual class** of objects, and
2. **all power sets** of these classes (including the **entire set**).

Within each of these sets, we additionally require the properties known from second order screening. This means that if we take the subset of all objects larger than a certain threshold from any of the aforementioned sets, they must again be well distributed.

These requirements are very restrictive, but they can easily be approximated if we assemble the underlying interaction matrix from known concepts. To this end, we interpret the different classes of objects as colours that may or may not interact with any other class. Within each class, we can have an arbitrary number of particles that underlie the laws set up in context of second order screening. If we re-use the idea of interaction matrices (see (7.36)), we can again write the interaction matrix as a linear combination of a weighted self-interaction matrix and a non-overlap matrix. However, both matrices are now block matrices with second order screening sub-matrices where an interaction shall happen, and zero sub-matrices where no interaction takes place. The attractive interactions for all classes then follow by straightforward computations.

Let $\boldsymbol{\mathcal{S}}$ be the repulsive part of a second order screening interaction matrix such as in Figures 7.5–7.6. If all $n$ classes of particles have the same size configurations, and all classes interact with each other, we obtain

$$\underbrace{\varphi \cdot I^n \otimes \mathcal{S}}_{\substack{\text{weighted block}\\ \text{diagonal matrix}}} + \underbrace{1^{n\times n} \otimes \mathcal{S}}_{\substack{\text{non-overlap}\\ \text{map}}} = \underbrace{\begin{pmatrix} (1{+}\varphi)\boldsymbol{\mathcal{S}} & \boldsymbol{\mathcal{S}} & \dots & \boldsymbol{\mathcal{S}} \\ \boldsymbol{\mathcal{S}} & (1{+}\varphi)\boldsymbol{\mathcal{S}} & & \boldsymbol{\mathcal{S}} \\ \vdots & & \ddots & \\ \boldsymbol{\mathcal{S}} & \boldsymbol{\mathcal{S}} & & (1{+}\varphi)\boldsymbol{\mathcal{S}} \end{pmatrix}}_{\text{interaction matrix}} . \tag{7.39}$$

In this context, $I^n$ denotes an identity matrix of size $n \times n$, $1^{n\times n}$ a matrix of the same size whose entries are all 1, and $\otimes$ represents the Kronecker product [BSMM08]. Independence between two classes can be expressed by replacing the respective sub-matrices in the non-overlap map by a zero block, i.e. by exchanging $1^{n\times n}$ by a suitable other matrix. Objects of these classes are then allowed to overlap and do not interact at all. In the same style, it is possible to weight the interactions between different classes by a scalar multiplied to the respective block. However, all modifications to the non-overlap map must preserve its symmetry, and the interaction entries for attraction must be adjusted accordingly.

| | $\mathbf{S}_a$ | $\mathbf{L}_a$ | $\mathbf{S}_b$ | $\mathbf{L}_b$ |
|---|---|---|---|---|
| $\mathbf{S}_a$ | $(\varphi+1)A_\mathrm{S}^2$ | $(\varphi+1)\ A_\mathrm{S}A_\mathrm{L}$ | $A_\mathrm{S}^2$ | $A_\mathrm{S}A_\mathrm{L}$ |
| $\mathbf{L}_a$ | $(\varphi+1)A_\mathrm{S}A_\mathrm{L}$ | $(\varphi+1)CA_\mathrm{L}^2$ | $A_\mathrm{S}A_\mathrm{L}$ | $CA_\mathrm{L}^2$ |
| $\mathbf{S}_b$ | $A_\mathrm{S}^2$ | $A_\mathrm{S}A_\mathrm{L}$ | $(\varphi+1)A_\mathrm{S}^2$ | $(\varphi+1)\ A_\mathrm{S}A_\mathrm{L}$ |
| $\mathbf{L}_b$ | $A_\mathrm{S}A_\mathrm{L}$ | $CA_\mathrm{L}^2$ | $(\varphi+1)A_\mathrm{S}A_\mathrm{L}$ | $(\varphi+1)CA_\mathrm{L}^2$ |
| $\mathbf{K}_a^+$ | $(\varphi+1)A_\mathrm{S}$ | $(\varphi+1)(1+w_{\mathrm{L},a}(C-1))A_\mathrm{L}$ | $A_\mathrm{S}$ | $(1+w_{\mathrm{L},a}(C-1))A_\mathrm{L}$ |
| $\mathbf{K}_b^+$ | $A_\mathrm{S}$ | $(1+w_{\mathrm{L},b}(C-1))A_\mathrm{L}$ | $(\varphi+1)A_\mathrm{S}$ | $(\varphi+1)(1+w_{\mathrm{L},b}(C-1))A_\mathrm{L}$ |

**Figure 7.8**: Second order multi-class sampling interaction matrix for two classes $a, b$ and two object sizes $A_\mathrm{S}$, $A_\mathrm{L}$.

As an example for an interaction matrix, assume a setup with two classes $a$, $b$ of objects and two possible object sizes $A_\mathrm{S}$ and $A_\mathrm{L}$. The arising interaction matrix is shown in Figure 7.8. Its block structure is clearly visible. As before, $w_{L,a}$ and $w_{L,b}$ describe the fraction of area covered by large particles within classes $a$ and $b$, respectively. The two parameters $\varphi$ and $C$ can be freely chosen, and steer priorities of one sub-class of objects over another. As for second order screening, the constant $C$ describes a priority of large particles over smaller ones. Even more than it is for second order screening, its choice depends on artistic or application-dependent arguments rather than on measurable concepts. A rising series such as the powers of two again gives very good results. However, because for multi-class sampling, all objects interact with all other objects, $C$ now implicitly also controls the priority of large objects of one class over small objects of another. We see examples for this case in Section 7.7.3. The choice of the reflexive weight $\varphi$ follows the same motivation as for colour halftoning. If it is chosen close to zero, the total density is well approximated while the densities for each class are badly represented. If it is chosen close to infinity, the opposite is the case. Hence, a good compromise is given by $\varphi = 1$. Again, we see examples in Section 7.7.3.

To this end, note that the combination of interaction matrices for second order screening and colour halftoning represents a very powerful framework for all kinds of adaptive sampling and halftoning. Both second order screening and colour halftoning are instances of this flexible model. This observation implies two important consequences:

1. Multi-class multi-size sampling is a **versatile** framework for many applications such as colour halftoning, digital pointillism, second order screening, and multi class sampling. It can easily be adapted and configured for future application cases. Moreover, it is possible and often mandatory to combine this general methodology with other ex-

tensions such as discrete dithering (cf. Section 7.4.1), or jittering (cf. Section 7.4.4). By this, it is well possible to extend the scope of the method into areas such as digital sensor design [Wei10], image-based lighting [KK03], discrete second order screening [LA08], or multi-class blue noise sampling [Wei10].

2. Principle **limitations** from colour halftoning and second order screening carry over to this case, and can even exacerbate each other. In a setup with particles of $n$ classes and $m$ different sizes, $n \cdot m$ types of particles must be distributed such that all of them fulfil multiple optimality constraints — which is in general impossible without allowing for compromises. The more types are available and the less particles are included in each individual type set, the more this issue reduces the quality of the final result.

This finishes our excursion about modifications of the electrostatic halftoning model from Section 7.3 for special purposes. The multitude of extensions presented here gives a good impression of the flexibility of electrostatic halftoning for a whole range of applications, which do not even need to be closely related to the original idea. To this end, the given list of extensions should not be regarded as comprehensive, but should instead be seen as a motivation to develop and introduce even more modifications that carry over the quality of electrostatic halftoning to similar applications.

Let us now focus on some technical aspects of electrostatic halftoning. In the following, we discuss two GPU-based algorithms for this purpose, and go into detail about their special adaptations to obtain a high runtime efficiency.

## 7.5 Direct Summation Algorithm

Electrostatic halftoning is an iterative process which requires an update of all particle locations within each iteration. For each particle $m$ and iteration number $k$, the update is given by (7.23). It consists of two computationally intensive parts:

1. Computation of the **attraction field** based on the input image $u$.
2. Evaluation of the **repulsion** acting between particles.

The repulsive force acting on a particle $m$ depends on the position of all other particles. It changes in every iteration if *any* of the particles move. As it turns out, this does not hold for the attractive forces. They depend on

the position of the particle $m$ itself, and on the image $u$ which is constant over all iterations.

We can benefit from this fact if the image $u$ is sufficiently smooth. By (7.19), this smoothness carries over to the (dense) force field $\{\boldsymbol{F}_m^{(A)}\}_{m\in I_P}$. Hence, we can precompute the force field at a sufficiently fine grid to the beginning of the program run, and interpolate to obtain an approximation at an arbitrary location. Fortunately, this modification does not even introduce an additional error in most cases, because the image $u$ is either already stored in a discrete manner, or it is discretised as soon as attractive forces are evaluated. For discrete images, it thus makes sense to use the same grid as a basis for evaluation. During the optimisation process, these values can then be queried at the surrounding grid point and interpolated by a suitable function. One of the most inexpensive but sufficiently accurate functions for this purpose is a simple (bi-)linear interpolation. In 2-D, it requires only four sampling points on surrounding grid locations. On graphics cards, this operation can even be boosted further by using 2-D textures. They perform bi-linear interpolation on dedicated circuits in hardware and provide 2-D aware caching strategies which allow an efficient access to the required input data. However, before we go into detail about the evaluation of attractive forces during the iterations, let us first discuss the precomputation of the attraction field which takes place at the beginning of the program run.

### 7.5.1 Attraction

A parallel GPU algorithm to compute sums such as the one arising from the discretisation of (7.19) was described by Nyland *et al.* in context of N-Body simulations [Ngu07]. Such systems are frequently applied to simulate interactions between solid bodies when they cannot be measured in the real world. Prominent examples are thermophysical effects between molecules which occur on too small spatial scales, or gravitational interactions between celestial bodies that live on too large temporal scales. Because N-Body simulations work with virtually any underlying potential function and since they are often highly consistent with real processes, they are frequently used in many areas of research such as physics, chemistry, and computer graphics.

It seems natural that a modified version of the direct summation algorithm of Nyland *et al.* works well for electrostatic halftoning. Although this algorithm does not provide an optimal runtime complexity – we discuss a more efficient algorithm in Section 7.6 – it is still worth a detailed analysis. Direct summation is among the simplest algorithms one can find for this purpose, which makes it much more flexible and versatile with respect to future extensions or modifications than other approaches. In many

more advanced algorithms such as the one described in Section 7.6, direct summation also reoccurs as one small component such that it is important to understand this process in detail. Also, the computational overhead induced by more complex algorithms is often very high such that small images can typically still be faster processed by a direct summation algorithm. We see examples for this issue in Section 7.7.4.

In order to set up the force field for attraction, we recall that attractive forces only depend on the location and the charge of one particle, but do not depend on other particles in the neighbourhood. This means that we can evaluate the individual forces acting on a particle in each grid location independently from each other. This can even happen in parallel. To this end, we assume unit test charges in all grid positions

$$\boldsymbol{g} \in \Gamma = \{0, \dots, n_x - 1\} \times \{0, \dots, n_y - 1\} = \Omega \cap \mathbb{N}^2 \,, \tag{7.40}$$

and evaluate their interaction with the image. In order to approximate the integral from (7.19), we sample the charge density $1 - u(\boldsymbol{x})$ at the grid $\Gamma$. On discrete images, this grid can either coincide with the original grid of the image, or any resampled version thereof obtained by a rescaling of the image. This leads to a discrete force

$$\left[\boldsymbol{F}_{\boldsymbol{p}}^{(A)}\right] = \sum_{\substack{\boldsymbol{g} \in \Gamma \\ \boldsymbol{g} \neq \boldsymbol{p}}} \frac{1 - u(\boldsymbol{g})}{\|\boldsymbol{g} - \boldsymbol{p}\|} \boldsymbol{e}_{\boldsymbol{p},\boldsymbol{g}} \,, \tag{7.41}$$

where $[\cdot]$ denotes a discretisation of the argument. In the same spirit as before, $\boldsymbol{e}_{\boldsymbol{p},\boldsymbol{g}}$ is the unit vector from $\boldsymbol{p}$ to $\boldsymbol{g}$. One important aspect in this context is the special treatment $\boldsymbol{g} \neq \boldsymbol{p}$. Different to the continuous case (cf. Theorem 7.2), the discrete sum is unbounded, such that it would approach infinity. By this correction, we exclude this case, but can potentially obtain a higher discretisation error.

As it turns out, an adapted version of the direct summation approach by Nyland *et al.* can also be applied to compute the sum from (7.41). To this end, let us consider all pair-wise forces

$$\left[\widehat{\boldsymbol{F}}_{i,j}^{(A)}\right] = \begin{cases} \frac{1-u\left(\binom{x_i}{y_i}\right)}{\left\|\binom{x_i}{y_i} - \binom{x_j}{y_j}\right\|^2} \left(\binom{x_i}{y_i} - \binom{x_j}{y_j}\right), & i \neq j \\ \boldsymbol{0}, & \text{else} \end{cases} , \tag{7.42}$$

where $i, j \in \{0, \dots, n-1\}$ and $n := n_x \cdot n_y = |\Gamma|$. The coordinates $x_i, y_i$ of the grid point $i$ can be immediately computed, assuming that the grid points are enumerated in row-first or column-first ordering. Following the

canonical ordering in memory, we assume in the following a row-first ordering, i.e. $i \in \{0, \ldots, n_x - 1\}$ describes the first row, $i \in \{n_x, \ldots, 2n_x - 1\}$ the second row, and so on. Using the pairwise forces from (7.42), we set up a tensor

$$\mathsf{F}^{(A)} = \left[ \widehat{\boldsymbol{F}}_{i,j}^{(A)} \right]_{i,j} . \tag{7.43}$$

Every column of $\mathsf{F}^{(A)}$ describes the contribution of one grid point to the attraction. In turn, each row represents one test charge that is influenced by the grid. As a consequence, $\left[ \boldsymbol{F}_{\boldsymbol{p}}^{(A)} \right]$ can be computed as the sum over the respective row of $\mathsf{F}^{(A)}$:

$$\left[ \boldsymbol{F}_{(x_j, y_j)^\top}^{(A)} \right] = \sum_{i=0}^{n} \left[ \widehat{\boldsymbol{F}}_{i,j}^{(A)} \right]_j . \tag{7.44}$$

Since the rows are independent of each other, they can be computed in parallel. On a GPU, this option does not only exploit the full data parallelism of the device, but can even be applied to re-use data that was already computed or read from global memory [Ngu07].

To this end, we decompose $\mathsf{F}^{(A)}$ into horizontal bands, where each band is integrated by one CUDA block that consists of a one-dimensional thread grid. The integration result lies in shared memory and needs only be written once when the full band is processed. However, since all threads read the same data at the same time, we can also have the threads cooperating in computing the coordinates $x_j$ and $y_j$. To do so, we split the bands into strides. During integration, all threads first jointly compute all potentials $x_j$ and $y_j$ within one stride into shared memory. After a synchronisation step, they then evaluate 'their' integration over the distances to all other points in shared memory, and weight them with the grey value of the image as obtained by a texture fetch. Since texturing is cached and many threads access the same grey value at the same time, this access is extremely fast.

Figure 7.9 shows a graphical visualisation of this algorithm. For $n$ grid points and $d$ threads per band, $n$ global writes and at most $nd$ global memory reads for texture fetches are required. Moreover, this setup is also tailored to graphics cards when it comes to structural properties of the algorithm. By accessing the input image in regular patterns, considerably more texture cache hits occur. If we additionally set $d$ to a multiple of the internal warp size of the device, there are also no branch divergences taking place. This property holds independent of the fact that we perform 8-fold loop unrolling within each stride for an additional gain in performance. To this end, we obtain a highly efficient direct summation algorithm for elec-

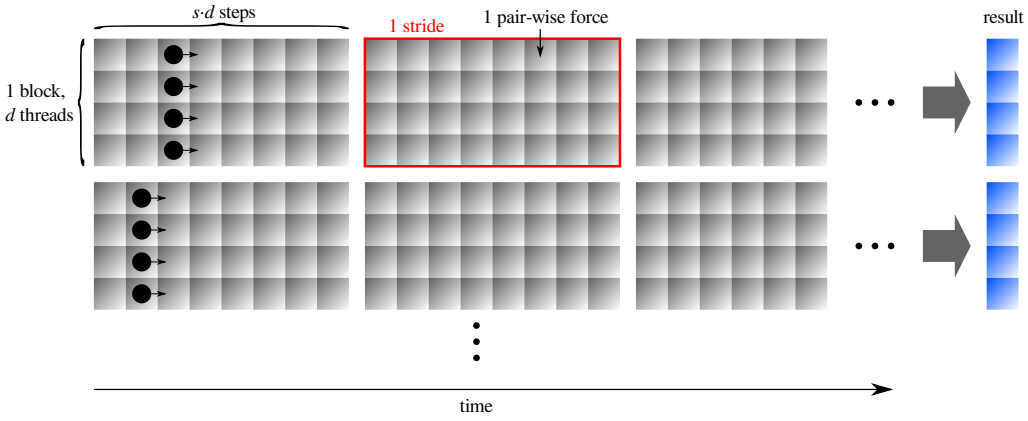



**Figure 7.9**: Integration over the rows of $\mathsf{F}^{(A)}$ or $\mathsf{F}^{(R)}$ similar to proposed by Nyland *et al.*. Every stride is computed on shared memory.

trostatic halftoning, which is even more efficient than the original algorithm by Nyland *et al.*

Once the forces at all grid points have been integrated, the resulting global memory buffer is ready to be used as a lookup table during the evolution of the particle system. Because these forces remain constant from this point on and because we need a fast interpolation to obtain an attractive force at arbitrary points in the image, a texture is the best choice for this purpose. By attaching a texture reference to the global memory buffer, the attractive forces can immediately be evaluated by the particle system without the need to copy data in memory. Forces are returned as `float2` vectors using bilinear interpolation in hardware.

### 7.5.2 Repulsion

Let us now discuss the fast parallel evaluation of the repulsive forces $\boldsymbol{F}_m^{(R)}$ from (7.18). This sum is very similar in structure to the one for $\boldsymbol{F}_{\boldsymbol{p}}^{(A)}$ as in (7.41). However, while there is no variable weight such as the image $u$ involved, the positions of the interaction partners can no longer be computed. This means we must maintain a vector of coordinates at iteration $k$ in global memory. Still, we can use an instance of the algorithm of Nyland *et al.* to compute the repulsion between all particles. The separation into bands and strides is even more important for a good runtime than before. Because loads from global memory are expensive, we load all actors that are relevant for one stride into the fast shared memory. This makes the algorithm very efficient.

In order to give us a good intuition of how the algorithm works, let us

construct a tensor $\mathsf{F}^{(R)}$ in analogy to $\mathsf{F}^{(A)}$ from (7.43). This is straightforward by a similar argument as in in (7.41) and (7.42). During the program run, all threads first load the pair of coordinates corresponding to the particle they compute the integration for, and keep it until the band is entirely processed. Since the length of one stride is chosen as a multiple of the number of threads per band, all threads are equally occupied. After a synchronisation flag, each thread then processes one row within this stride, which is can be accomplished by only accessing the fast shared memory. During this process, all load and store operations are free of bank conflicts, which causes a high efficiency of the algorithm.

To this end, the computations necessary to evaluate the interaction between a pair of particles are not the limiting factor for the runtime of the process. Instead, the high bandwidth to global memory turns out to be a critical aspect that the CUDA kernel must be optimised for. Considering $d$ threads per band and a total number of $M := |P|$ particles, the algorithm requires $M$ reads and writes for 'own' particles and $M\lceil\frac{M}{d}\rceil$ reads for the 'other' particles. All these operations are highly coalesced, and can thus be performed very fast. The larger $d$ is chosen, the more operations are solely computed on fast shared memory. Moreover, it is always beneficial to lay out strides with a larger width than height, i.e. to have $s > 1$ (see Figure 7.9). Such choice helps to mask access latencies to global memory and to gain performance from strategies such as loop unrolling. Hence, the larger and the more 'asymmetric' a stride is, the better becomes the runtime of the algorithm.

However, strides can also not become arbitrarily large. In order to perform all computations on shared memory, each stride must be represented by one CUDA block. Hence, one streaming multiprocessor must be able provide the whole shared memory requirements of one stride. These are two `float2` vectors of length $d$ for the 'own' position and the result, as well as one `float2` vector of length $sd$ for the 'remote' positions. On an NVidia GTX 480, a configuration of $d = 128$ threads per block, and a stride-width of $sd = 256$ elements turned out to be the fastest choice. This choice even leaves space for the default fraction of shared memory being used as a cache to global memory. Whenever two concurrent blocks on this Fermi architecture request the same input data (which is likely to occur in a truly parallel execution of blocks), all second requests can be served much more quickly than the first one.

### 7.5.3 Transporting Particles

Up to this point, we already know how to compute the two summands from the update equation (7.23). In each iteration, the attraction term is read out of a texture (cf. Section 7.5.1), while the repulsion is computed directly (cf. Section 7.5.2). It remains to add both contributions for all particles, to weight them with the time step size $\tau$, and to transport them to their new location.

In this context, it is also important that we robustify the process with respect to numerical instabilities. Because two particles can lie close to each other, their repulsion may become very large. In this case, our model would transport particles by an arbitrary distance into an arbitrary direction – which is seldomly desired. Thus, we limit the maximal displacement by a reasonable upper bound, and additionally forbid particles to leave the image domain by projecting them back to the closest boundary point. As a maximal step size, the side length of one pixel seems to be a good choice. Note that the projection is not required from a technical perspective, because CUDA textures can be requested outside the defined domain, where they return the value from the closest boundary point. However, doing so can tremendously speed up the process. In the desired solution, all particles are supposed to lie inside the domain anyway, such that this additional constraint leads to a much faster convergence.

Compared to the pure repulsion, performing the update still has a computationally low complexity. Assume a number of $M = |P|$ pixels. While repulsion possesses in the runtime complexity $\mathcal{O}(M^2)$, the update can be computed in $\mathcal{O}(M)$ and needs only a few additions, multiplications, and comparisons per particle. Still, a standalone implementation of this operator is expensive, as it involves $3M$ reads and $M$ writes to global memory. The process can thus be significantly accelerated if this operation is integrated into the repulsion kernel from Section 7.5.2. It is executed as a last step after all repulsion forces are computed, and before they are written back to global memory. This modification saves $M$ reads and $M$ writes, because the repulsive force can be re-used immediately from shared memory. Only the computed new position is written back to global memory. The remaining load of the old positions, as well as the texture fetch for attractive forces, are masked behind the arithmetic operations performed by the repulsion kernel.

This process causes the positions to be updated on-the-fly, thus making them invalid for being used in repulsion computations that are still taking place. As a remedy, we perform a double buffering approach in global memory. While the old positions are still stored in one vector, the new

positions are written into a second one. In the next iteration, the role of the two vectors are exchanged by a pointer swap. To the very end of the program run, the most currently written vector is downloaded by the CPU. Hence, this small modification does not introduce any additional computations, but makes the algorithm thread-safe.

### 7.5.4 Additional Features

So far, we have only seen the implementation of the basic algorithm that sets up on (7.23). However, there are many additional features such as the shaking step to avoid local minima, or the extensions from Section 7.4.

The shaking procedure presented in Section 7.3.3 deserves special attention, as it requires a pseudo random number generator (PRNG). Simple algorithms for this purpose are included in the C Standard Library and can thus be used by any CPU program. However, such algorithms are by construction sequential. This might be a reason why a similar implementation was not 'officially' available for CUDA for a long time. In the literature, one finds several approaches to generate numbers with different degrees of statistic randomness, but also with varying runtimes [MS00, Pod07, Ngu07, vMAF+08]. However, for the application to electrostatic halftoning, we only require that pseudo random numbers applied to a certain neighbourhood of particles are 'fairly well' distributed. Thus, a very simple implementation suffices. Recently, NVidia published `cuRand`, an official random number generator [NVi10b]. It is about four times faster than the simple PRNG presented in [vMAF+08] and does not produce significantly worse random numbers. To this end, is is very well suited for 'shaking' the particle vector each 10 iterations.

Extensions are even easier to implement. Many of them, like point size adjustment, grey value correction, or edge enhancement do not affect the CUDA kernels at all, but are applied to the input parameters at the beginning of the program run. Thus, they can be implemented on the host side and do not affect the runtime of the program considerably. All remaining extensions can be implemented with a few lines of optional code in the CUDA kernels, each. While dithering comes down to one additional interaction term and one projection per particle, the perturbation field for jittering can directly be read out of a texture that has been precomputed at the beginning of the program run. Multi-colour halftoning, second order screening, and multi-class blue noise sampling require an additional vector containing the class assignment of each particle. It is loaded along with the positions, and determines if particles interact with each other at all and, if yes, by how much.

## 7.6 Fast Summation Algorithm

The direct summation approach presented in the previous section is very flexible and relatively easy to implement but suffers from a quadratic runtime complexity in the number of particles. This property makes it inapplicable to images where a large number of particles is used.

The slow runtime of this algorithm arises from the fact that all particles must communicate with each other before a direction vector can be computed that describes the offset for the next iteration step. Although this argument is justified from an information theoretic consideration, it somehow contradicts our intuition. Because the potential around a particle drops off rapidly, nearby particles experience a much stronger repulsion than distant ones. To this end, a particle's neighbourhood is very important for the accurate computation of the direction vector, while distant particles only have a significant influence if they are part of a large dominant cluster. This observation is the key to a whole range of fast approximative algorithms which enjoy a lower runtime complexity such as $\mathcal{O}(M \log M)$ in the number of particles $M$.

The first method in this spirit was proposed by Hockney and Eastwood [HE81]. It performs a decomposition of the domain into cells and assigns the charge density of each cell to a virtual super-particle in its centre. The effects on any particle can efficiently be computed in the frequency domain and, by the use of fast Fourier transforms (FFTs) [CT65], even the transformation can be realised in a short amount of time. This approach requires a sufficiently flat potential field at any point and is thus well suited if particles are far apart. In order to extend the method to interactions on all distances, particles thus interact directly (particle-particle) if they lie close together, and indirectly (particle-mesh) otherwise. Hence, this approach is often called the particle-particle/particle-mesh ($P^3M$) method.

A more simple yet efficient method was introduced by Barnes and Hut [App85, BH86]. It decomposes the image into a quadtree. Similar to $P^3M$ methods, all cells of this tree offer the capacity or mass of its children as one accumulated value in the centre of mass. Nodes can thus interact with individual leave nodes in their direct neighbourhoods, and use approximative values from a higher level in the tree if the respective clusters are far enough away. Because this interaction takes place in the spatial domain and discards the local structure of the point clouds, it only has an accuracy of up to about 99%. Still, it is frequently used in astrophysical simulations due to its simplicity and computational efficiency.

Since the 1980s, a new class of algorithms rose in popularity. As it is for the aforementioned methods, such techniques possess a typical runtime

complexity of $\mathcal{O}(M \log M)$ in the number of particles $M$, which can even drop to $\mathcal{O}(M)$ for potentials with special characteristics. Unlike the previous approaches, however, these techniques can approximate the optimal solution up to an arbitrary accuracy. For this purpose, they possess a set of parameters which steer a tradeoff between runtime and quality. As the most prominent member of this class, the *fast multipole method* is widely used today for all kinds of N-Body simulations such as celestial or molecular dynamics [GR87]. Within a hierarchic decomposition of the image, it computes interactions between distant particles by means of multipole expansions. This interaction can also be interpreted as the multiplication of a vector of particles with a huge interaction matrix. Compared to a direct summation method which possesses a dense operator, the fast multipole method comes down to a sparsification of this operator [SP01]. Moreover, this notation immediately reveals the data-parallel character of this operation. It has been exploited for a variety of algorithms for massively parallel hardware, such as the Intel Paragon [KK95], or GPUs [GD08].

In [FS02], the authors show that, apart from application-dependent details, a number of other methods is closely related or even equivalent to this technique. Similar ideas, also arising from the field of particle simulations, are the so-called *fast mosaic-skeleton approximations* [Tyr96] and the *fast $\mathcal{H}$-matrix multiplications* [Hac99]. Other approaches are closely related, but differ in small details. For kernels whose analytical properties are unknown, the authors of [BCR91] propose to compute interactions in the wavelet domain. In [BC03], another method is presented which sets up on a double-step design. While the first step describes a fast approximation of interactions in the Fourier domain, the second step involves a correction within a small local neighbourhood to obtain a higher approximation quality where the underlying potential is large.

For electrostatic halftoning, we use a related technique which reduces expensive pairwise interactions to cheap multiplications in the frequency domain [FS04]. It sets up on the idea that the force on one particle is just given by the sum of (radial) potentials around all other particles, evaluated at its location. To this end, we can perform a convolution of a signal which contains Dirac peaks at the positions of all particles. By the convolution theorem (cf. (3.25) in Chapter 3), this operation can efficiently be performed in the frequency domain where it comes down to a point-wise multiplication. However, since the positions of the particles do not reside on a regular grid, we require *non-equispaced fast Fourier transforms* (NFFTs) for this purpose [PST00]. As it is for standard FFTs, they enjoy a runtime complexity of $\mathcal{O}(M \log M)$, but are algorithmically more challenging.

On the next few pages, we develop a novel efficient GPU-based algorithm

for the NFFT-based fast summation method. This work is also published in [GSWT11]. The theoretical foundations of this work, as well as a CPU-based prototype, can be found in [TSG+11]. One major ingredient of this algorithm is the design of a new GPU-based algorithm for the NFFT which is better applicable to electrostatic halftoning than previous approaches from the literature [SSNH08, Gre08, GD08]. Later in this chapter, we go more into detail about this issue. Because our new NFFT implementation is very versatile and does not make special assumptions on the underlying structure of the problem, it is also very universal and can be applied in many other applications beyond halftoning.

### 7.6.1 Repulsion by Fast Summation

Let us first address the efficient computation of the repulsive forces in electrostatic halftoning. Following the fast summation technique from [FS04], we rewrite the repulsive forces $\boldsymbol{F}_m^{(R)}$ from (7.18) such that they can be computed by means of NFFTs. Later, we can use these insights to perform a similar acceleration to precompute *attractive* forces much more efficiently than before. For simplicity, let us assume that $q_m = q_n$. This allows to normalise the constant $kq_mq_n$ to 1 and obtain

$$\boldsymbol{F}_m^{(R)} = -\sum_{\substack{n\in I_P\\ n\neq m}} \frac{1}{\|\boldsymbol{p}_n - \boldsymbol{p}_m\|}\boldsymbol{e}_{m,n}\ . \tag{7.45}$$

This sum can be decomposed into three sums with scalar-valued arguments [FS04, TSG+11]:

$$\begin{aligned}\boldsymbol{F}_m^{(R)} &= -\sum_{\substack{n\in I_P\\ n\neq m}} \frac{\boldsymbol{p}_n - \boldsymbol{p}_m}{\|\boldsymbol{p}_n - \boldsymbol{p}_m\|^2}\\ &= \boldsymbol{p}_m \sum_{\substack{n\in I_P\\ n\neq m}} \frac{1}{\|\boldsymbol{p}_n-\boldsymbol{p}_m\|^2} - \begin{pmatrix}1\\0\end{pmatrix} \sum_{\substack{n\in I_P\\ n\neq m}} \frac{\boldsymbol{p}_{n,x}}{\|\boldsymbol{p}_n-\boldsymbol{p}_m\|^2} - \begin{pmatrix}0\\1\end{pmatrix} \sum_{\substack{n\in I_P\\ n\neq m}} \frac{\boldsymbol{p}_{n,y}}{\|\boldsymbol{p}_n-\boldsymbol{p}_m\|^2}\end{aligned} \tag{7.46}$$

Each of these sums describes a discrete convolution of the form

$$\sum_{n\in I_P} \gamma(n) K(\|\boldsymbol{p}_n - \boldsymbol{p}_m\|) \tag{7.47}$$

where the vector $\gamma$ takes either of the values $\gamma(n) \equiv \mathbf{1}$, $\gamma(n) = \boldsymbol{p}_{n,x}$, or $\gamma(n) = \boldsymbol{p}_{n,y}$. $K(x)$ is a radial kernel

$$K(x) = \frac{1}{x^2} \tag{7.48}$$

with $x = \|\boldsymbol{p}_n - \boldsymbol{p}_m\|$. In order to perform these summations very fast, we transform $\gamma$ and $K$ into the frequency domain. By the convolution theorem, their convolution can then be computed as a point-wise multiplication of their Fourier coefficients. To this end, we scale the image domain such that it fits into the circle with radius $\frac{1-\varepsilon_B}{4}$ around $(0,0)^\top$, where $\varepsilon_B$ is a small constant with $0 < \varepsilon_B \ll 1$. In order to make $K$ smooth, we regularise it near 0 and $\pm\frac{1}{2}$ in each dimension such that we obtain a 1-periodic kernel

$$\mathcal{K}_R(x) = \begin{cases} K_I(|x|), & |x| < \varepsilon_I \\ K(|x|), & \varepsilon_I \leqslant |x| < \dfrac{1-\varepsilon_B}{2} \\ K_B(|x|), & \dfrac{1-\varepsilon_B}{2} \leqslant |x| < \dfrac{1}{2} \\ K_B(\dfrac{1}{2}), & \dfrac{1}{2} \leqslant |x| \end{cases} . \tag{7.49}$$

Note that we write $\mathcal{K}_R(x)$ in calligraphic letters to underline its approximative character with respect to $K$. In this context, $\varepsilon_I$ denotes a small positive constant on which we detail later in this section. As suggested in [FS04], both $K_I$ and $K_B$ can be obtained by a two-point Taylor interpolation with polynomials of degree $\hat{p}$. The parameter $\hat{p}$ steers the quality of the approximation: the higher, the better. Finally, we can approximate $\mathcal{K}_R$ by its truncated Fourier series $\mathcal{K}_R$

$$\mathcal{K}_R(x) \approx \mathcal{K}_F(x) = \sum_{\boldsymbol{j} \in I_N} b_{\boldsymbol{j}} e^{2\pi i \langle \boldsymbol{j}, \boldsymbol{x} \rangle}, \tag{7.50}$$

with

$$b_{\boldsymbol{j}} = \frac{1}{N^2} \sum_{\boldsymbol{k} \in I_N} \mathcal{K}_R\left(\frac{k}{N}\right) e^{-2\pi i \frac{\langle \boldsymbol{j}, \boldsymbol{k} \rangle}{N}} . \tag{7.51}$$

The size of the plane $I_N$ in the frequency domain has an immediate influence on the approximation error between $\mathcal{K}_R$ and $\mathcal{K}_F$: the larger $N$, the better the approximation. In accordance with [TSG+11], we choose the same resolution in time and frequency domain, i.e. $N \approx \max(n_x, n_y)$. From an information-theoretic standpoint, this choice allows that all information can be recovered after transformations to and from frequency space. Moreover, since we can assume that $M \sim n_x \cdot n_y$ and that $\hat{p}$ constant, it holds that $N \sim \sqrt{\hat{p}M}$ as required in [FS04].

The new kernel $\mathcal{K}_F$ is small below $\varepsilon_I$, and close to $K$ if the argument lies above this threshold. The case $x > \frac{1-\varepsilon_B}{2}$ never occurs, because the image domain is scaled such that it fits in a circle with radius $\frac{1-\varepsilon_B}{4}$ (see above).

Hence, the largest distance between two particles is by construction the diagonal of the image plane $x = \frac{1-\varepsilon_B}{2}$.

On the other hand, the difference $\mathcal{K}_N = K - \mathcal{K}_R$ has finite support, since $\mathcal{K}_N(x) = 0$ for $x > \varepsilon_I$. This observation allows to approximate the kernel $K$ by

$$K \approx \mathcal{K}_F + \mathcal{K}_N \,, \tag{7.52}$$

where $\mathcal{K}_N$ describes *only* local interactions between particles, and $\mathcal{K}_F$ is dominated by global interactions. Because of the shape of $K$, the forces occurring in the near-field kernel $\mathcal{K}_N$ are much higher than the ones in the far-field kernel.

As a consequence, we require a high accuracy in the near-field but can accept an approximate solution for far-field interactions. Thus, we compute near-field interactions by a direct summation approach, while we perform a frequency-based convolution for the far-field contributions. This leads to the following two steps for either sum from (7.46):

- **Far-field interactions.** Let us first consider the far-field contributions for the vectors $\gamma$ in the discrete convolution from (7.47). For every $\gamma$, we obtain the double sum

$$\begin{aligned} &\sum_{n \in I_P} \gamma(n)\ \mathcal{K}_F(\|\boldsymbol{p}_n - \boldsymbol{p}_m\|) \\ &\overset{(7.50)}{=} \sum_{\boldsymbol{j} \in I_N} b_{\boldsymbol{j}} \left( \sum_{n \in I_P} \gamma(n)\ e^{2\pi i \langle \boldsymbol{j}, \boldsymbol{p}_n \rangle} \right) e^{-2\pi i \langle \boldsymbol{j}, \boldsymbol{p}_m \rangle} \,. \end{aligned} \tag{7.53}$$

  While the outer term of (7.53) has the form of a forward Fourier transform, the inner term describes an inverse transform. However, we cannot use standard FFTs to efficiently compute these Fourier transforms, since the arguments $\langle \boldsymbol{j}, \boldsymbol{p}_m \rangle$ and $\langle \boldsymbol{j}, \boldsymbol{p}_n \rangle$ do not reside on a regular grid. As proposed in the literature, we solve this problem by applying a non-equispaced fast Fourier transform (NFFT) [PST00, FS04].

  To this end, we compute (7.53) in three steps [FS04]:

  1. Transformation of $\gamma(n)$ with an *adjoint* ("backwards") NFFT with negated source nodes $\boldsymbol{p}$.
  2. Point-wise multiplication of the results with the precomputed kernel coefficients $b_j$.
  3. Back-transformation using a *standard* ("forward") NFFT with negated source nodes $\boldsymbol{p}$.

Note that in electrostatic halftoning, source nodes for the NFFTs are given by particle positions which are per definition real-valued. For this particular application, it is thus sufficient to have a real-to-complex valued adjoint NFFT, as well as a complex-to-real valued standard NFFT. This specialisation saves roughly half of the required operations such that implementations benefit both in runtime and memory requirements.

Equivalently, we could also compute two real-valued NFFTs jointly by setting one input vector as the imaginary part of a standard complex-to-complex valued transform. However, doing so for two of the three sums to be computed results in a comparable runtime, but a higher memory consumption. Since memory is a restricted resource in particular when it comes to GPUs, we do not pursue this idea any further.

In the next section, we detail more about the NFFT and its adjoint, and discuss efficient GPU-based algorithms for this purpose.

- **Near-field interactions.** Let us now briefly address the computation of near-field interactions. Because these interactions take place in local neighbourhoods with radius $\varepsilon_I$ around a particle and because the kernel $\mathcal{K}_N$ can be directly computed, this operation comes down to a direct summation as shown in Section 7.5. However, particles are not stored in locality-preserving data structures. This makes an identification of nearest neighbours of a particle nontrivial. Maintaining such a structure would be very expensive, because particles move in every iteration. On the CPU, this problem can easily be solved by binary search trees in which particles are inserted once per iteration, and which can then be used to reduce the number of candidates to a reasonable value [TSG+11]. Unfortunately, such data structures are unsuited for GPUs because they are very memory-intense and scale badly to massively parallel architectures. In Section 7.6.3, we thus develop a fast alternative to this algorithm which does not suffer from the mentioned limitations.

For the sake of completeness, let us briefly come back to our assumption $q_m = q_n$ that we made for (7.45). As it turns out, the same strategies as above carry over to the case $q_m \neq q_n$, i.e. for second order screening. If we have different charges for all particles, we multiply each $\gamma$ point-wise with $q_n$, perform the computation, and weight the result $\boldsymbol{F}_m^{(R)}$ by $q_m$. Additional concepts such as the priorities $C$ (see Figure 7.5) can be incorporated by computing a separate repulsion on the subset of large particles. Since the

partition of large particles is in general very small, such sums are relatively inexpensive compared to the whole process.

### 7.6.2 Non-Equispaced Fast Fourier Transform

The core of our fast summation approach is an efficient GPU implementation of the 2-D non-equispaced Fourier transform (NFFT). In the literature, there are already some GPU implementations for this purpose [SSNH08, Gre08, GD08]. However, neither of these approaches is suited for the type of problem arising in electrostatic halftoning. The reason for this is the highly irregular structure of the vector of locations. Classical parallel approaches assume that nodes are still sorted and aligned in some geometric structure such as special irregular grids, spirals, or other primitives. As we are going to see later in this chapter, the most expensive step of the NFFT involves reading 2-D neighbourhoods of nodes. To accelerate this step, all existing GPU-based algorithms exploit this fact by re-using memory where these neighbourhoods overlap. However, these assumptions do not hold for electrostatic halftoning. Here, particles are freely distributed over the image domain. Even in regions where they cluster, all nodes within one cluster are in general not stored in subsequent memory cells. On the other hand, sorting the vector $\boldsymbol{p}$ prior to computing the NFFT is also no option. Although this operation is in the same complexity class $\mathcal{O}(M \log M)$, it would roughly double the overall runtime due to expensive memory interactions. In the following, we design the first GPU-based algorithm to work efficiently with random particle positions that are not aligned in some special structural arrangement. Because the NFFT nowadays represents a standard tool for many applications, this new algorithm has a broad range of applications beyond halftoning.

Let us start with a short recapitulation of NFFTs. Given a set of nodes $(\boldsymbol{p}_j)_{1 \leqslant j \leqslant M} \subset \Pi^2 = [-\frac{1}{2}, \frac{1}{2}]^2$, we are interested in an operator

$$\boldsymbol{A} := \left(e^{-2\pi i k \boldsymbol{p}_j}\right)_{j=1,\ldots,M;\ k \in I_N} , \tag{7.54}$$

such that

$$\boldsymbol{f} = \boldsymbol{A}\hat{\boldsymbol{f}} , \qquad \boldsymbol{f} \in \mathbb{R}^M , \tag{7.55}$$

is the Fourier transform of $\hat{\boldsymbol{f}}$. $I_N$ denotes the 'shifted' index set in the frequency space, and is given by $I_N := \{k \in \mathbb{Z}^2 | -\frac{N}{2} \leqslant k < \frac{N}{2}\}$. For the computation of electrostatic halftoning, we use the same resolution in the spatial domain and the frequency domain, such that $M = N$. Moreover, we want to compute the adjoint NFFT

$$\overline{\boldsymbol{A}}^{\top} := \left(e^{2\pi i k \boldsymbol{p}_j}\right)_{j=1,\ldots,M;\ k \in I_N} , \tag{7.56}$$

such that

$$\hat{\boldsymbol{h}} = \overline{\boldsymbol{A}}^{\top} \boldsymbol{f} \ . \tag{7.57}$$

One should note that, unlike for the equispaced case, the adjoint NFFT $\overline{\boldsymbol{A}}^{\top}$ does not constitute an inverse to the NFFT $\boldsymbol{A}$. However, $\hat{\boldsymbol{h}}$ represents a good approximation of $\hat{\boldsymbol{f}}$. Hence, we treat it as such for the rest of this chapter, but keep in mind that a pair of transformations $\overline{\boldsymbol{A}}^{\top} \boldsymbol{A}$ can introduce additional approximation errors. As it turns out, they are irrelevant in practice.

In [PST00], the NFFT $\boldsymbol{A}$ from (7.54) is approximated as a composition of a scaling operation $\boldsymbol{D}$, a standard discrete Fourier transform $\boldsymbol{\mathcal{F}}$, and a sampling operation $\boldsymbol{B}$:

$$\boldsymbol{A} \approx \boldsymbol{B}\boldsymbol{\mathcal{F}}\boldsymbol{D} \ . \tag{7.58}$$

As is shown in [KKP09], this approximation holds for transforms in arbitrary dimensions. The complex-valued 2-D Fourier transform $\boldsymbol{\mathcal{F}}$ used in (7.58) is given by

$$\boldsymbol{\mathcal{F}} := \left( \frac{1}{\tilde{n}^2} e^{-2\pi i k l / \tilde{n}} \right)_{k,l \ \in \ I_{\tilde{n}}} \ . \tag{7.59}$$

$I_{\tilde{n}}$ denotes the Fourier plane, which has been oversampled by a factor $2 \leqslant \alpha < 4$, such that $\tilde{n} = \alpha N$ and such that there exists a $k$ with $\tilde{n} = 2^k$. By this property, $\boldsymbol{\mathcal{F}}$ can efficiently be computed by a standard fast Fourier transform, which performs best on data with the edge length of a power of 2 [CT65]. The real-valued sparse matrix

$$\boldsymbol{B} := \left( \psi \left( \boldsymbol{p}_j - \frac{1}{\tilde{n}} \boldsymbol{l} \right) \right)_{j=0,\ldots,M-1; \ \boldsymbol{l} \ \in \ I_{\tilde{n}}^2} \tag{7.60}$$

describes a convolution of the transformed result with a Kaiser-Bessel window function [KS80] $\psi$ that has been truncated at $\tilde{m}$. It accounts for the fact that the nodes are not residing on grid positions, and approximates the contributions to the sparsely distributed nodes. The choice of the cut-off parameter $\tilde{m}$ describes a trade-off between accuracy and speed, and thus steers the approximation quality. Finally, the real-valued 'diagonal' scaling matrix $\boldsymbol{D}$ is given by

$$\boldsymbol{D} := \bigotimes_{t \in \{x,y\}} \left( \boldsymbol{O} \ \middle| \ \operatorname{diag} \left( \frac{1}{c_{k_t}(\varphi)} \right)_{k_t \ \in \ I_N} \ \middle| \ \boldsymbol{O} \right)^{\top} \ . \tag{7.61}$$

In this context, $\boldsymbol{O}$ denote zero matrices of size $N \times (\tilde{n} - N)/2$, while $\varphi$ is the 1-periodic continuation of the Kaiser-Bessel window function on a

torus. The operator $\bigotimes$ denotes the tensor product. Moreover, the Fourier coefficients $c_{k_t}(\varphi)$ are computed as

$$c_{k_t}(\varphi) = \int_{\Pi} \varphi(v) e^{2\pi i k_t v} \, \mathrm{d}v \quad (k_t \in \mathbb{Z}) \; . \tag{7.62}$$

The purpose of $\boldsymbol{D}$ is to correct the roll-off of the Kaiser-Bessel kernel from $\boldsymbol{B}$. To this end, it can be regarded as a normalisation step.

The computation of the adjoint NFFT $\overline{\boldsymbol{A}}^{\top}$ follows the same argumentation. Using the same terminology, we can approximate $\overline{\boldsymbol{A}}^{\top}$ by

$$\overline{\boldsymbol{A}}^{\top} \approx \boldsymbol{D}^{\top} \overline{\boldsymbol{\mathcal{F}}}^{\top} \boldsymbol{B}^{\top} \; . \tag{7.63}$$

The operators $\boldsymbol{D}$ and $\boldsymbol{B}$ are real-valued, such that their adjunct matrices are given by their transposes. In addition, we need the complex-valued inverse 2-D Fourier transform $\overline{\boldsymbol{\mathcal{F}}}^{\top}$. However, because the nodes are again arranged in a regularly spaced grid of size $|I_{\tilde{n}}|$, we can apply a generic inverse FFT at this point.

As it turns out, the choice of $\varepsilon_I$ is a crucial criterion for the approximation quality and the runtime of the process. Because it represents the size of the circles around each particle in which we compute interactions by means of direct summation, we should not make $\varepsilon_I$ too large. If we assume that each circle holds about $k$ particles, this process has a runtime complexity $\mathcal{O}(Mk)$, such that the whole process approaches the direct summation method from Section 7.5 for $\varepsilon_I \to 1/2$. On the other hand, $\varepsilon_I$ should also not be too small because interactions with a high magnitude are then computed by the approximative frequency-based summation technique. A good tradeoff between these extremes seems to be given by $\varepsilon_I = \frac{\hat{p}}{\tilde{n}}$. This choice also fulfils the constraint $\varepsilon_I \sim \frac{\hat{p}}{N}$ from [FS04] (with a factor $\frac{1}{\alpha}$).

Let us now discuss some implementation details of the GPU algorithms for the application of $\boldsymbol{A}$ and $\overline{\boldsymbol{A}}^{\top}$. From the last section, we know that electrostatic halftoning only requires the instances of real-to-complex valued adjoint transforms, as well as complex-to-real valued standard transforms. This insight allows to reduce the bandwidth of the $\boldsymbol{B}$ and $\boldsymbol{B}^{\top}$, which is particularly important because they represent the most expensive part of the algorithm (cf. Section 7.7.6).

For the central operation, the fast Fourier transform $\boldsymbol{\mathcal{F}}$ of length $I_{\tilde{n}}$, we apply the `cuFFT` library from the CUDA toolkit as provided by NVidia [NVi10a]. This decision follows the same arguments as given in context of the frequency-based convolution for linear diffusion (see Section 3.5.4).

The computation of the two diagonal matrices $\boldsymbol{D}$ and $\boldsymbol{D}^\top$ is data-parallel, but highly memory-bound. By using the sparsity of the matrices, every thread can compute one of the $I_N$ point-wise divisions on the main diagonal. However, to account for the tight memory bandwidth, we do not precompute $c_k(\varphi)$ as it was suggested in [PST00]. Instead, we evaluate this function on-the-fly. This is very efficient, in particular because each value can be applied to four values in parallel without re-computing the respective value. By this procedure, memory latencies are well hidden behind computations.

Because of their special memory patterns, the two operators $\boldsymbol{B}$ and $\boldsymbol{B}^\top$ turn out to be the most expensive part of the NFFT on parallel architectures. Both operations describe a convolution. While for the application of $\boldsymbol{B}$, local neighbourhoods of size $(2\tilde{m}+1)^2$ around the positions of all particles must be read, they are written in context of $\boldsymbol{B}^\top$. In the literature, algorithms usually require special arrangements of particles in order to enforce a coherent access to global device memory [SSNH08, Gre08, GD08]. The closer particles lie together, the more their neighbourhoods are overlapping. Data lifetime in on-chip memory is thus being significantly increased such that bandwidth to global memory can be saved. However, these assumptions do not hold for electrostatic halftoning. Particles can reside at random positions and do not necessarily cluster or arrange in regular structures. Hence, we need a strategy which works for arbitrary local relations of particles and which optionally draws advantage from data locality when this is possible. For the computation of $\boldsymbol{B}$, all memory accesses are read-only. This observation suggests the use of the texturing engine of graphics cards as it provides 2-D-aware caches. Whenever a random datum is requested for the first time, we additionally need a whole patch of size $(2\tilde{m}+1)^2$ in the near future. There is a good chance that most of this data is already in the 2-D caches after the first call. Hence, many cache misses can be avoided.

It remains to compute $\boldsymbol{B}^\top$. Here, the memory patterns are more expensive since CUDA knows write caches only from the Fermi architecture on, and random write operations are still very expensive. However, since there is no way to circumvent this problem without requiring special conditions, we can only make sure that coinciding writes from different threads do not interfere. This can be achieved by atomic operations. For older architectures that support compute capability 1.x, such atomic operations exist only for integers. Still, we can make use of the fact that these devices are 32-bit architectures, which means that the data types `int` and `float` are equally large. Thus, we can exchange the value in global memory with 0 by using an integer atomic exchange, reinterpret the result as a float, add it to the increment, and exchange it again with the (potentially new) value

in the global memory cell [Wor08]. If we repeat this process until it returns 0, the increment to be added to the global datum was successfully applied in a thread-safe manner. This looks as follows:

```
while (data)
  data = atomicExch(addr, data + atomicExch(addr, 0.0f));
```

On Fermi architectures with compute capability 2.x, this restriction is no longer present. Here, the above instructions collapse to a single float atomic addition, which again saves a significant amount of memory bandwidth.

### 7.6.3 Near-Field Evaluation

The computation of near-field interactions by using the kernel $\mathcal{K}_N$ is simple from a theoretical perspective, but algorithmically challenging and computationally expensive. Because forces arising from near-field interactions are much stronger than those of the far-field, the near-field requires a much higher computational precision. Hence, only a direct summation approach can be used in this context. At the same time, we want the near-field to be as small as possible. This means, we only want to read and process those particles for which the resulting force on a target particle does not vanish. The reason for this is that, even though an evaluation of the kernel would result in a zero vector for any of these particles, reading their position and performing the computation still requires the same computational burden as for actual near-field interaction partners. Thus, the overall runtime of the process crucially depends on whether potential neighbours of a particle can reliably be identified, or not.

On sequential hardware, tree structures such as k-d-trees provide a good solution to this problem [Ben75, TSG+11]. The runtime for traversing either of these structures has a logarithmic complexity with little overhead, and almost vanishes in the complete process. However, such structures are usually no option for parallel architectures. This is because traversals of a tree involve branching on all levels, which contradicts the constraint that GPUs require whole warps to take the same branches at a time. Likewise, because these threads potentially descend into different branches of the tree, memory access patterns become more complicated the deeper the level. Thus, we require a data structure that respects and supports the parallel nature of our algorithm.

The key observation leading to our new data structure is the electric neutrality of regions as shown in Theorem 7.3. Leaving aside numerical issues such as shaking, we can expect each region to contain at most the number of particles which correspond to plain black. It thus seems justified

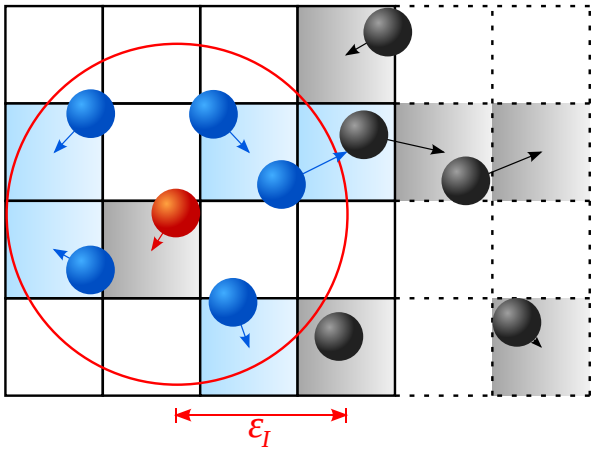



**Figure 7.10**: Neighbourhood in a lookup map for the near-field of a particle (red): Neighbours (blue) and false positives (black). Arrows indicate mappings.

to allocate a 2-D hash map in the size of the image domain which contains the exact coordinates of the included particle, or an empty pair if the cell is empty. Note that if two particles are close to each other, our simple hash function assigns two particles to the same cell. However, this does also not pose a problem to the algorithm. There are at least as many cells as particles available, such that there is always a cell in the near vicinity which can still consume a particle. Hence, we are only required to increase the search radius slightly, as the entry for particle might be dislocated by a few cells from the actual position of the particle.

Figure 7.10 shows a sample neighbourhood as arising from this strategy. As a reservoir for a potential relocation of particles, we reserve a small stripe to the right of a neighbourhood. This is denoted with dashed lines. All candidates for interaction partners of a particle are thus required to be found in a search window that covers the circular area with radius $\varepsilon_I$ around this particle, plus the attached small stripe to the right. This strategy allows to obtain all neighbours of a particle with little effort while the number of false positives, and thus the computational expense, is kept small.

In CUDA, these lookups can most efficiently be realised by texture fetches. By the 2-D aware caches of the texturing unit, cache misses are reduced to a minimum, and boundaries are handled automatically. Still, we must take care that the hash map also possesses a surplus stripe at the right side such that particles are allowed to be slightly pushed off the map. The empty pair is denoted by the bit string $(1^{64})_2$, i.e. by a series of 64 binary 'ones'. This corresponds to a pair of two not-a-number entries (NaNs). On traditional platforms, such assignment would cause performance problems,

since NaNs take longer to be consumed by the processor than actual values. However, NaNs on a GPU are non-signalling, which means our choice does not impair the runtime of the algorithm. Instead, it allows for a quick initialisation of the whole map by one call to `cudaMemset`. Note that $(1^{64})_2$ is also the only pattern which is both easy to initialise and unambiguous, since $(0^{64})_2$ is indistinguishable to an actual particle residing in $(0,0)^\top$.

During the nearfield evaluation, we look up all entries in the search window, check if they represent a candidate, and evaluate if they are a neighbour to be considered. For all found neighbours, we then compute the arising force on the particle in question, and accumulate the force increments in shared memory. This algorithm can process a large number of particles in parallel, and scales well over all streaming multiprocessors of the graphics card. Only once per iteration, this force is added to the respective far-field contributions and written to global device memory which again saves memory bandwidth.

Finally, let us discuss the construction of the map. It changes with every iteration, such that its set-up phase must be highly efficient as well. To the same time, it must pay attention to special cases such as multiple assignments of particles to the same cell. In order to realise this process as a parallel algorithm, we thus use an idea inspired by cuckoo hashing [PR04]. To this end, each particle is inserted into its designated cell, regardless of whether this cell already contains particles or not. Potential residents are pushed to the right side, until the process converges. In CUDA, this process can be modelled by an atomic exchange operation. Beginning at the target cell and advancing to the right, the register initially containing the new particle is exchanged with the content of the memory cell in charge, until the process returns an empty pair. Because there are no atomic exchange operations for **`float2`** data types, we can use the `atomicExch` operation for `unsigned long long` types, as it works on bit strings of the same length. Assuming the local **`float2`** entry is called `d` and the designated cell has address `addr`, we obtain:

```
while (d != 0xFFFFFFFFFFFFFFFF)
{
  d = atomicExch((unsigned long long*)addr, d);
  addr++;
}
```

Finally, let us briefly analyse the width of the surplus stripe at the right side, which constitutes a trade-off between efficiency and the ability to process a larger number of particles. As stated before, it depends on the fill level of the image with particles, as well as on the degree to which the electric

neutrality of regions is preserved over the program run. As it turns out, the latter property is easily impaired if we activate shaking (cf. Section 7.5.4). Surprisingly however, an additionally boundary of 4 cells already suffices for most images, if we start with an initialisation as in Section 7.3.3. Even if this boundary does not suffice to the beginning of the process, experiments show that the evolution still continues as desired.

### 7.6.4 Attraction

Now that we can compute repulsive forces efficiently, it remains to compute the attractive forces from (7.23). As for the direct summation approach, it suffices to compute them once at the beginning of the program run, and to retrieve them from a CUDA texture during the particle evolution. Since the precomputation has the same problem description as in the direct summation approach, we could use a similar algorithm as presented in Section 7.5.1.

However, given the theory developed in the previous section, this procedure is not optimal if the image is very large. Attractive forces are of the same form as repulsive ones, with the additional constraint that all interaction partners now reside on a regular grid. This allows us to split the sum $\left[\boldsymbol{F}_{\boldsymbol{p}}^{(A)}\right]$ from (7.41) in a similar fashion as in (7.46). Since we can now even apply standard FFTs rather than NFFTs, the fast summation is no longer subject to approximation errors. Hence, there is no necessity to distinguish between near-field and far-field interaction but we can use the fast frequency-based approach for the full domain. To this end, the precomputation of the force field consists of the following steps:

1. **Forward FFTs** to transform the vectors $(1 - u(\boldsymbol{g}))$, $\boldsymbol{g}_x$, and $\boldsymbol{g}_y$ into the frequency domain.

2. **Point-wise multiplication** of the coefficients from 1. with the Fourier coefficients of the kernel $K$.

3. **Inverse FFTs** to transform the modified coefficients back.

4. **Assembly** of $\left[\boldsymbol{F}_{\boldsymbol{p}}^{(A)}\right]$ and linking of the CUDA texture.

In Section 7.7.4, we perform a detailed comparison of the fast summation algorithm on the GPU with the previously presented direct summation approach, as well as with the corresponding CPU algorithms.

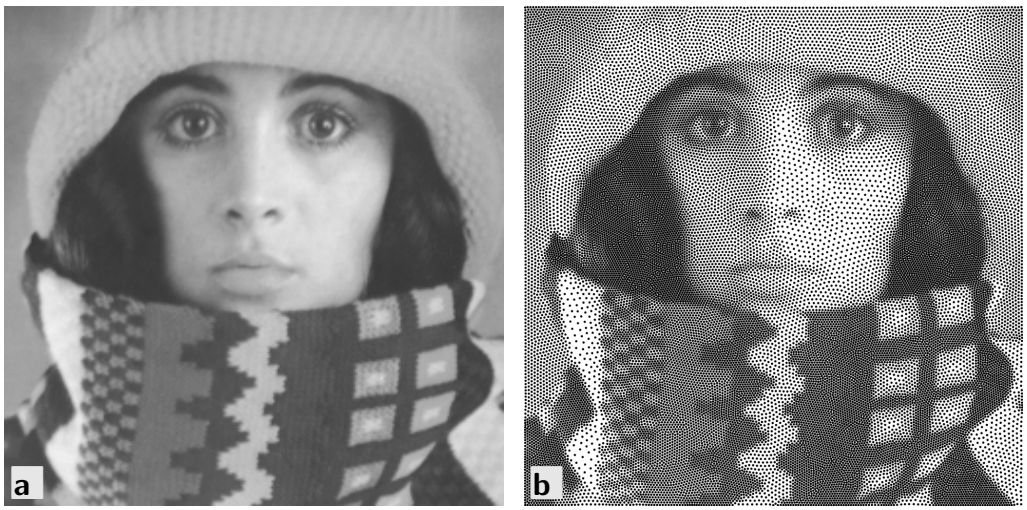



**Figure 7.11**: Halftoning with 30 150 dots on *Trui*, 256×256 pixels. This image serves as a basis of comparison in the following chapters. **a.** Original. **b.** Electrostatic halftone.

## 7.7 Experiments

Let us now evaluate the quality of the halftones produced by our algorithms with all their extensions and modifications, as well as the runtime required to obtain these results. On the next pages, we are first going to see principle quality properties of electrostatic halftoning. They tell us how the system behaves under varying environmental conditions, and what its benefits and drawbacks are compared to other methods from the literature. Secondly, we go into detail about the extensions and modifications presented in Section 7.4. For each of these supplements, we see several examples and its effect compared to standard halftoning. Whenever one of these extensions relies on a free parameter, we are going to evaluate its effect on the solution. Finally, we have a look on the runtime of the process depending on the number of particles and the image size. In this context, we also compare the runtimes of the two different algorithms from Sections 7.5 and 7.6.

### 7.7.1 Examples

Before we perform a detailed analysis of electrostatic halftoning and its extensions, let us first see two examples for this technique. Both were obtained without any extension being applied, and shall demonstrate the high visual quality of this method in its general form.

Figure 7.11 shows the famous *Trui* test image, and an electrostatic

halftone of it using 30 150 dots. This number corresponds to a setting in which each dot has the same area as one pixel in the original image. As can be seen in this figure, electrostatic halftoning creates results that preserve both the grey value and the structures of the original image. Details in the shawl are equally well represented as smooth gradients between face and hair or homogeneous regions such as the cheeks. However, if we view the digital version of this document in a viewer software, we notice moiré-like artefacts at different zoom levels. These are a drawback of the energetic optimality of the solution and are remedied by the jittering extension which we evaluate in Section 7.7.3. Finally, let us note that this image has favourable characteristics which suggest it as a basis of evaluation. It contains high frequencies, large homogeneous regions, and smooth transitions between grey values such that all critical situations for halftoning methods are covered. Moreover, it is small enough to allow evaluation methods and reference implementations to yield results in a reasonable amount of time. Hence, we use this image as a source of benchmarks in the course of this chapter, unless the fine differences occurring in an experiment can better be shown on a different image.

Secondly, we regard one huge example which involves a high number of dots on a relatively large image. This is shown in Figure 7.12, by using 647 770 dots on *Tiger*, 1 024×1 024 pixels. Again, the seemingly arbitrary choice for the number of particles arises from a normalisation of their individual areas to pixels from the input image. Although the image covers a significant partition of the page, one can hardly identify single dots. The overall impression is smooth but very detailed. Small structures, such as the fur or the reflections in the eye, are preserved with a high quality. This is also visible in the zooms in parts b. and c. Nevertheless, this image can be computed in a very short time. Using the fast summation approach, $420 \cdot 10^9$ interactions per iteration are evaluated in less than 820 milliseconds. Given that about 200 iterations suffice to obtain a result that can hardly be distinguished from the one given in Figure 7.12, a reasonable result can already be computed in about 3 minutes. With 1 000 iterations using time steps of 0.1, the process is almost entirely converged. This corresponds to a runtime of less than a quarter of an hour, which is much faster than most stippling approaches from the literature [Sec02, BSD09]. Later in this chapter, we see more examples for this issue.

### 7.7.2 Evaluation of Quality

From Section 7.2, we know that the quality of a halftoning process is caused by two different factors. One is the rendering operator, i.e. the way of how

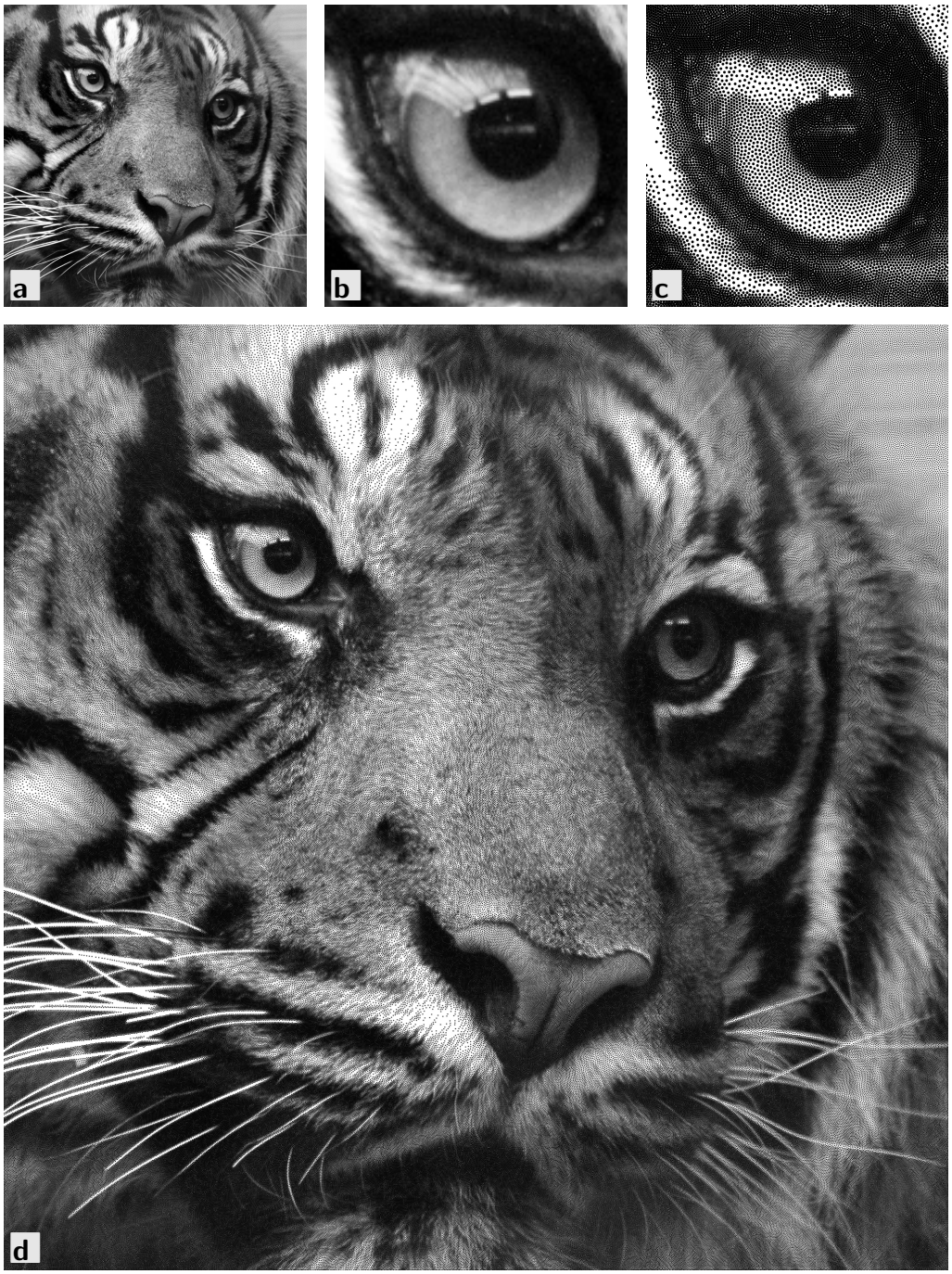



**Figure 7.12**: Electrostatic halftoning with 647 770 dots on *Tiger*, 1 024×1 024 pixels (License: TeryKats, flickr.com, CC-BY). **a.** Original. **b.** Zoom into eye region. **c.** Excerpt of halftone in the eye region. **d.** Full halftone (1 000 iterations, 820 ms per iteration using the fast summation technique).

results are reproduced on the target medium. The quality of this operator is deeply connected to its implementation. Different printers and screens give different results, while the theoretical prototype for such rendering operator is well understood (cf. Section 7.2.1). Hence, we now focus on objective measures for the quality of the essential component of electrostatic halftoning, the sampling operator.

### Approximation Errors

One key property of halftoning is a good approximation of the underlying image. The better a halftone, the less an observer should be able to distinguish it from the original. It is also evident that a halftone can in general never reach the quality of the original, since its co-domain is limited to two binary values while an image can map a coordinate to a value from a potentially continuous interval. This difference causes halftones to contain artificial high frequencies which are not present in the original image. In order to illustrate this issue, one can imagine a flat area of medium grey value. It is clear that a good halftone for it should contain about as many black as white pixels, and that those should be fairly uniformly distributed on the plane. Independent of whether the halftoning method chooses a regular checkerboard pattern or an irregular stochastic pattern, it always generates new frequencies, since the original image does not possess any variations. To this end, we exploit the fact that the human visual system most closely corresponds to a low-pass filter (cf. Section 7.2). When an observer moves away from an image, high-frequent features begin to disappear while low-frequent features survive much longer.

Figure 7.13 shows an excerpt of a halftone on the *Trui* image by means of electrostatic halftoning, and by the method of Balzer *et al.* [BSD09]. In the second row, these results and the original image are blurred by convolution with a Gaussian of standard deviation $\sigma = 1$. The blurred original is a ground truth against which the blurred halftones can now be evaluated. This comparison is frequently called a *difference of HVS responses*, and can either be obtained by a subjective comparison or by computing an error measure such as the mean square error [LA08]. First, we are interested in the visual difference, which is also depicted in Figure 7.13. For a better visibility, the difference images depict errors scaled by a factor 10. A zero error is shown in 50% grey.

Note that the method of Balzer *et al.* has the number of sampling points per dot as an adjustable quality parameter. This number can freely be chosen and must not be confused with the number of dots, which is given by the size of one dot and the average grey value of the image. Unless

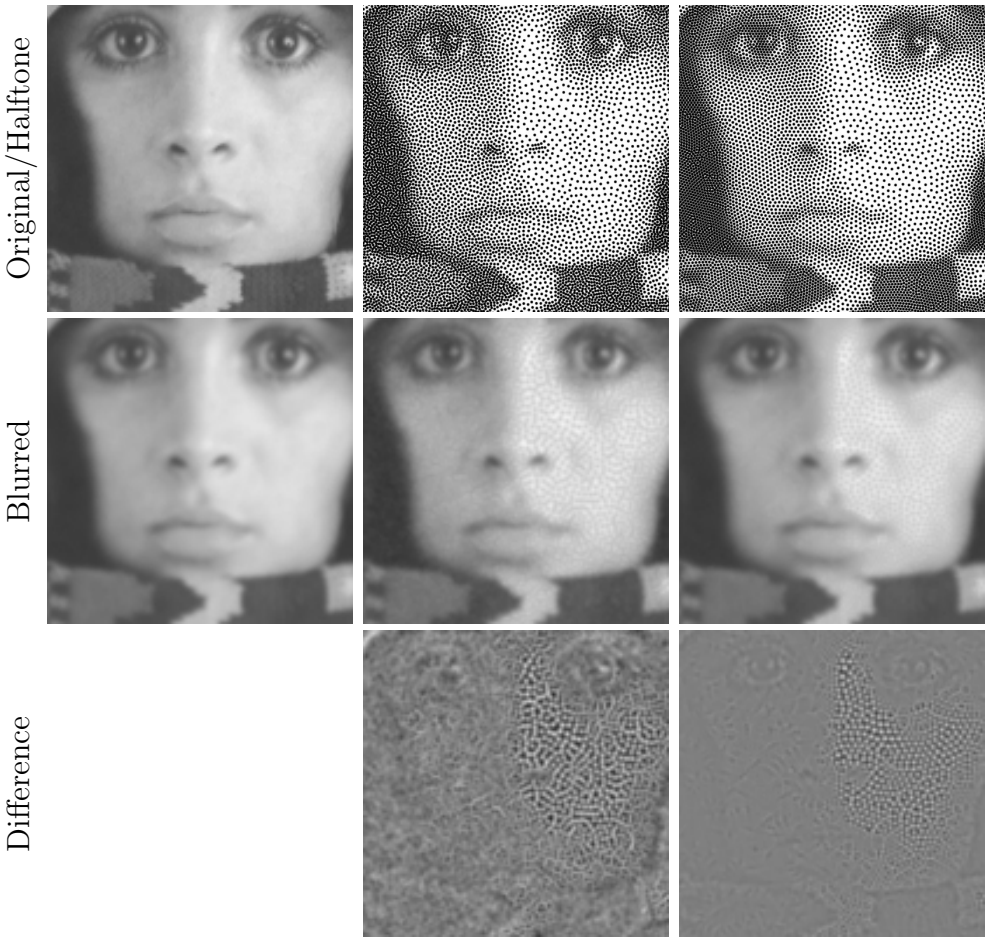



**Figure 7.13**: Approximation quality of electrostatic halftoning compared to the method of Balzer *et al.*, under a Gaussian with $\sigma = 1$. Difference images are scaled by a factor 10. Medium grey means no difference.

specified differently, we always use the choice 1 024 as specified in [BSD09].

Compared to the capacity-constrained method of Balzer *et al.*, electrostatic halftoning creates much smoother and more consistent results. This is in particular visible in the difference plot. Almost no image structures are visible in the error image, which hints at a good approximation of the image. Moreover, the result does not reveal striking noise artefacts in the whole image domain. Only in the cheek region within the right half of the image, the error is slightly higher. This phenomenon is related to the bright tone of this region and the resulting large distances between particles. To blur these relatively low frequencies to a flat area, higher standard deviations are required. Thus, it results from a principle weakness in the evaluation which does not account for different grey values rather than being an error of the method. As a consequence, this artefact is also strikingly visible in the error plot for the method of Balzer *et al.*

Accounting for the dependency on the choice of the standard deviation $\sigma$ is nontrivial. From the motivation of an evaluation under conditions of the human visual system, the standard deviation corresponds to a certain viewing distance. At the other hand, different grey values cause different average particle distances, which in turn requires different standard deviations to distribute the colour information homogeneously in the area of influence. In a physical interpretation, this insight leads to a dilemma because different parts of the image would require different viewing distances.

Even from a technical point of view, such spatially variant evaluation technique bears problems. In general, we consider images with sharp edges, and an isotropic smoothing with a large kernel in bright regions would also diffuse information from nearby dark regions into this area. However, kernels in dark areas were smaller, such that less bright tones are transported into these regions. Hence, the arising process does not preserve the average grey value, and can consequently not be used as a solid basis for comparison. However, anisotropic diffusion processes (cf. Chapter 4) are also no option. Although they can preserve edges, they represent in general no valid model for the human visual system.

As a consequence of these considerations, we refrain from an evaluation under Gaussian filtering, but leave the standard deviation as a free parameter. To this end, we call a method better than a second technique if the similarity to the original is higher under convolution with Gaussians of a larger range of 'meaningful' standard deviations. In this context, 'meaningful' is a rather flexible term and depends on subjective or application-dependent preferences, the rendering operator, as well as on the resolution of the halftone on the target medium. Most experts would probably agree that the interval of about $\sigma \in [1.0, 10.0]$ is important for human percep-

tion. In the literature, one often finds comparisons that are based on more complex HVS models which additionally take into account under which environmental conditions the halftone is viewed [LA08]. However, since all of these filters have a lowpass character, they can fairly well be compared to a Gaussian filter as applied here and map to a similar range of admissible standard deviations.

Note that for $\sigma \ll 1$, almost no information is diffused, such that such choices require a similarity on a per-pixel basis. This error is minimised by a simple thresholding approach whose results usually do not conform to our intuitive understanding of a halftone. The other extreme, $\sigma \to \infty$, does also not provide a distinctive measure for comparison, as it can only detect changes in the average grey value.

A standard method for similarity is the *peak signal to noise ratio* (PSNR)

$$\mathrm{PSNR}(\boldsymbol{a}, \boldsymbol{b}) = 10 \log_{10} \left( \frac{1}{\frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{a}_i - \boldsymbol{b}_i)^2} \right) \tag{7.64}$$

between the smoothed original $K_\sigma * u$ and the smoothed halftone $K_\sigma * H(u)$. $K_\sigma$ denotes a Gaussian with standard deviation $\sigma$. Since this process depends on $\sigma$, we plot the PSNR against all meaningful standard deviations. To this end, a higher value indicates that the $L^2$ distance between the arguments is lower for this particular standard deviation such that the halftone is more similar to the original under the corresponding viewing distance.

Figure 7.14 depicts the PSNR for electrostatic halftoning and for the method of Balzer *et al.* [BSD09] for standard deviations $\sigma$ from 0 to 15 on the *Trui* image. Besides the standard setting of 1 024 points per site for the latter approach, we also compare against this method with a high-quality setting of 8 192 points per site. Due to their high memory requirements, these solutions could no longer be computed on the available hardware. However, the authors of the original publication kindly agreed to compute and supply these reference solutions. [1]

Our experiment shows that electrostatic halftoning performs in all scales much better than its competitor, regardless of the quality setting of the latter method. The perceptual distance between the graphs in the plot is in particular astonishing if we recall that the PSNR describes a logarithmic measure. Hence, electrostatic halftoning even possesses a more than ten times smaller approximation error for $\sigma > 10$ on a linear scale.

In particular, electrostatic halftoning seems to be more optimal in a global sense than the method of Balzer *et al.*. In order to support this

[1] My thanks go to Thomas Schlömer from the University of Constance for kindly providing reference results for comparison.

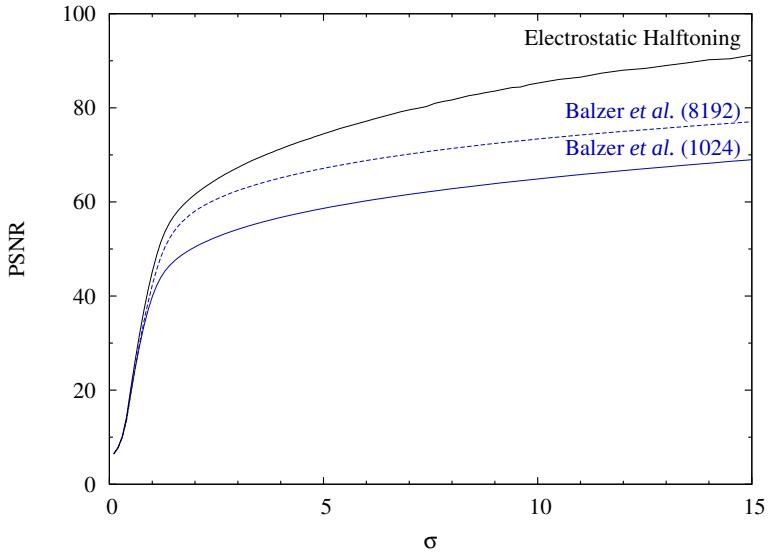



**Figure 7.14**: PSNR of electrostatic halftones for *Trui* smoothed by Gaussian convolution with standard deviation $\sigma$ with respect to the smoothed original. Results for the method of Balzer *et al.* with 1 024 and 8 192 points per site are given for reference.

claim in more detail, let us briefly go into detail about the co-domain of the PSNR. By construction, the PSNR is unbounded, since two identical images let the denominator in (7.64) vanish. However, a halftone can in general not approximate an image to an arbitrary degree. The reason for this is that we halftone with a certain number of black discs with unit area. These discs, together with the empty white space in between, are supposed to have the same average grey value as the image. However, since the image consists of values that are taken from a continuous interval, there is often a slight inconsistency between the average grey values of the image and the halftone. For instance, the *Trui* image has an average grey value of about 0.53994165, while its halftone has an average grey value of about 0.53994751. This difference bounds the PSNR to 104.6380. As a consequence, the PSNRs of all evaluated methods approach this value when the grey value information is equally diffused over the whole image domain, i.e. for $\sigma \to \infty$. Electrostatic halftoning approaches this value much faster, which means that it creates a very good approximation of the image, locally as well as globally. Moreover, one should remark that the optimum for any halftoning method at any standard deviation cannot exceed the maximal PSNR given above. Compared to its competitor, and

considering the logarithmic nature of the co-domain, electrostatic halftoning has reduced the gap between theory and practice tremendously. By this, it represents the best halftoning technique available today.

### Spectral Analysis

Next, we evaluate the spectral properties of electrostatic halftoning. They are an indicator for the regularity and noisiness of a halftone $h$. To analyse these properties, the method in charge is applied to a flat image. In the pointwise squared Fourier spectrum of this result, called the *power spectrum* $\boldsymbol{P}_h(\boldsymbol{x})$ of $h$, one can then check if the halftone fulfils so-called *blue noise* characteristics [Uli88]. The closer the halftone mimics this behaviour, the better is its quality.

The idea behind these methods is the requirement that dots are distributed equally over the image domain and that they do not suffer from aliasing artefacts. The average distance between two neighbouring dots is characteristic for the grey value the halftone approximates. If two points are moved closer to each other, the result appears darker, if they are moved further apart, the image becomes brighter. For every grey tone that we halftone, we can thus compute the *principal frequency* $f_p$ in which we expect the points to appear [Uli88, LA08]. In the power spectrum, this property is reflected by a high peak at the radial frequency $f_p$.

So far, the measurement bears one important drawback. Because it was only generated by one halftone, it shows the properties of this particular image, rather than those of the whole scheme. As an example, consider the directional dependency of the power spectrum. A flat grey image can ideally be represented by a regular grid. Although the method might be rotationally invariant, each individual halftone still does reveal one direction in which its grid is oriented. An individual power spectrum can thus be biased, although the family of power spectra for the halftoning method does not possess any bias. These stochastic discrepancies can be eliminated if several independent samples are drawn. This means we compute several halftones for the same grey value, compute their power spectra, and average these to obtain a well-founded basis for comparison [Bar78]. Note that we can even draw different samples out of the result for a deterministic method if we render a large image, and cut out subplanes at random positions.

Figure 7.15 shows an example for a power spectrum. The peak at the radial frequency $f_p$ appears as a bright circle, and has been annotated with a blue line. To characterise this peak, we distinguish two measures:

- The **radially averaged power spectrum** (RAPS) is a histogram

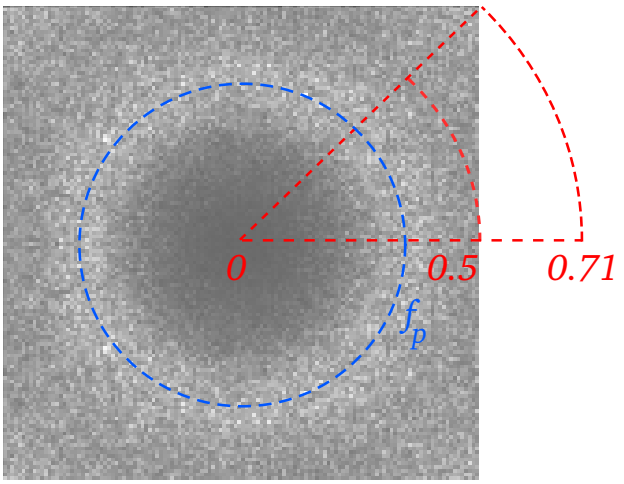



**Figure 7.15**: Annotated power spectrum with blue noise properties.

of the power on concentric circles, normalised by the perimeter of the circle. Algorithmically, one can imagine this measuring process as a clock hand that performs one full turn, while integrating all entries in distance $n$ from the centre to obtain the value of the RAPS at point $n$. To this end, the RAPS to a halftone $h$ is defined as

$$\mathrm{RAPS}_h(f) := \frac{1}{2\pi f} \oint_{C(f)} \boldsymbol{P}_h(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \tag{7.65}$$

with

$$C(f) := \{ (x, y) \mid x^2 + y^2 = f^2 \}, \tag{7.66}$$

and $\boldsymbol{P}_h$ again denoting the power spectrum of $h$. The desired properties of the RAPS can best be seen if we plot it against the frequency $f$ (cf. Figure 7.16). A good blue noise behaviour is characterised by a vanishing low-frequency band, a steep ascent to a dominant peak at the principal frequency $f_p$, and a flat high-frequency area [Uli88]. The shape of the peak is an indicator for the regularity of the halftone. A thin high peak denotes a very regular halftone, while a flat but wide peak shows that the distribution is more random and noisy. The latter has a direct effect on the approximation quality with respect to the image. The requirement of a flat high-frequency area arises from principal limitations. High frequencies cannot be avoided at all, because they are omnipresent as multiples of low frequencies. However, their low variation indicates that no additional frequencies are generated by an unintentional clustering of dots. Finally, one should note

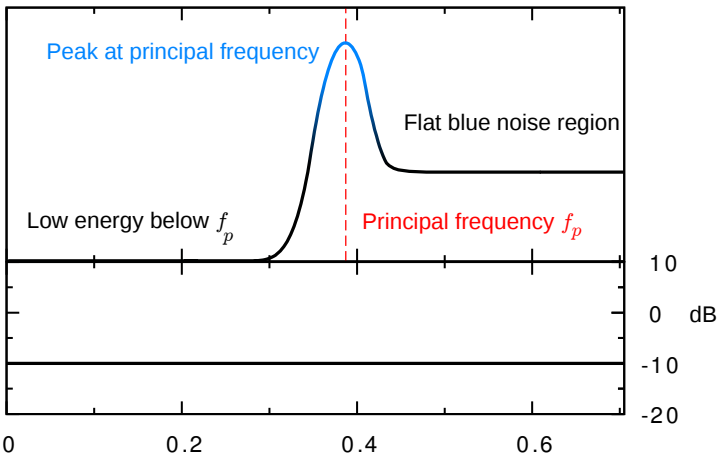



**Figure 7.16**: Ideal blue noise behaviour as defined by Ulichney [Uli88].

that the stability of the measurement drops above $f = 0.5$ such that measurements for very high frequencies are less reliable than for lower ones. The reason can be seen in Figure 7.15. While up to $f = 0.5$, full circles are available for averaging, only a few values in the corners of the power spectrum must suffice for larger $f$. In real measurements, the error thus rises slightly for $f \to \sqrt{0.5}$.

According to Ulichney, the principal frequency $f_p(u)$ for a flat image with grey value $u$ is given by

$$f_p^{(r)}(u) = \sqrt{\frac{1}{2} - \left|u - \frac{1}{2}\right|}\,, \tag{7.67}$$

if the halftone lies on a rectangular grid, and by

$$f_p^{(h)}(u) = \frac{2}{\sqrt{3}} \sqrt{\frac{1}{2} - \left|u - \frac{1}{2}\right|}\,, \tag{7.68}$$

if it lies on a hexagonal grid [Uli88]. For continuous halftones as we evaluate them here, neither of these two principle frequencies are exact. Although we expect dots to form hexagonal structures, there is no defined grid direction. Moreover, the result can be biased by the fact that we consider rectangular input images on a regular grid. To this end, the expected principal frequency lies somewhere between these two values. This is also the reason why throughout this thesis, plots for continuous methods indicate the interval for admissible principle frequencies with two dashed lines in the RAPS.

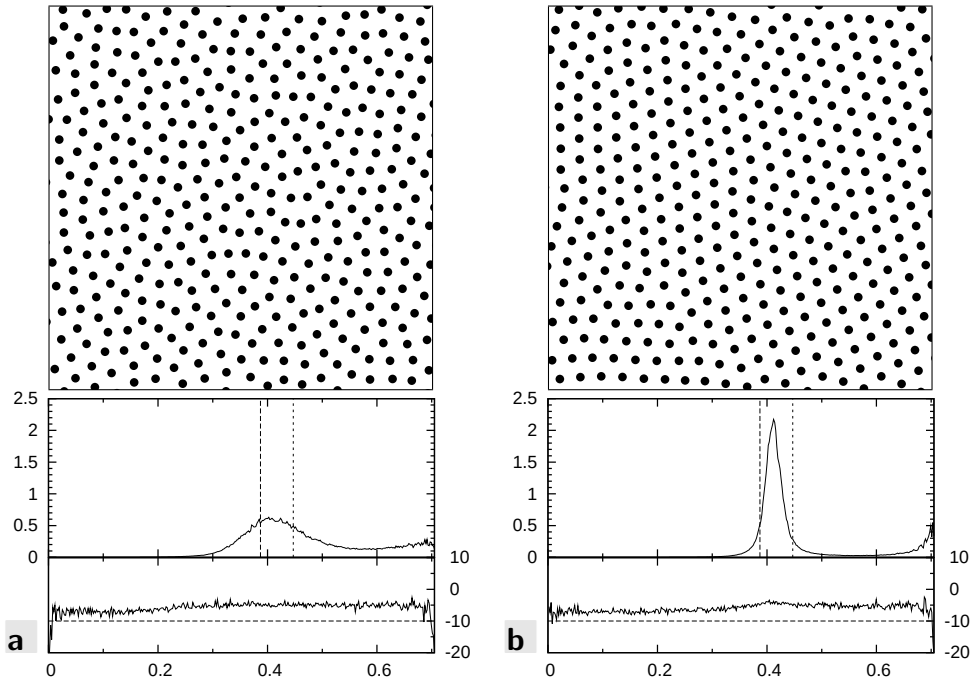



**Figure 7.17**: Spectral analysis of **a.** the method of Balzer *et al.*, and **b.** electrostatic halftoning.

- Directional preferences of a method are given by its **anisotropy** measure. As opposed to the RAPS, this measure denotes the variance over each concentric circle. A high anisotropy for a certain frequency $f$ means that this frequency occurs more often in specific directions than in others, thus indicating a directional bias of the method. The anisotropy of a halftone $h$ is given by

$$\mathrm{Ani}_h(f) := \frac{1}{2\pi f} \oint_{C(f)} (\boldsymbol{P}_h(\boldsymbol{x}) - \mathrm{RAPS}_h(f))^2 \, \mathrm{d}\boldsymbol{x}. \tag{7.69}$$

  As stated before, the measured anisotropy depends on the number of stochastic samples drawn from the halftone. For 10 samples, one can show that the anisotropy is bounded to a background noise level of $-10$dB (cf. Figure 7.16). In all plots in this thesis, this theoretical limit of the evaluation technique is indicated by a dashed line in the anisotropy plot.

Let us now analyse the spectral properties of electrostatic halftoning and compare it against those of the capacity-constrained method of Balzer *et al.* Both methods were run on an uniform image with grey tone $u = 0.85$.

Excerpts from their results, as well as a blue noise analysis, are shown in Figure 7.17. In a direct comparison of both results, the method of Balzer *et al.* generates halftones which look more random than those obtained by electrostatic halftoning. The latter method in turn convinces by large patches of almost perfect hexagonal grids. However, this comes at the expense of striking crystallisation artefacts at the boundaries between these areas. Still, neither of the two methods shows striking anomalies such as clusters or inhomogeneous white spaces between the dots.

The high quality but different regularity of the two halftones can also be seen in the RAPS. Both methods match the principle frequency very well and show clearly visible bluenoise characteristics. However, the peak at the principle frequency is more than 3 times higher for electrostatic halftoning than it is for the capacity-constrained method. When it comes to an energetically optimal, noise-free approximation of the given image, electrostatic halftoning thus performs much better than its competitor. The anisotropy lies for both methods at about $-5$dB, which is a very good result considering that we have non-toroidal images.

One should note that some applications prefer a more random distribution of points over an accurate approximation of the underlying density. However, additional randomness in the results can as well be obtained with electrostatic halftoning. Using the jittering extension from Section 7.4.4, the degree of randomness can even be freely adjusted. The approach of Balzer *et al.*, in contrast, can neither be tuned to give more accurate nor to give more random results. We see a detailed evaluation of the jittering extension later in this chapter.

### Quality of Fast Summation

Our NFFT-based fast summation algorithm from Section 7.6 approximates rather than accurately imitates the direct summation algorithm. As a consequence, this process introduces approximation errors, which are caused by the cut-off parameter $\tilde{m}$, as well as by the degree of the polynomial $\hat{p}$ used to describe the kernel. Without going into detail about the theoretical background of these two parameters, let us regard both as one abstract quality parameter. It constitutes a tradeoff between runtime on the one hand and quality on the other.

We start with a comparison of halftones for *Trui* as in Section 7.7.2. Even at the very first glance, we see that the results obtained with $\tilde{m} = \hat{p} \in \{1, 2\}$ constitute a bad approximation of the original. For $\tilde{m} = \hat{p} = 2$, particles are even stacked, which makes this choice of parameters unusable for any purpose. This changes for $\tilde{m} = \hat{p} \in \{3, 4, 5, 6\}$. The results obtained

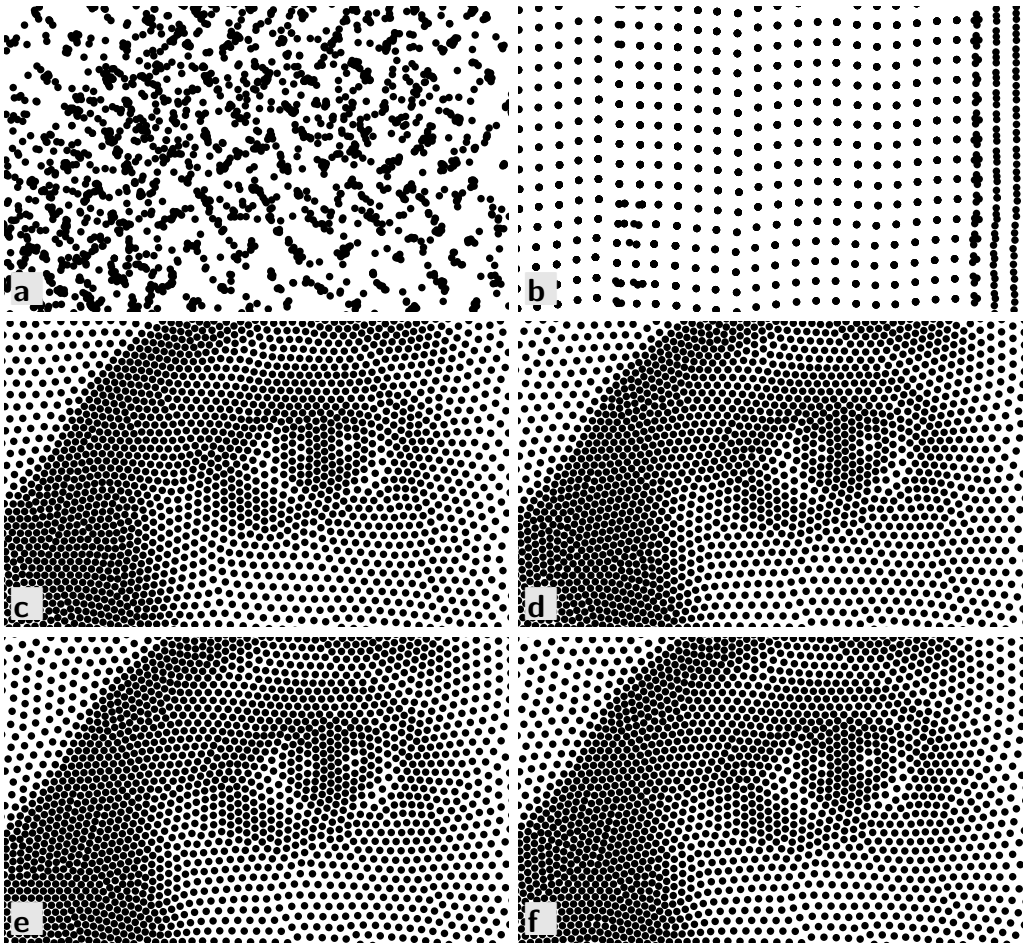



**Figure 7.18**: Excerpt from fast summation halftones on *Trui* with quality setting $\tilde{m} = \hat{p} \in \{1, 2, 3, 4, 5, 6\}$, respectively.

with any of these values are hardly distinguishable. Unsurprisingly, this is also reflected in the PSNR for the blurred results against the blurred original. As shown in Figure 7.19, the outcomes for $\tilde{m} = \hat{p} \in \{4, 5, 6\}$ are almost identical and very close to the solution obtained with the direct summation approach, while the result for $\tilde{m} = \hat{p} = 3$ follows close up. In contrast, the graphs for $\tilde{m} = \hat{p} \in \{1, 2\}$ show a clearly worse behaviour. This indicates that $\tilde{m}$ and $\hat{p}$ should be chosen greater or equal to 3 to obtain a good approximation.

As a second criterion for quality, we additionally evaluate these different instances of the fast summation algorithm with respect to their spectral properties. This is done in Figure 7.20. Like in the previous section, a uniform image of grey value 15% was halftoned and the results were analysed with respect to their blue noise properties. The outcome of this experiment

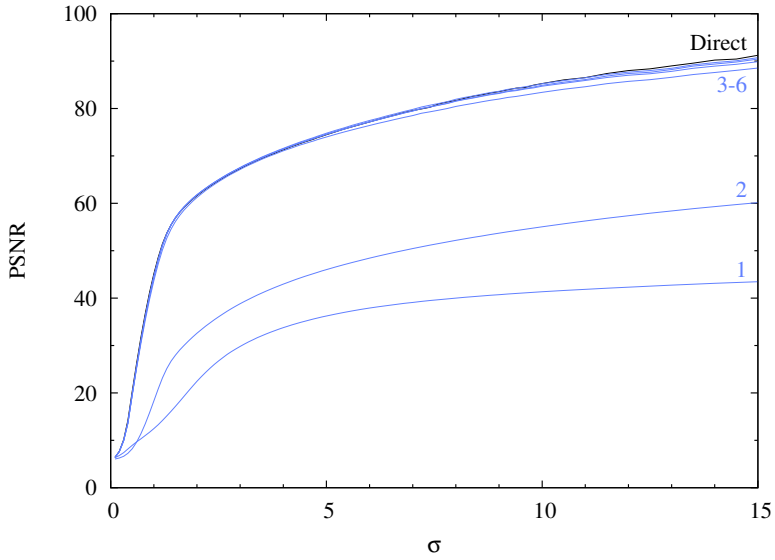



**Figure 7.19**: PSNR of halftones for *Trui* smoothed by Gaussian convolution with standard deviation $\sigma$ with respect to the smoothed original. From bottom to top: Fast summation for $\tilde{m} = \hat{p} \in \{1, 2, 3, 4, 5, 6\}$, and direct summation.

is surprising. All instances with $\tilde{m} = \hat{p} \leqslant 4$ perform unsatisfactory with respect to both the RAPS and the anisotropy measure. The bad behaviour of the result for $\tilde{m} = \hat{p} = 4$ can best be explained if we have a closer look on the sample patch rendered in Figure 7.20d. It reveals that points are arranged much too regular, almost like residing on a fixed grid. If we go back to Figure 7.19, we can spot such artefacts as well, although they are less prominent. They seem to be grey-value dependent, and can best be seen on the bright background in the top left corners. For $\tilde{m} = \hat{p} \geqslant 5$, these regular structures are replaced by small patches that contain a regular, but randomly rotated structure. As a consequence, no directional preferences can be identified and the process possesses blue-noise properties.

To this end, we can draw two conclusions out of these experiments. Whenever we are interested in a result with the same high quality as the direct summation approach, we should set $\tilde{m} = \hat{p} = 5$. This is also done for all following experiments, unless stated otherwise. However, we also have the opportunity to accelerate the process even further if we only require a visually convincing result which does not necessarily need to be perfect. In either case, however, we should not choose $\tilde{m} = \hat{p} < 3$. In Section 7.7.5, we compare the runtimes for different quality settings with each other. This way, we obtain a feeling for the tradeoff between quality and runtime.

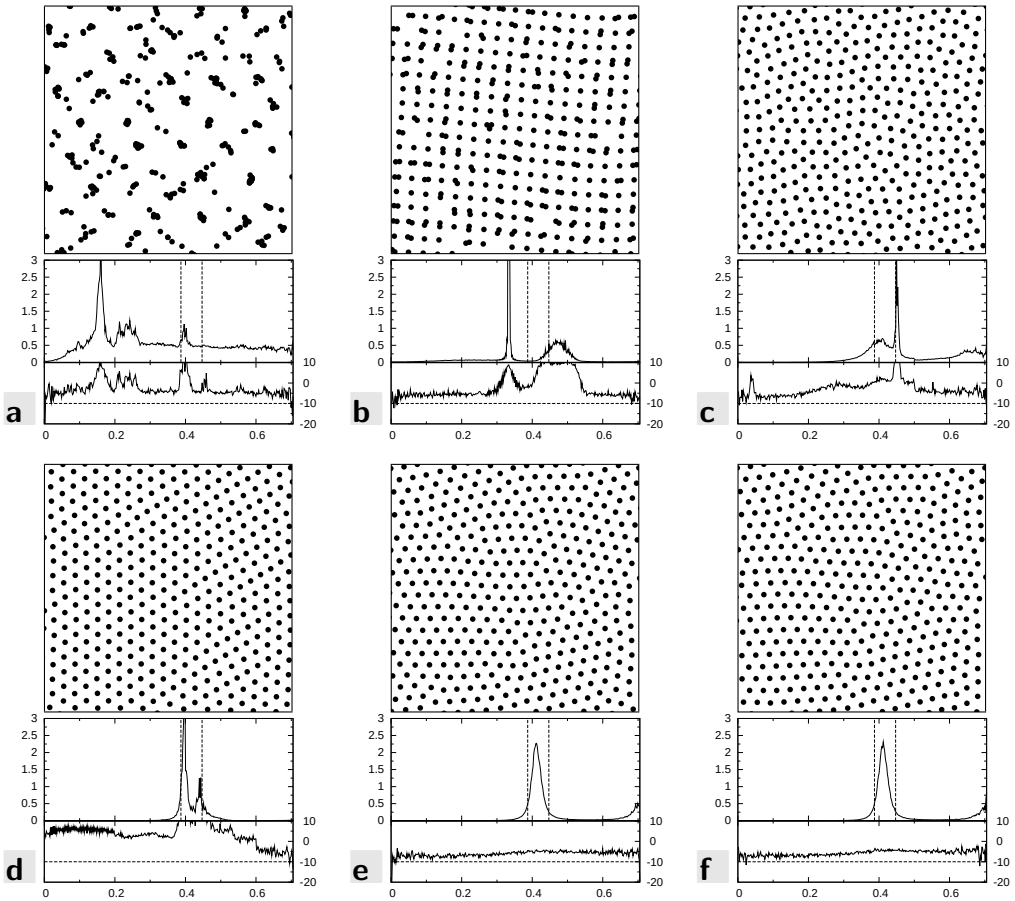



**Figure 7.20**: Spectral analysis of fast summation halftones obtained with quality settings $\tilde{m} = \hat{p} \in \{1, 2, 3, 4, 5, 6\}$, respectively.

## 7.7.3 Modifications and Extensions

### Dithering

As a next step, we evaluate the extensions proposed in Section 7.4, and start with dithering. Because in many applications discrete results are preferred over continuous ones, the experiments performed here are very important. They show whether electrostatic halftoning is well suited for these applications and can reveal potential drawbacks.

However, before we analyse the quality of electrostatic dithering compared to other methods from the literature, there is one fundamental question to be answered. In Section 7.4.1, we discussed that a replacement of the rendering operator by rasterisation is a necessary but insufficient change to obtain the desired results. In addition to this, we need an 'egg-crate' shaped potential along the grid points, as well as a projection of particles to the

closest grid line. The effects of each of these extensions is subject of the next experiment.

In Figure 7.21b, we see the halftone of the seagull image which was obtained by a rasterisation of the continuous electrostatic sampling. The quality of this halftone is very low. Regions with dark or medium grey tone are dominated by line-like aliasing artefacts. At the same time, 364 pixels contain more than one particle. In image c, these locations have been circled in red. Although we can still print this image if we interpret pixels with an arbitrary number of particles as a black one, this thresholding process also modifies the average grey value. Thus, the halftone does no longer approximate the original image. In Figure 7.21d, the sampling operator was amended by the additional attractive potential towards grid positions. The gain in quality is clearly visible. The white artefacts in the background disappear completely, and all regions are represented by a suitable amount of homogeneously distributed particles. Still, there are 8 locations in which a pixel was assigned more than one particle. This means the average grey value of the image is still not preserved. In part e, the sampling operator was exclusively extended by the projection step. This modification entirely resolves the problem with double assignments, as can be seen in the image. However, the halftone is still not optimal. In the black region in the top right corner, irregular line-like patterns arise. Even in the background, the rendering looks more coarse than for image d. Finally, both modifications are combined to obtain the result shown in Figure 7.21f. It combines the high visual quality of d with the freedom of double assignments from e and provides a smooth and realistic representation of the original.

Next, we perform a qualitative analysis similar to the evaluation in Section 7.7.2. This experiment compares discrete electrostatic halftoning to other popular methods from the literature. Compared to the continuous case, where very few methods dominate most of the applications, the dithering market is highly contested. This might be due to the long history of dithering methods, their numerous applications, and their different tradeoffs between speed and accuracy. Most of these techniques try to find a halftone which minimises a local error to the original. A prominent class of representatives for this purpose is the so-called error diffusion. By processing the image on a certain trajectory like scanlines, these algorithms binarise one pixel by thresholding and distribute the approximation error into the unprocessed neighbourhood. Starting with the famous work of Floyd and Steinberg [FS76], several optimisations were proposed in the literature. As two modern representatives of this class, we additionally consider the approaches of Ostromoukhov [Ost01] and Zhou *et al.* [ZF03]. In particular if a dithering method shall be implemented on printer hardware, one frequently

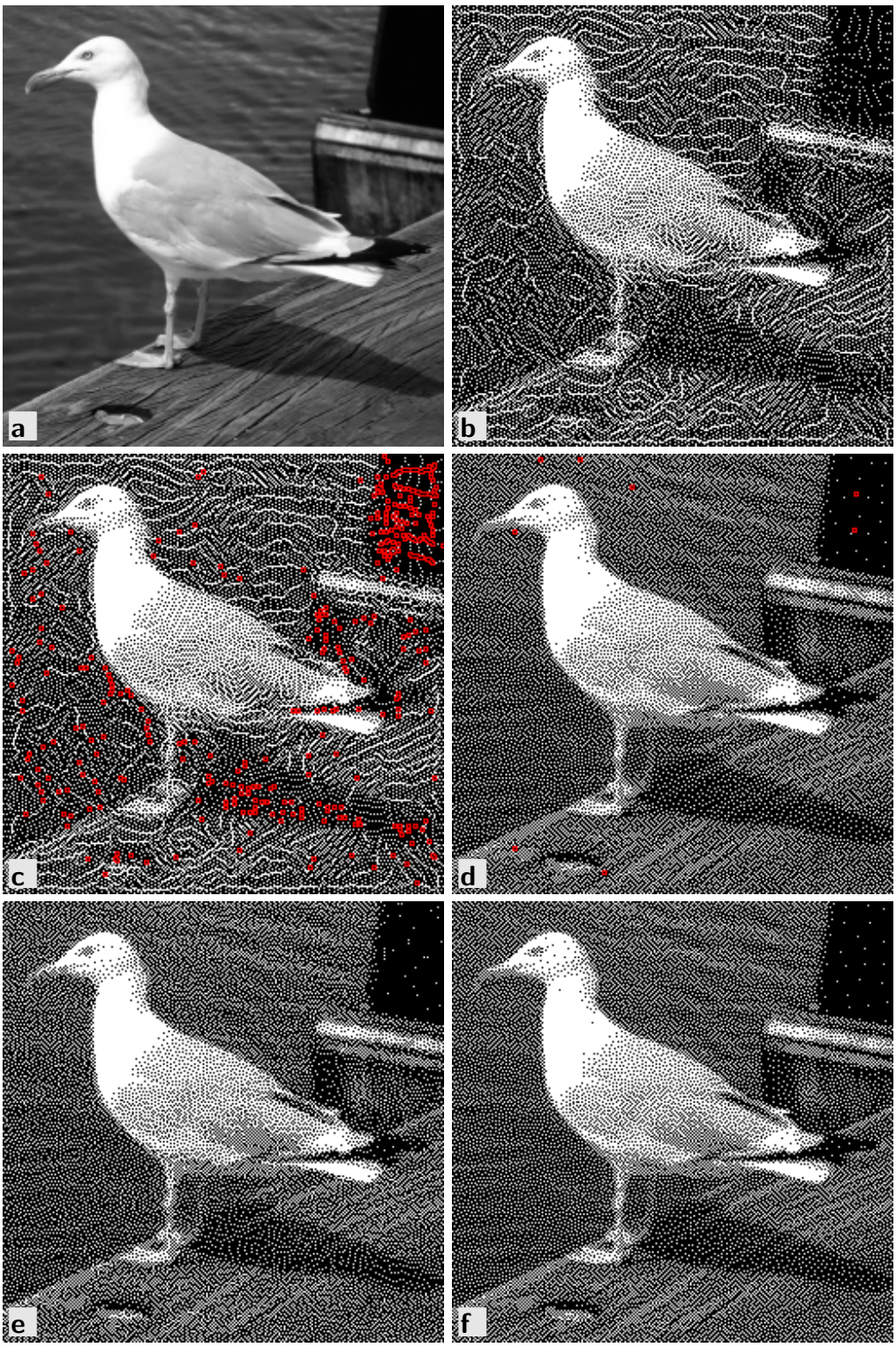



**Figure 7.21**: Effect of the egg-crate potential and projection for dithering, with double assignments annotated in red (c–f). **a.** Original (256×256). **b,c.** Rasterisation of the continuous result. **d.** Potential, but no projection. **e.** Projection, but no potential. **f.** Potential and Projection.

refrains to algorithmically more simple and data-parallel threshold screening techniques. These methods use a mask of continuously distributed grey values. If the mask at a certain pixel is darker than the continuous-valued input image, the corresponding pixel is white, and vice versa. The quality of halftones typically depends on the stochastic properties of the used mask. In this experiment, we compare the famous forced random dithering technique [PTG94] with a commercial method. From the latter, samples were kindly provided by a company which prefers to remained unnamed. Finally, we additionally take into account two methods which both aim at a perceptual similarity of the original and the halftone. While the method of Geist *et al.* [GRS93] sets up on a Markovian framework to obtain these visually enhanced results, the structure-aware halftoning method of Pang *et al.* [PQW$^+$08] minimises a similarity measure that takes into account local gradients. In order to make the latter method better comparable in terms of tone similarity, we additionally consider a variant of this method in which the structure similarity term was disabled. Unfortunately there is no such option for the method of Geist *et al.*

All methods mentioned so far are compared against the (discrete) electrostatic halftoning approach presented in Section 7.4.1. Similar to the experiments from Section 7.7.2, we use the *Trui* test image and dither it with the different methods. In a first step, excerpts of these dithering results, as well as blurred versions under a Gaussian of standard deviation $\sigma = 1.0$, are visually compared to each other.

The results of this experiment on *Trui* are shown in Figure 7.22, where the commercial approach is annotated with 'threshold screening'. For the method of Purgathofer *et al.*, a mask with the size of the whole image was chosen such that the depicted result can be seen as the optimal instance that can be obtained with this method. As a first impression, we observe a noticeable contrast enhancement by the approaches of Geist *et al.* and Pang *et al.*, while the latter method particularly amplifies small variations in fine structures. This can for instance be seen in the eye region of the depicted woman. However, the enhancement of structures clearly come at the expense of an accurate approximation of the underlying image, as is best visible in the difference plots. The result for the tone-similarity term from Pang *et al.* is qualitatively equivalent to the error-diffusion approaches of Floyd-Steinberg and Ostromoukhov, which indicates that the energy minimisation taking place for this part of the method only marginally affects the initialisation with the result of Ostromoukhov. All three results reveal striking grid-like artefacts in medium grey regions. Such artefacts are not visible in the results produced by the commercial threshold screening approach, the method of Purgathofer *et al.*, and the technique of Zhou *et al.*,

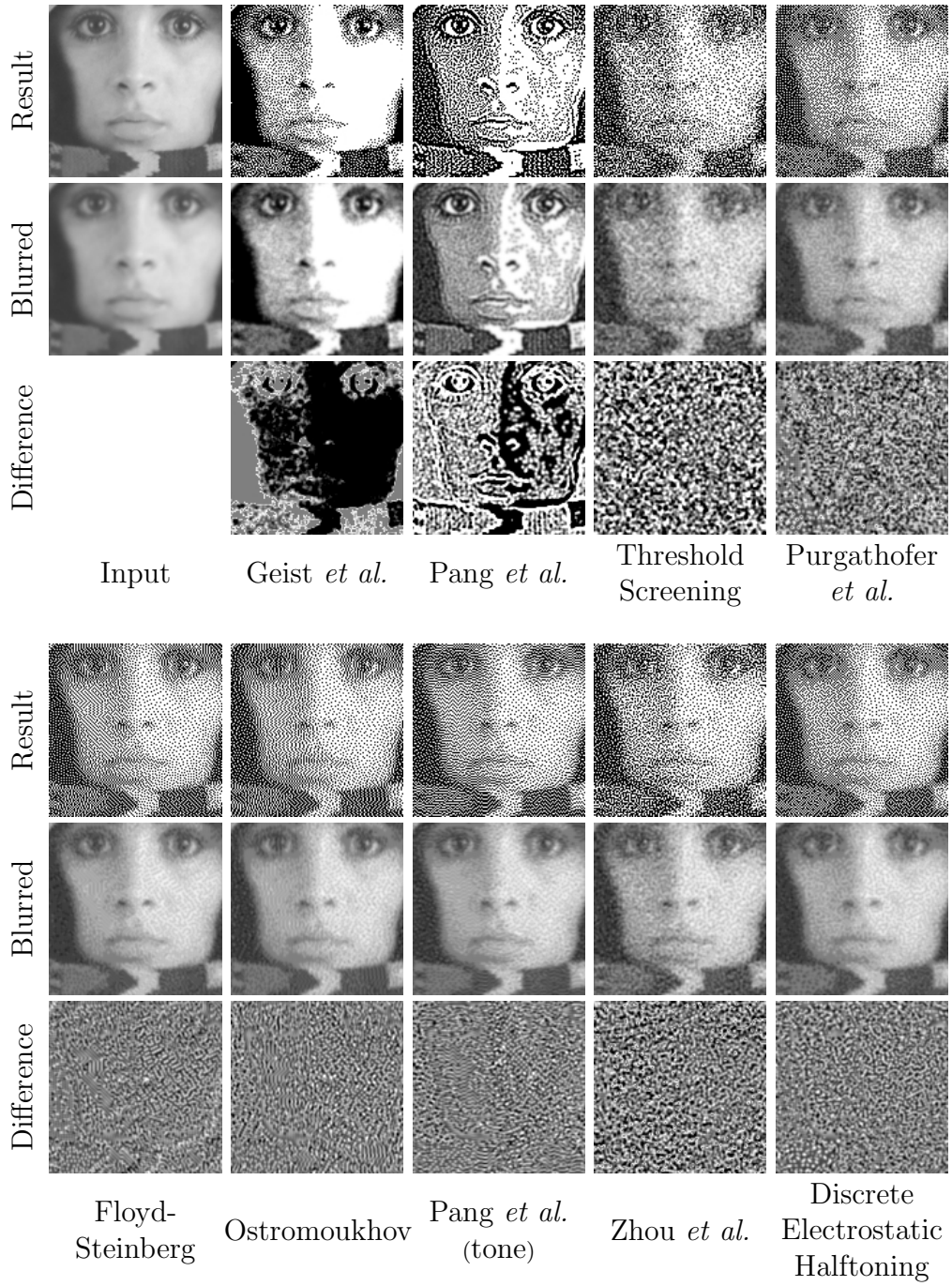



**Figure 7.22**: Approximation quality of discrete electrostatic halftoning compared to other dithering techniques, under a Gaussian with $\sigma = 1$. Difference images are scaled by a factor 10. Medium grey means no difference.

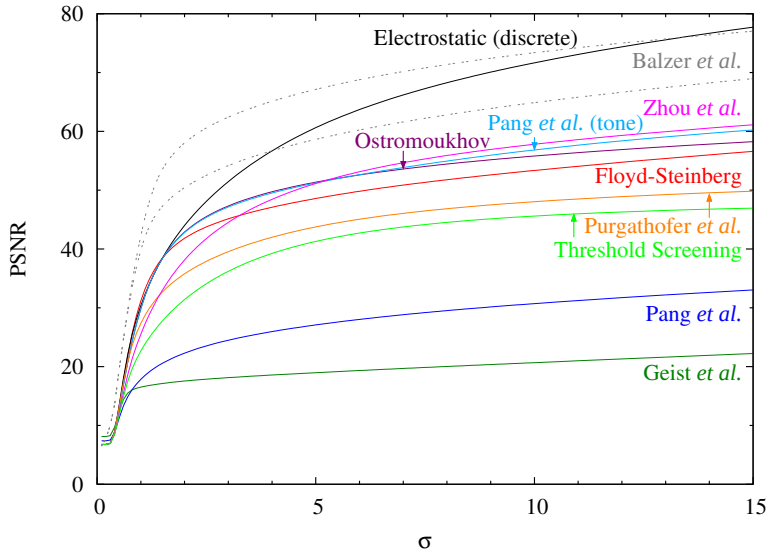



**Figure 7.23**: PSNR of ditherings of *Trui* smoothed by Gaussian convolution with standard deviation $\sigma$ with respect to the smoothed original. The continuous method of Balzer *et al.* (1 024, 8 192 points per site as in Figure 7.14) is given for comparison purposes.

but these images possess a more coarse appearance than those prodeuced by all other methods. In contrast to its competitors, electrostatic halftoning produces very fine-grained results without visible artefacts. Its difference plot against the smoothed original shows a fairly random pattern without visible image structures. This is an indicator for a high quality.

Next, we want to confirm this subjective impression by a numerical quality measure. As for the experiment described in Section 7.7.2, we thus convolve the results from the previous experiments with Gaussians in different standard deviations and compare their PSNR to the smoothed original This is shown in Figure 7.23. In order to give a rough estimate of the quality of discrete methods with respect to the continuous comparison from Figure 7.14, the graphs for the method of Balzer *et al.* with 1 024 and 8 192 points are again given as a reference.

Our subjective impression from the previous experiment is also reflected in this measurement. Electrostatic dithering outperforms all other methods significantly, in particular when it comes to a globally optimal approximation, i.e. large $\sigma$. For such settings, discrete electrostatic halftoning even outperforms the continuous method of Balzer *et al.* — although it obeys the additional constraint to place particles only on grid positions. Error

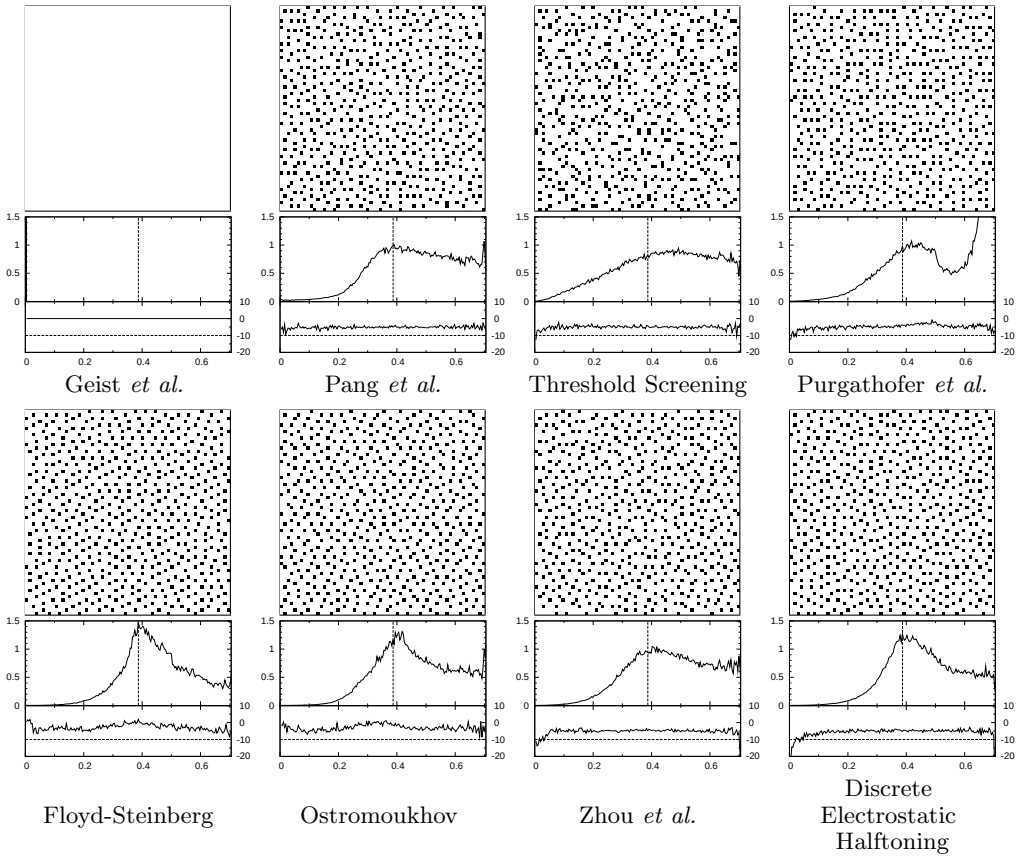



**Figure 7.24**: Spectral analysis of dithering methods.

diffusion algorithms and the tone-similarity variant of the method of Pang *et al.* constitute the second best class of dithering methods. These methods are qualitatively similar to each other, while the coarseness of the results obtained by the method of Zhou *et al.* is also reflected in this experiment. It dominates the other error diffusion algorithms for large $\sigma$, but performs much worse for small values. The threshold screening algorithms provide results of medium quality, only followed by the contrast enhancing approaches. Again we should note that the bad performance of the algorithms of Geist *et al.* and Pang *et al.* in this experiment is no meaningful statement, since the structure enhancing property they are targeting is not covered by this evaluation series.

Finally, we are interested in the spectral properties of the different halftoning techniques. Since this experiment works on flat grey images, the results are comparable among all evaluated techniques, independent of an optional structure enhancement. Moreover, we do not need to distin-

guish the two variants of the method of Pang *et al.* any longer because their results are bitwise identical due to a lack of structure in the images. Figure 7.24 shows the outcome of this experiment on a image of grey value 0.85. Due to the contrast enhancing property of the method of Geist *et al.* and the bright grey tone, it yields a plain white output image. This is a clear disadvantage of this method as it does not only lack a basis of comparison, but does also not preserve the average grey value of the original. Results generated by electrostatic halftoning and the method of Ostromoukhov possess the best blue noise properties, closely followed by the method of and of Floyd and Steinberg. All of them possess a steep slope towards the high peak at the principal frequency and a relatively flat high frequency area. The distinctive peak is missing for the results of Pang *et al.* and the commercial threshold screening technique, while the approach of Purgathofer *et al.* is dominated by undesired high frequencies.

To this end, we conclude that discrete electrostatic halftoning outperforms the best and most frequently used dithering algorithms from the literature by all criteria. Globally, it performs even better than the capacity-constrained approach of Balzer *et al.* which represents the second best continuous approach after electrostatic halftoning. This insight shows that electrostatic halftoning is not only able to create convincing continuous point distributions, but is also the first method to create better discrete dithering results than the class of error diffusion algorithms.

### Point Size Adjustment

In Figure 7.25, the image *The skull bisected and sectioned* by Leonardo da Vinci was halftoned in different resolutions. Given the input image in the size 200×300 pixels, the circular discs have been adapted to the radii 0.5, 1.0, 1.5, 2.0, and 2.5 pixels. Independent of the size of the dots, however, all halftones approximate the original very accurately. While for the one with the highest resolution, even small details such as single strokes are clearly visible, the coarsest halftone preserves all important structures. By this, the depicted object can clearly be identified. Furthermore, it also still contains fine details of the object such as the y-shaped widening of the vertical crack at the forehead or the exact shape of the ditch at the right temple. Electrostatic halftoning is thus very well suited as a preprocessing stage to printing. Even if details live beyond the physical resolution of the printer, they can still be represented in the print if dots can be placed on a continuous domain. Moreover, this experiment also remarkably shows the application of electrostatic halftoning for artistic stippling. By pronouncing both dominant structures and fine details, the method mimics the works of

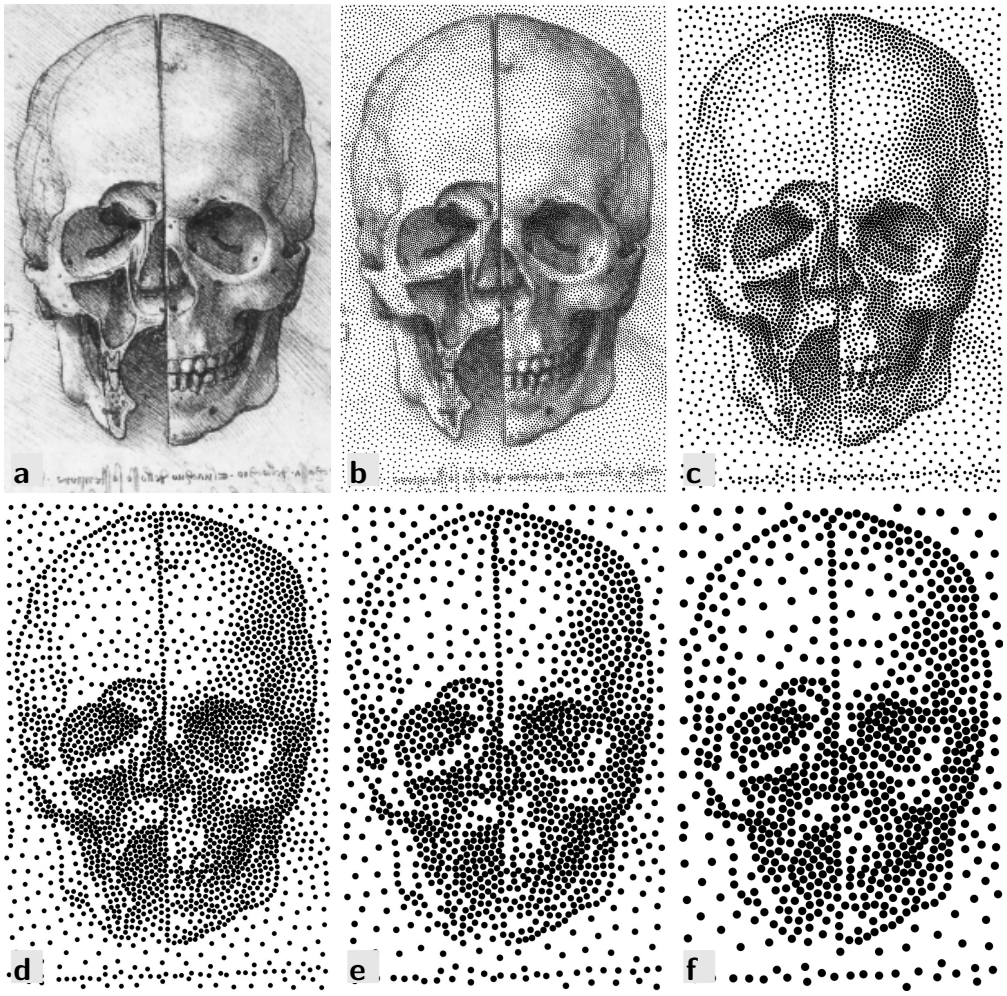



**Figure 7.25**: Electrostatic halftones of the *Skull* image (**a.**) with point radii of **b.** 0.5, **c.** 1.0, **d.** 1.5, **e.** 2.0, and **f.** 2.5 pixels of the original.

painters whose skill enables them to transport important information and impressions with a few dots of paint.

### Grey Value Correction

Let us now evaluate the visual quality gain of images when applying a grey value correction to dark tones (cf. Section 7.4.3). As it turns out, a sound evaluation of this modification is challenging by itself. All changes introduced in this context are tailored to a theoretic model of the rendering operator. From Section 7.2.1, we know that rendering is subject to characteristics of the display or printing device. Hence, in both a hardcopy and a digital version of sample images, the overall impression can differ significantly. This holds in particular since only the 9% darkest tones are affected, and for those a multitude of effects such as dot merge can occur. To this end, the visual results obtained in this experiment can best be observed and evaluated in a digital version of this thesis using a document viewer that supports anti-aliasing for graphics.

Figure 7.26 shows the effect of grey value correction on an image of blast furnaces. While the traditional halftone looks greyish in dark tone areas, the corrected one approximates the original much better. This becomes even better visible if we regard the zoom-ins into the centre of the bottom image boundary. This image region features many details such as openings in the depicted metal structures. Since the contrast of these structures is low and falls into the tonal range a traditional point-based rendering cannot properly visualise, many of these details disappear completely in the uncorrected halftone. Because the corrected halftoning method does not suffer from these restrictions, it accurately represents all fine structures.

Considering that grey value correction has no major impact on the runtime of the algorithm, this extension should always be applied to enhance the quality of the arising halftones. Depending on the printer or display device used in the reproduction of halftones, it can even be adapted to yield comparable results in real-world rendering processes. This works either by a theoretical approach as in Section 7.4.3, or by an experimental optimisation. The latter procedure additionally allows to take into account all kinds of effects described in [Kip01] such as paper structure, suction, or perturbation of the inkblots' trajectories.

### Jittering for Stippling

Let us now evaluate the effect of the jittering extension on stippling results. As it turns out, it is again challenging to find a good evaluation since

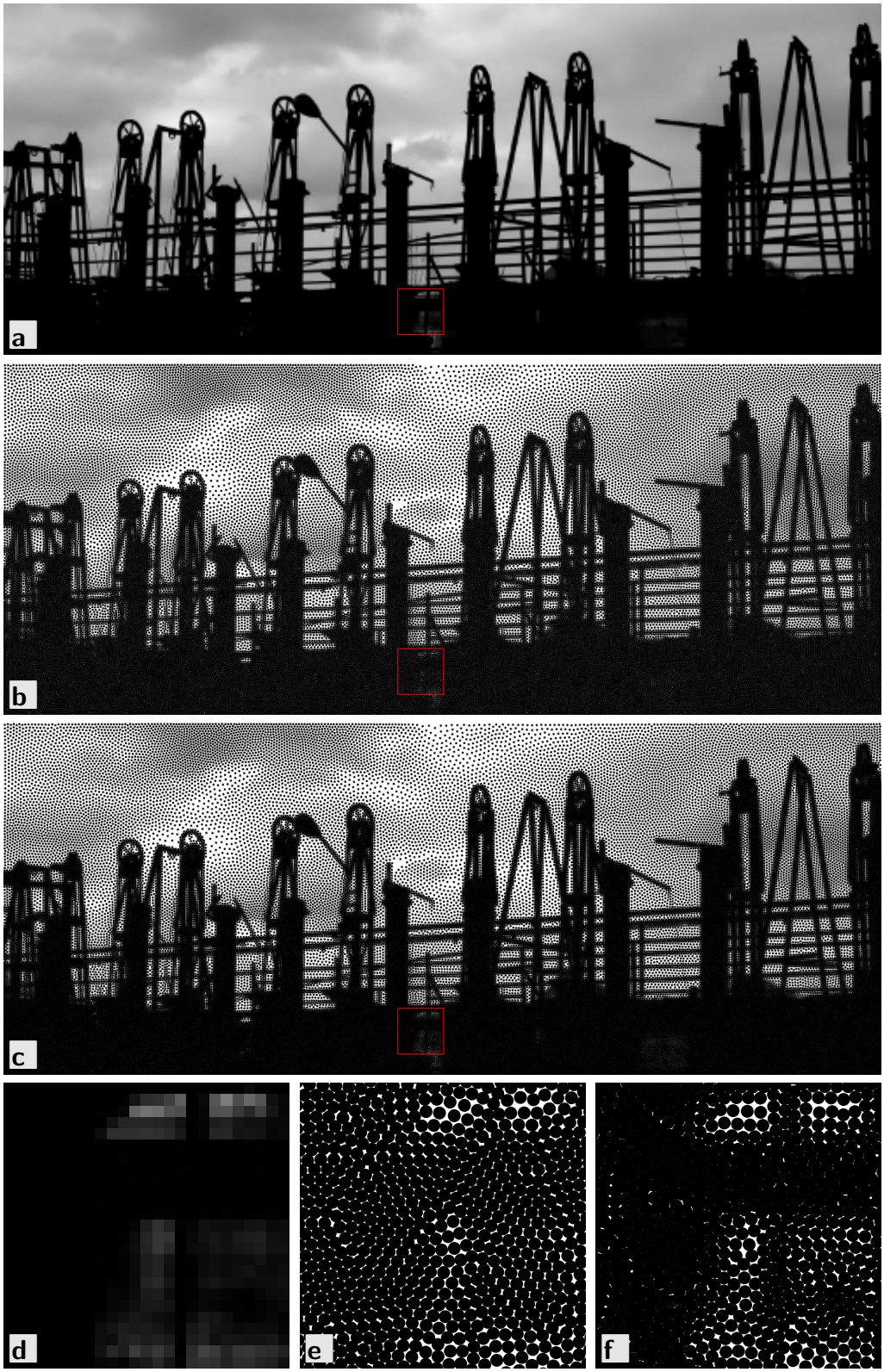



**Figure 7.26**: Effect of grey value correction on dark tones of a picture of blast furnaces at Völklinger Hütte. **a.** Original. **b.** Standard halftone. **c.** Corrected halftone. **d–f.** Zooms to red square regions of a–c, respectively.

the objective is underdetermined. It is clear that we can add an arbitrary amount of jittering to the image, thus obtaining any result from an energetically optimal solution to pure white noise. However, neither of these extremes is desired. While in the first case, the extension has no effect and artefacts remain in the image, the second case creates an image that only preserves the average grey value but is not similar to the original anymore. The tradeoff between these two extremes lies somewhere in the middle and depends on subjective rather than on objective factors.

In order to perform a valid evaluation, we thus add an increasing factor from $\psi_0 = 0$ to $\psi_5 = 0.05$ of jittering to the image, and observe the changes in the resulting halftones. Figure 7.27 shows magnified excerpts from results of this experiment on *Trui*. The result for $\psi_0 = 0$ shows the typical hexagonal structures that arise for electrostatic halftoning, but also the crystallisation artefacts that are caused by this setup. With increasing $\psi$, these structures and artefacts disappear more and more until we end up with a highly irregular, but also artefact-free, approximation of the original image. If we would further increase $\psi$, this point cloud approaches a white noise distribution of particles without any relation to the original image. From the important parameter range covered by Figure 7.27, we can clearly see the local influence of the perturbation. For example, consider the result for $\psi_2 = 0.02$. Here, dark regions are significantly perturbed while the regular structures in brighter regions, such as around the eye, are still well preserved. This is because the relative distance of particles is in bright regions much higher than in dark regions, such that additional perturbation of the electrostatic field has less influence. As a conclusion, let us note that a perturbation as described in Section 7.4.4 is very well suited to remove artefacts, while still preserving the grey value and important structures of the image. Depending on the desired effect, the degree of jittering should be chosen as $\psi \approx 0.02$.

Since this visual analysis is prone to subjective preferences and can only provide a rough qualitative measure, we also perform a numerical evaluation in terms of error measures. Since the desired tradeoff between a good approximation of the original and freedom of artefacts cannot be directly measured, we apply two measures. The approximation quality is best indicated by the peak signal to noise ratio on error images as done in Section 7.7.2. Artefacts can best be seen in a spectral analysis as performed in 7.7.2. The problem is that both measures cannot be applied to the same image: While the first requires a textured image, the second makes only sense on a uniform grey image. We thus need to perform the experiment on two series of results and reason on a combination of the insights acquired by the two experiments.

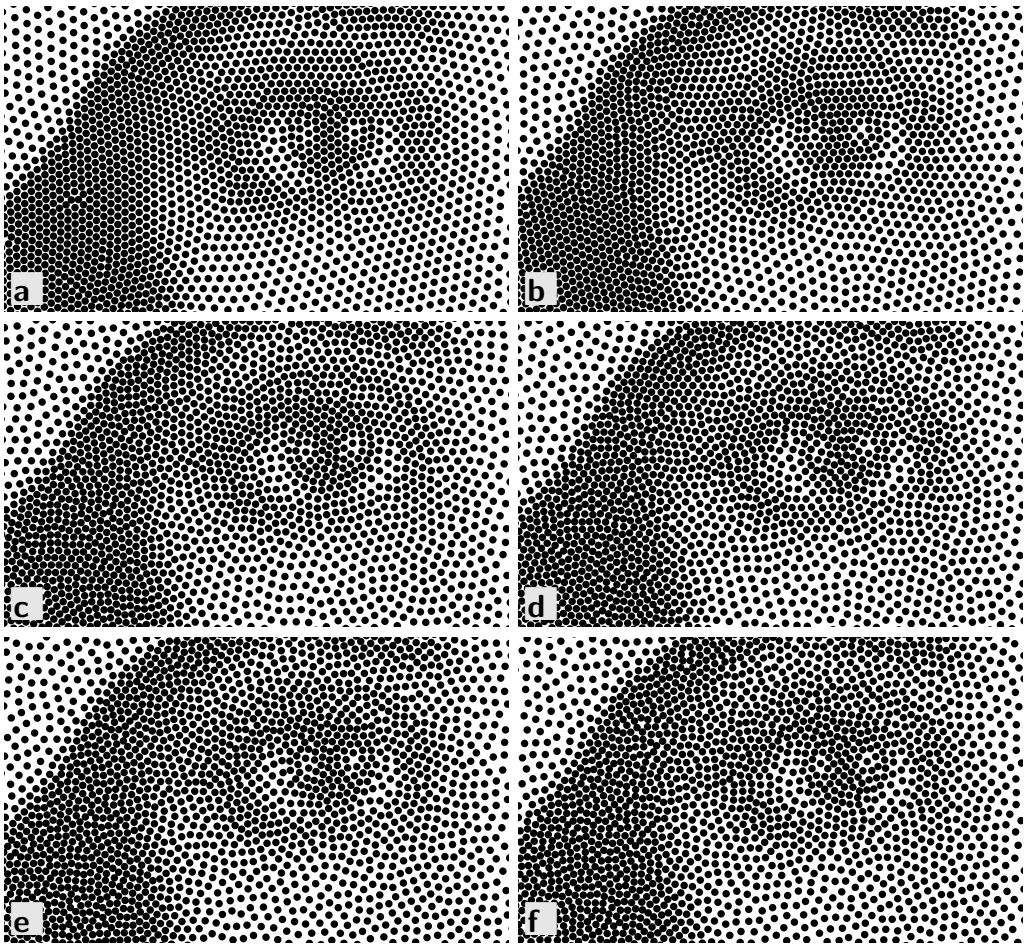



**Figure 7.27**: Excerpt from halftones on *Trui* with **a.** $\psi = 0.00$, **b.** $\psi = 0.01$, **c.** $\psi = 0.02$, **d.** $\psi = 0.03$, **e.** $\psi = 0.04$, and **f.** $\psi = 0.05$.

To this end, let us first compare the PSNR of the *Trui* halftones from Figure 7.27 under Gaussians of varying standard deviation $\sigma$. This is depicted in Figure 7.28. Surprisingly, the well visible randomness introduced to the halftone does not impair its PSNR by too much. This is a consequence of the locality of the added perturbation field. While locally, regular structures are broken up to remove artefacts, the underlying density is still well approximated globally. If we compare jittered electrostatic halftoning to the capacity-constrained approach of Balzer *et al.*, we observe a much higher quality in a global sense and a freely adjustable approximation quality in the range of about $1 < \sigma < 3$. In this local scale, the quality of jittered electrostatic halftoning with $\psi > 0.02$ even drops below the one of Balzer *et al.* with 8 192 points. This corresponds to the scale in which the artifi-

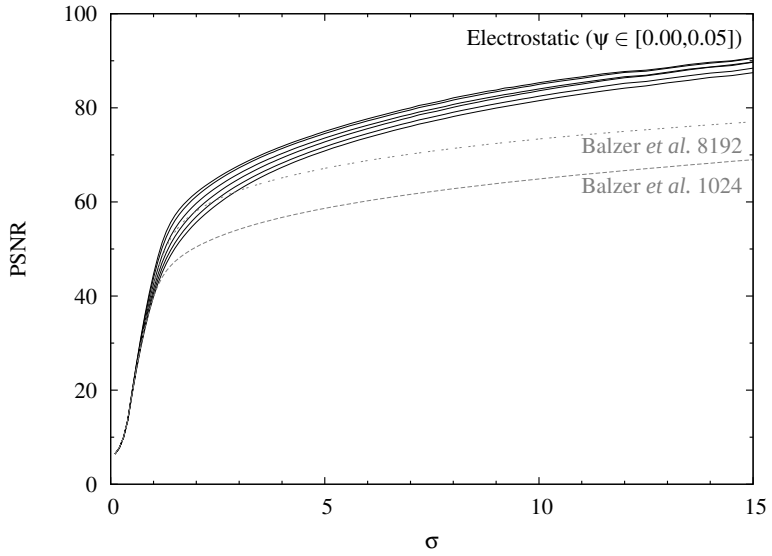



**Figure 7.28**: PSNR of jittered halftones of *Trui* smoothed by Gaussian convolution with standard deviation $\sigma$ with respect to the smoothed original. The method of Balzer *et al.* is given for comparison purposes.

cial perturbations of the electrostatic field live. The fact that we can freely adjust the approximation quality of electrostatic halftoning in this scale indicates that the jittering process works as intended. It represents a major advantage over the inherent parametrisation of the approach of Balzer *et al.* whose quality can only be adjusted by modifying the minimisation process of one particular problem. Hence, electrostatic halftoning allows for any desired jittering in a continuous range, while the capacity-constrained method only allows discrete steps as a by-product of the minimisation process. Finally, let us again remark that electrostatic halftoning performs much better than its competitor when it comes to globally optimal solutions, i.e. for $\sigma \to \infty$. One can even observe that the quality for different choices of $\psi$ converges in this case. This indicates the desired, purely local influence of jittering: Local clusters are broken apart to yield pleasing results, while the overall density is still well approximated in a global sense.

Next, we evaluate the influence of additional jittering to the spectral properties of the results. This is shown in Figure 7.29. Albeit the characteristic peak for the principal frequency in the RAPS drops off rapidly, the system preserves the bluenoise properties of its results very well. This is in particular visible for the results with larger values for $\psi$, where the high-frequency area almost describes a parallel line to the abscissa. If we

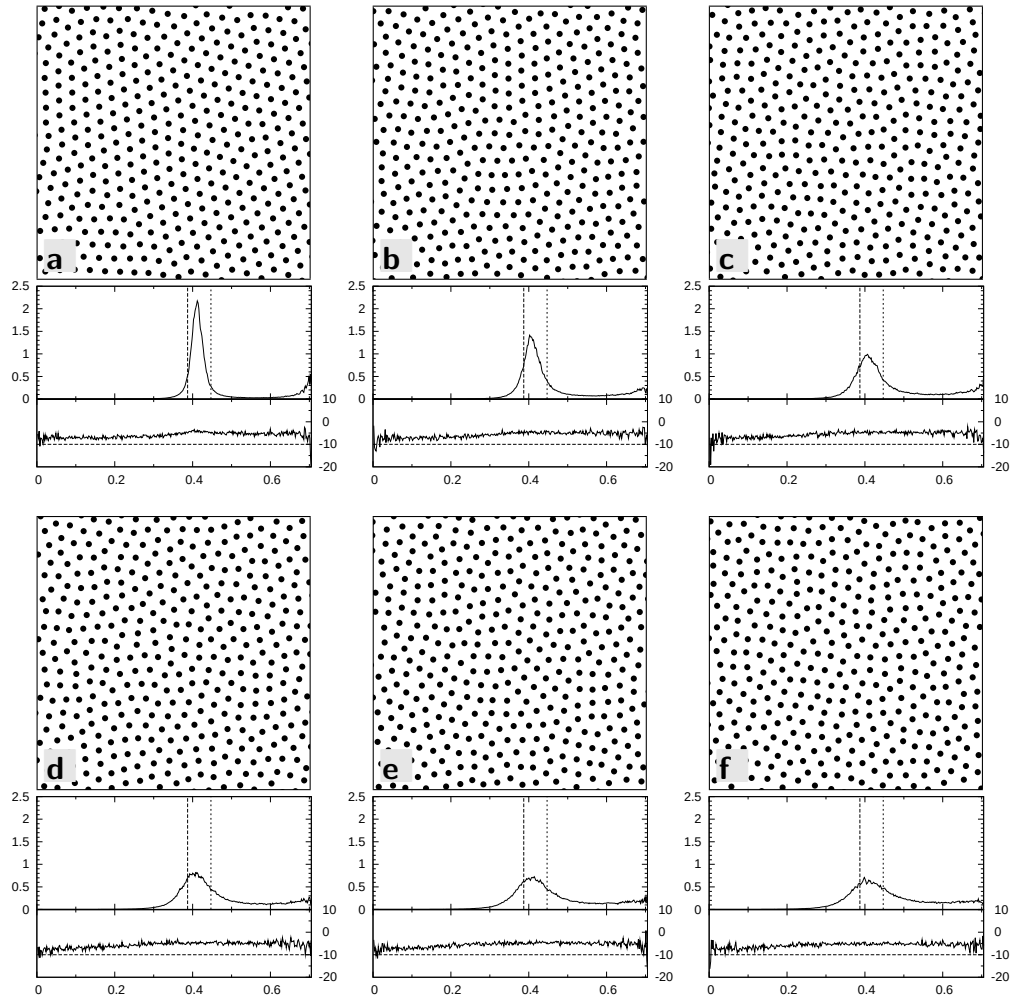



**Figure 7.29**: Spectral analysis of electrostatic halftoning with additional jittering of **a.** $\psi = 0.00$, **b.** $\psi = 0.01$, **c.** $\psi = 0.02$, **d.** $\psi = 0.03$, **e.** $\psi = 0.04$, and **f.** $\psi = 0.05$.

compare these results to the method of Balzer *et al.* (cf. Figure 7.17), we see that electrostatic halftoning performs much better. Even the result for $\psi = 0.05$, which is entirely artefact-free, still has a higher peak at the principal frequency than the capacity-constrained approach. The anisotropy of the method improves up to about $\psi = 0.02$, and does not significantly change for larger values for $\psi$. This indicates that choosing $\psi \approx 0.02$ suffices to smooth out regular patterns arising on halftones of the selected grey tone.

### Edge Enhancement

Let us now evaluate edge enhancement by means of unsharp masking. As was specified in Section 7.4.5, this modification is not applied to the model itself, but comes down to a change of the input image. To this end, we are free to use any image manipulation software to apply unsharp masking, or even to tweak the image by means of other filters. As a simple example, we are now considering a sole application of the unsharp masking filter from the GIMP suite [Ham07].

Different to the mathematical notation from (7.33)-(7.34), the filter allows the user to specify a *radius of effect* $r$ rather than the standard deviation $\sigma$ of the used Gaussian [dJ07]. Both values can be translated into each other by the relation

$$\sigma = \sqrt{\frac{-\hat{r}^2}{2\ln\frac{1}{255}}}\,, \qquad \hat{r} = |r| + 1\,. \tag{7.70}$$

So, a radius $r = 1$ corresponds to $\sigma \approx 0.60$. Figure 7.30 shows the result of edge enhancement on the *Peacock* image. In this experiment, the original is halftoned without edge enhancement, with edge enhancement using $r = 1$ and $c = 1$, and finally using $r = 1$ and $c = 2$. Compared to the original, the classic halftone looks washed-out. This can in particular be seen at the long feathers on the bird's head, as well as in the plumage. In contrast, the edge enhanced version with $c = 1$ does not suffer from these problems and provides a visually better approximation of the image. Characteristics of both homogeneous and textured regions are very well preserved in this case. The last example with $c = 2$ shows that unsharp masking should not be carried to excess. Although the image is again sharper, it also has striking 'halo' artefacts around all objects. However, if further sharpening is desired, such problems can at least partly be remedied by using more complex filters such as morphological operations, or osmosis [Soi99, Wei11d].

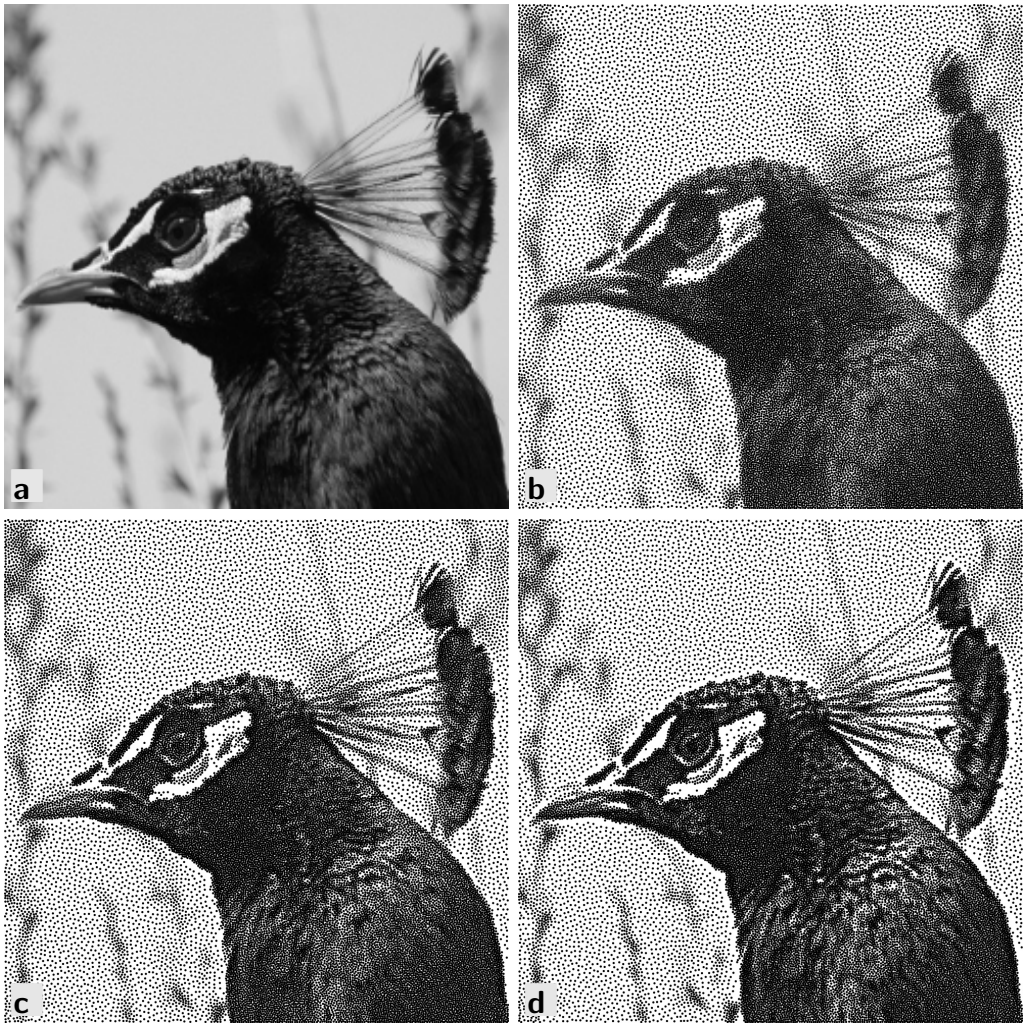



**Figure 7.30**: Edge enhancement by unsharp masking. **a.** Original image. **b.** Electrostatic halftone. **c.** Ditto, with unsharp masking with $\sigma = 0.6, c = 1$. **d.** Ditto, but $c = 2$.

### Colour Halftoning

In this series of experiments, we evaluate and compare the visual quality of colour halftones with a different amount of colours. We regard CMY and CMYK as the two most frequently used colour models for printing, and CMY-RGB-K as a representative for a class of models with a higher colour diversity. In the end, we analyse the applicability of electrostatic halftoning to commodity printers with extraordinary colour models such as CMY-RG-K, or CMY-RB-K.

As a first experiment, we test the halftoning methods on a colour circle. This artificial image is well suited to observe the generation of secondary colours which are not available as single inks. Apart from sampling, this image contains all hues, as well as all graduations towards black. If we assume that secondary colours arise by a mixture of two colours that are closest in the colour spectrum, as well as by black, we find all these combinations in the colour circle. To this end, the halftones created in this experiment tell us whether the halftoning method can reproduce the whole colour space.

Figure 7.31 shows the original, as well as the results obtained by halftoning on the three colour models. All results turn out to represent colours equally well, as can be seen along the boundary of the circle. However, the more colours are involved, the coarser grained becomes the overall appearance. This is particularly visible towards the centre of the circle. The reason for this behaviour is the relatively smaller resolution induced by the non-overlap maps. While for CMY, particles from different channels can overlap and are thus unrestricted in their location, they are required to find a compromise with conflicting channels in the remaining cases. Thus, the CMY model is capable of an up to three times higher effective resolution than a purely disjunct system such as CMY-RGB-K. A good indicator for this issue is the total number of particles involved in halftoning. Assuming a unit area for all particles, the amount of 1 5741 particles for CMY on the colour circle reduces to 10 478 if we use CMYK, and even to 7 857 for CMY-RGB-K. As a consequence, there is always a tradeoff between many printing colours and the minimal dot size the device supports: the more inks, the smaller the dot size must be.

To this end, let us remark that, similar to the evaluation of the grey value correction extension in Section 7.7.3, the visual quality of the results shown in Figure 7.31 depends strongly on the used output device. It is in the nature of this extension that the 4-colour CMYK version of the colour circle has a better quality in a hardcopy of this thesis than on a screen, while it is the other way round for the CMY version. Another good example for this device-dependency can often been spotted in the CMY-RGB-K circle

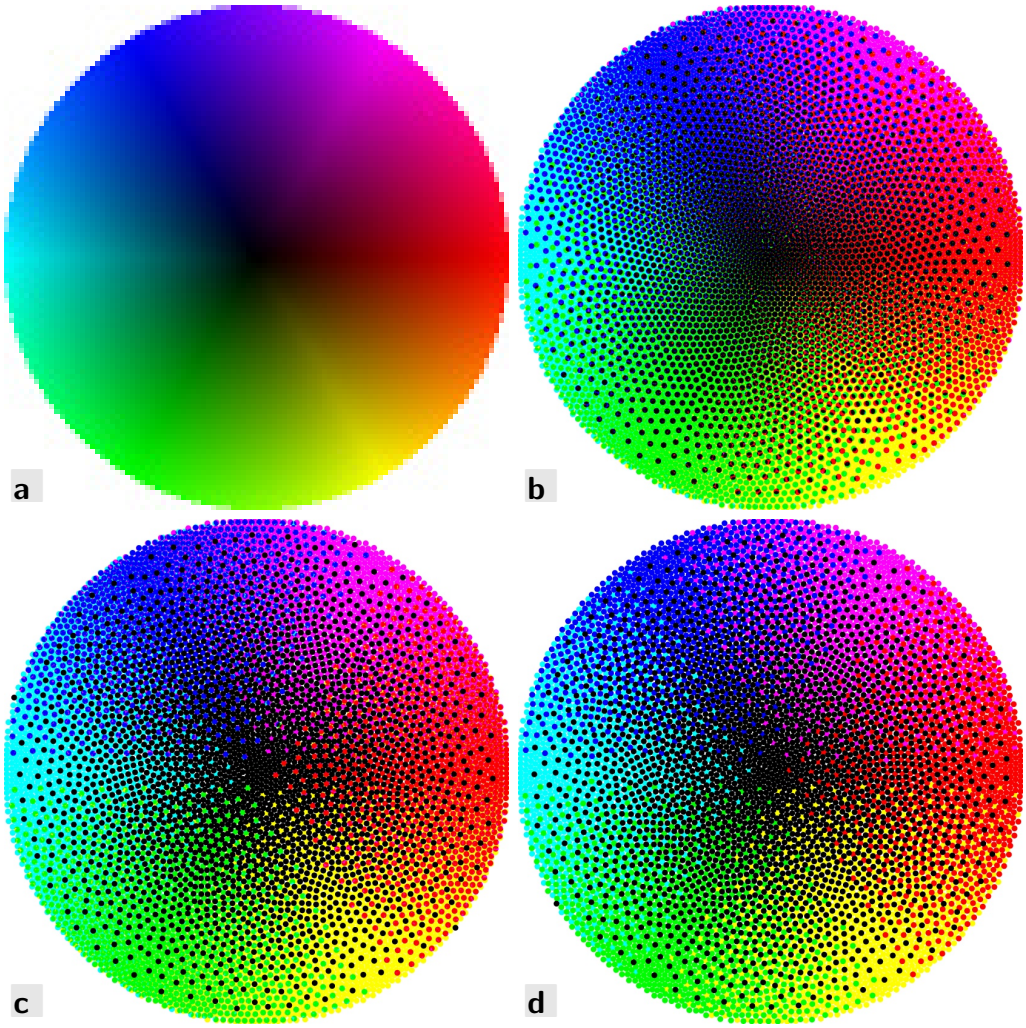



**Figure 7.31**: Comparison of halftoning with different sets of colours on a colour circle ($100 \times 100$). **a.** Original. **b.** 3-colour (CMY). **c.** 4-colour (CMYK). **d.** 7-colour (CMY-RGB-K).

which often seems to have darker shadows from the centre to the red and blue borders, or to the red and green areas (depending on the device). Hence, we should always keep in mind that colour halftoning is designed with respect to an imagined perfect rendering function. A fair comparison of different colour systems with the same output device is thus technically impossible. Still, the properties of the presented halftones provide a good hint of the appearance and quality on an actual device.

The colour circle features many colour gradients but no edges or homogeneous regions. As a next experiment, we are thus interested on the behaviour of the different colour halftoning variants on an image that contains such information. The insights gathered in this experiments are representative for many real-world images. Figure 7.32 shows the *Comic* image, as well as halftones of it using the CMY, CMYK, and CMY-RGB-K colour models. While the images look very similar from distance, they differ significantly on a small scale. This is best visible in the zoom-ins. While the CMY result has the overall smoothest appearance, it also lacks a sharp contrast along the black contours in the image. The reason for this is that edges can be dislocated depending on the concentration of particles in the neighbouring areas, thus blurring out the edge. This is not only an issue for cartoon-like images. The more colours are used in printing, the more likely it is that two sides of a prominent edge are halftoned with two different inks, thus creating a sharp transition. However, while such sharp transitions are desired along edges, they are to a certain degree also present as artefacts in flat areas. A good example for this issue is the purple area in the zoom-ins. The CMY screen overlays the three channels independently such that colour information can be distributed uniformly. In contrast, the CMY-RGB-K halftone enforces a non-overlapping distribution of colour particles which creates a much coarser impression. Taken to the extreme, this separation can even generate artificial salient features, such as the yellow inkblots in the green shirt. This additional yellow colour is required to correctly represent the light green tone of the region and it is included in all presented halftones. However, in the CMY and CMYK models, these occurrences are not clustered in single dots, but uniformly distributed over the whole area.

This overall increased coarseness for a higher number of inks is also reflected in the approximation quality with respect to the original image. Similar to our experiment performed in Section 7.7.2, the graphs in Figure 7.33 show the peak signal to noise ratio (PSNR) between the halftone smoothed by a Gaussian with standard deviation $\sigma$, and the smoothed original. The PSNR on colour images is obtained as the average over all channel-wise measurements. For all $\sigma$, the CMY-based model has the highest PSNR, while CMYK and CMY-RGB-K are on second and third place.

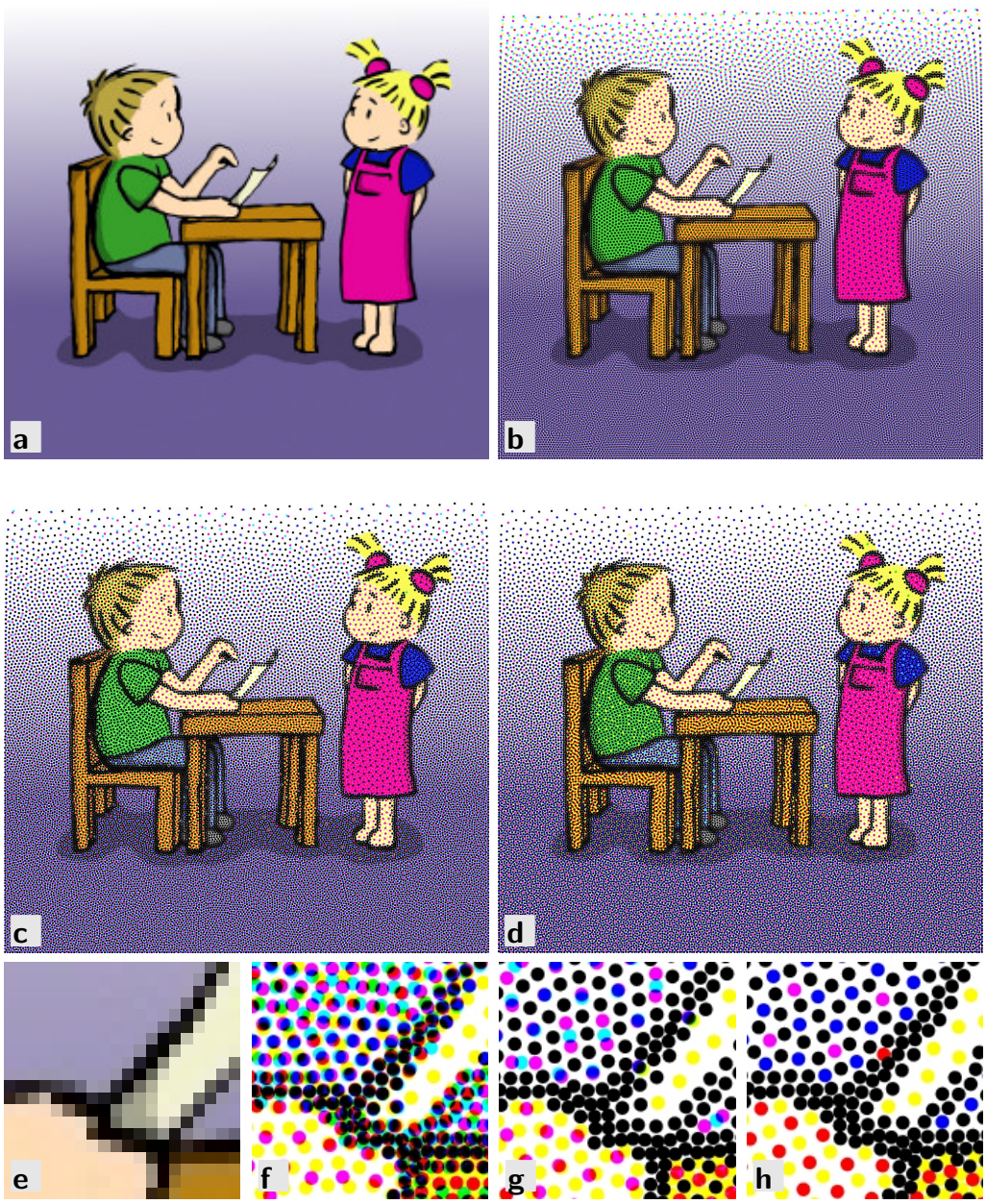



**Figure 7.32**: Comparison of halftoning with different sets of colours on *Comic* (256 × 256). **a.** Original. **b.** 3-colour (CMY). **c.** 4-colour (CMYK). **d.** 7-colour (CMY-RGB-K). **e–h.** Zooms into the centre regions of a–d, respectively.

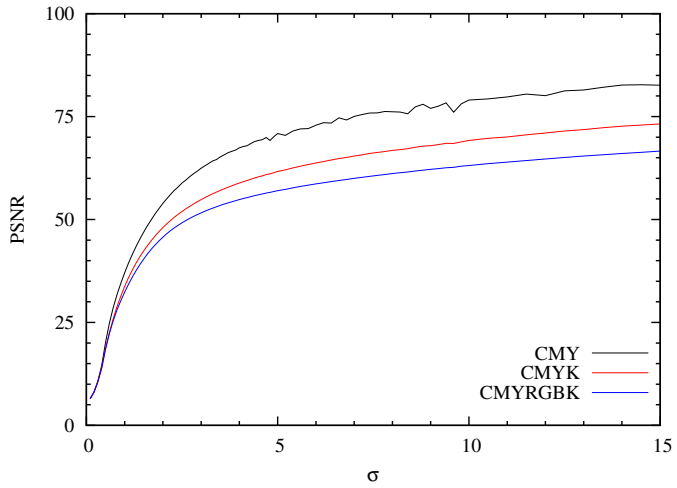



**Figure 7.33**: PSNR of the results from Figure 7.32 for *Comic* smoothed by a Gaussian $K_\sigma$ against the smoothed original.

This confirms our observation that this variant makes use of its higher degree of freedom in placing particles onto the image plane, and in the higher average number of particles per channel. The fact that the graph for CMY is more noisy than the others must be an artefact of the evaluation rather than the method, since the same halftone was used for the whole evaluation. It might be that due to the high number of points, the floating point accuracy did not suffice to properly accumulate the error up to all necessary decimal places, which is reflected in a visual jitter on the logarithmic scale.

Finally, let us test the applicability of multi-colour electrostatic halftoning for real-world applications. Although it lies beyond the scope of this thesis to perform experiments on real printers, we can at least take some off-the-shelf printers as an example, assume they reproduce colours as we see them on the screen, and compute halftones for the subset of inks that they support. To this end, we compare the following halftone types against each other [Kip01, CR11]:

- Early low-budget desktop inkjet printers such as the Hewlett-Packard Deskjet 550c supported the three-colour **CMY** model. If the user wanted to print in colour, he had to exchange the black-only cartridge by a CMY-only cartridge. Today, these printers are largely squeezed out of the market by modern four-colour printers, and have been relegated to a niche existence for pocket printers such as the Hewlett-Packard PhotoSmart A310.

- The **CMYK** model is among the most frequently used colour models,

and can be found in all sorts of printers from cheap inkjet printers such as the Canon Pixma i560, via business-scale colour laser printers like the Kyocera FS-C5200DN, up to mass-production printing machines.

- Recently, more and more 'asymmetric' colour models find their way into the desktop printer market. By adding additional colours such as red, green, or blue to the classical spectrum, the manufacturers try to extend the colour gamut for a more 'brilliant' colour impression. While Canon frequently provides additional red end green inks for its printers such as the Pixma iP8500, Epson usually supports additional red and blue inks in its professional level printers such as the Stylus Photo R1800. Apart from this, suppliers often add different inks for matte and glossy black (Canon), or transparent glossy finish (Epson). However, the arising redundancy can easily be resolved by the context on the page. While glossy colour is preferred for photographs, matte colour is typically used to print text. Hence, we only focus on the theoretical concepts of the colour models **CMY-RG-K** and **CMY-RB-K**, but do not go into detail about the technical realisation.

- Finally, we also repeat this experiment for the **CMY-RGB-K** model which is the standard for high-end printing machines for large scale production.

Before we see the results of the experiment, let us briefly remark how to obtain 'asymmetric' colour models. We start with an interaction matrix as for the 'full' CMY-RGB-K model, and delete the columns and rows for the colours which are not present. In addition, we must allow these missing colours to be approximated by subtractive colour mixing. This results in a zero entry for primary colour pairs which are able to reproduce the missing secondary colours. The interaction matrices resulting from these considerations are given in Figure 7.34.

The results of this experiment are shown by Figure 7.35 on an image of hands with a Rubik's cube. Viewed on a computer screen, all of these results approximate the image very well. As a critical aspect for real printing purposes, the hues of the saturated patches on the cube, as well as of the hands are very well preserved. One should note that this property is not self-evident. Other techniques often suffer from tonal shifts, in particular if the different colour layers are processed and printed independently [SGBW10, Kip01]. The two 'asymmetric' halftones are expectedly more coarse than the CMY halftone due to the much smaller number of particles, but do not exceed the coarseness of the CMY-RGB-K halftone. This shows that electrostatic halftoning is very well suited as a preprocess-

**a**

| | C | M | Y | R | G | K |
|---|---|---|---|---|---|---|
| **C** | 2 | 0 | 1 | 1 | 1 | 1 |
| **M** | 0 | 2 | 1 | 1 | 1 | 1 |
| **Y** | 1 | 1 | 2 | 1 | 1 | 1 |
| **R** | 1 | 1 | 1 | 2 | 1 | 1 |
| **G** | 1 | 1 | 1 | 1 | 2 | 1 |
| **K** | 1 | 1 | 1 | 1 | 1 | 2 |

**b**

| | C | M | Y | R | B | K |
|---|---|---|---|---|---|---|
| **C** | 2 | 1 | 0 | 1 | 1 | 1 |
| **M** | 1 | 2 | 1 | 1 | 1 | 1 |
| **Y** | 0 | 1 | 2 | 1 | 1 | 1 |
| **R** | 1 | 1 | 1 | 2 | 1 | 1 |
| **B** | 1 | 1 | 1 | 1 | 2 | 1 |
| **K** | 1 | 1 | 1 | 1 | 1 | 2 |

**Figure 7.34**: Interaction matrices for **a.** the CMY-RG-K model, and **b.** the CMY-RB-K model.

ing stage for real printing processes, independent of the colour model that the printer hardware supports.

### Second Order Screening

Let us now have a look on the results of the second order screening extension that was discussed in Section 7.4.7. Important quality criteria for halftoning are the good representation of all grey values and the rotational invariance of the method. To this end, we take the image of a 2-D Gaussian (depicted in Figure 7.36) and compare conventional halftones of it against a second order result. While the two classical results are assembled of points with area $A_S = 1$ and $A_L = 3$, respectively, the second order result consists of a mixture of these two classes in the ratio $w_S = 2/3$ to $w_L = 1/3$. Figure 7.36 shows the three halftones together with zoom-ins into the centre regions.

Let us first regard the halftones from a normal distance. Visually, the second order screen looks very similar to the classical result with small particles. This is due to the fact that many particles are still chosen from the set of small particles while the few large particles smoothly integrate into this concept without creating striking artefacts. This changes slightly if we consider dark regions, as shown in the zoom-ins. Here, the combination of both particle classes cannot fill spaces as good as if only small particles are used. Instead, the blank area arising from this setup is about the same as for the large-only configuration. To this end, the result obtained by second order screening lies qualitatively in between the two extremal cases of small and large dots. This meets our expectations and requirements of the method. Moreover, all results show no directional bias. Since the original image is rotationally symmetric, this fact hints at a good rotational invariance of electrostatic halftoning, and in particular of second order screening.

As a next experiment, we consider the application of second order screen-

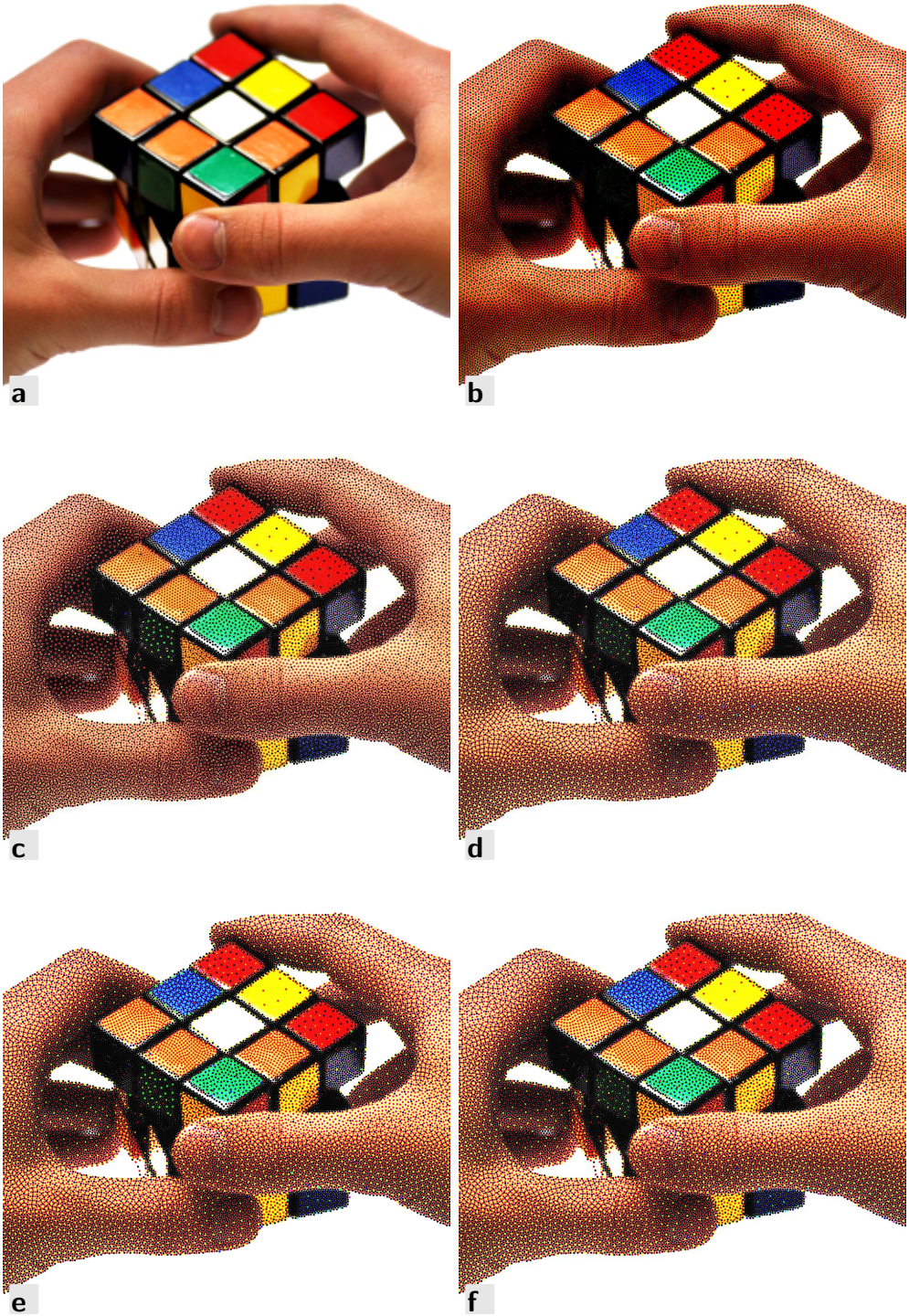



**Figure 7.35**: Comparison of halftones for different colour models that are used by real printers. **a.** Original, **b.** CMY, **c.** CMYK, **d.** CMY-RG-K, **e.** CMY-RB-K, **f.** CMY-RGB-K.

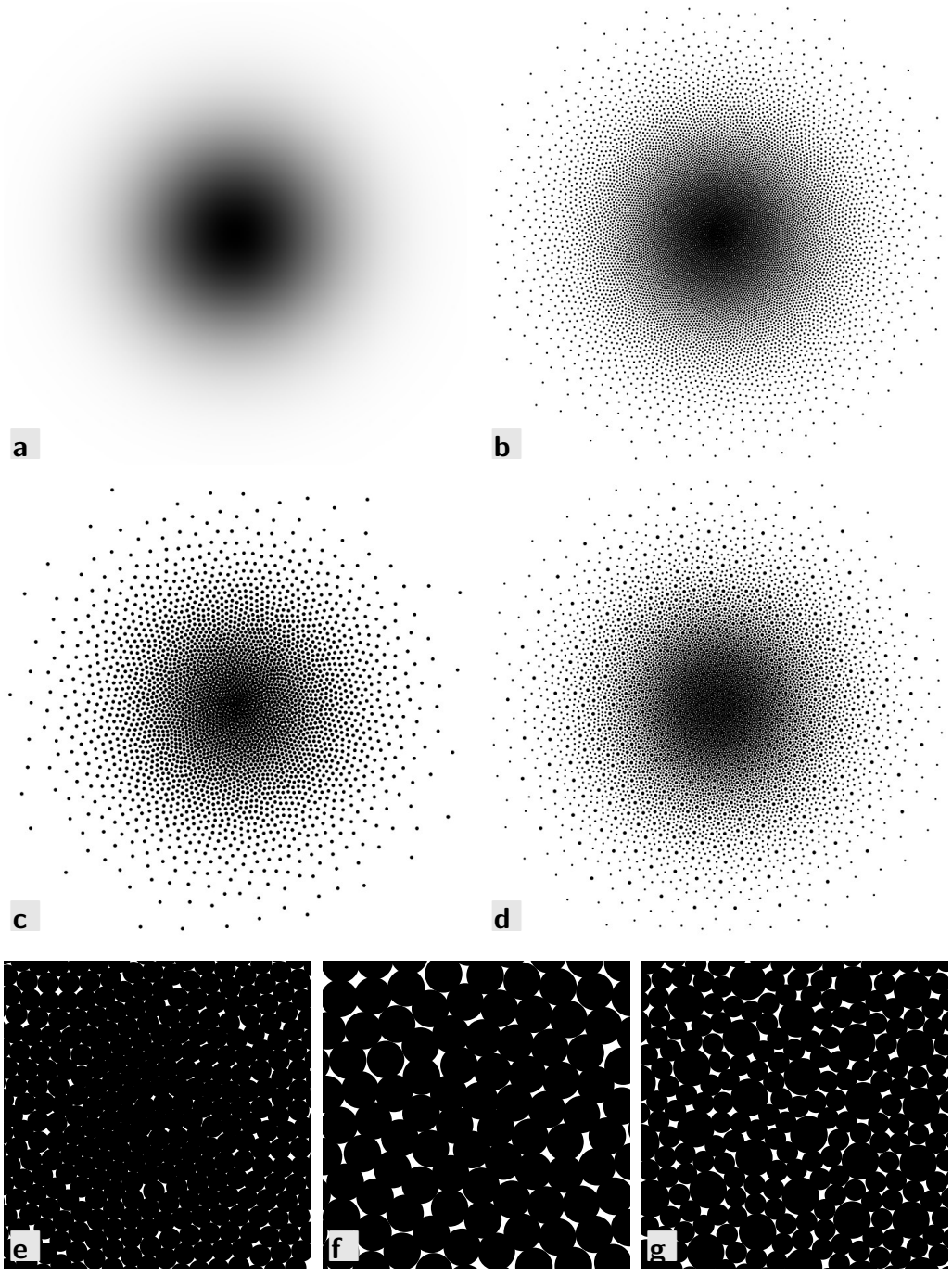



**Figure 7.36**: Second order screening with particles in two sizes. **a.** Gaussian kernel, halftones with **b.** small dots, **c.** large dots, and **d.** both types simultaneously. **e–g.** Zoom-ins into centres of b–d, respectively.

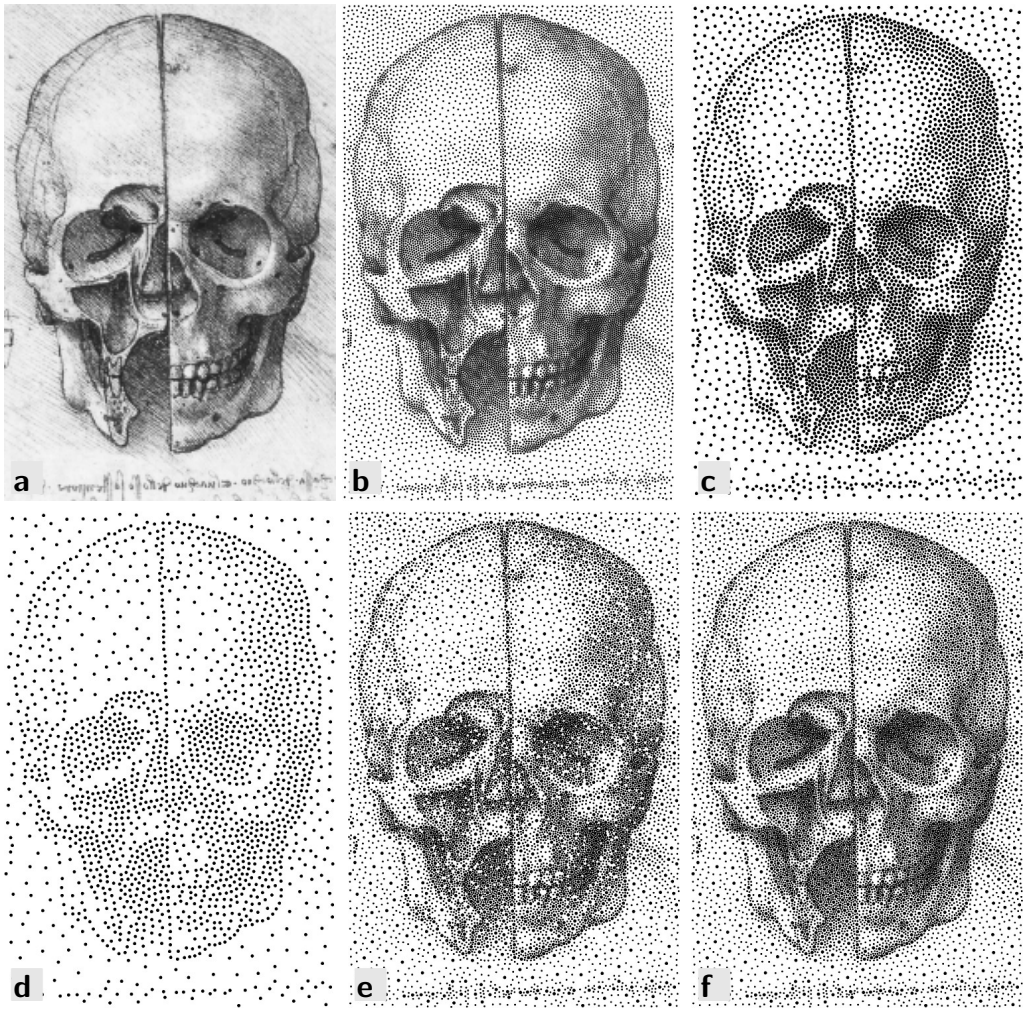



**Figure 7.37**: Second order screening with particles in two sizes. **a.** Original image. **b–c.** Electrostatic halftones with small and large particles, respectively. **d–f** Second order screen: Only the large particles, additional random 90% of small particles, complete result.

ing to an image which contains flat regions, slopes, and textures. This gives a good impression of the behaviour of second order screening on real-world images. Moreover, we are going to test the quality of the result considering random dot loss. Thus, we also gain insights about the applicability of this technique to real printing processes.

Figure 7.37 shows the image *The skull bisected and sectioned* by Leonardo da Vinci. As for the previous experiment, the original was

halftoned with small (1) and large (3) dots, as well as by second order screening using both classes in the ratio 2/3 to 1/3. The overall visual impression is very similar to the one obtained in the previous experiment. Compared to the small-only rendering, the presence of large dots does not noticeably impair the quality of the result. This is in particular visible in regions with fine structures, such as for the cracks at the side of the skullcap. These fine lines are well-preserved in both the small and the second order rendering, while they vanish completely in the result with large particles.

Surprisingly, this property even carries over if we assume that the printing device is subject to dot loss. In Figure 7.37 (bottom centre), the second order screen has been randomly thinned by 10% of all small particles. Consequently, this image looks noisier than the full screen. However, compared to the result obtained with an exclusive use of large particles, more small structures are still visible. As an example, consider again the fine cracks in the skullcap, the handwriting at the bottom of the image, or the shading in the opened paranasal sinus. This shows that it is still beneficial to use a second order screening method while accepting the loss of a certain amount of small particles, rather than printing only larger dots that a device is able to reproduce reliably. This holds even more if we create a halftone for a whole class of printers out of which only a few are subject to dot loss at a particular scale. Remember that dot loss is often related to the darkness of a neighbourhood. Although points were selected by a uniform distribution over the image, missing particles in dark areas are perceived much more striking than in bright areas — where they appear much more frequently in real printing processes.

In order to understand the advantageous behaviour of this second order screening extension, consider the bottom left part of Figure 7.37. As an extremal case, all small particles have been removed. Still, the remaining large particles contain most of the features present in the traditional halftone with the same particle size. Hence, small particles are only needed to preserve the average grey value, and to complement this very good result by the fine structures that cannot be represented at this scale. Whenever small particles get lost, details disappear, the image becomes noisy, and the average grey value is changed. However, the overall impression of the image is preserved.

As a next experiment, we check whether this impression can be confirmed by a numerical analysis. In the same spirit as done in Section 7.7.2, we convolve the two two classic halftones and the full second order screen from Figure 7.37 with Gaussian $K_\sigma$ of different standard deviations $\sigma$. We compare them to the blurred original. Figure 7.38 depicts the PSNR of the difference. As expected, the graph for the second order screen lies between

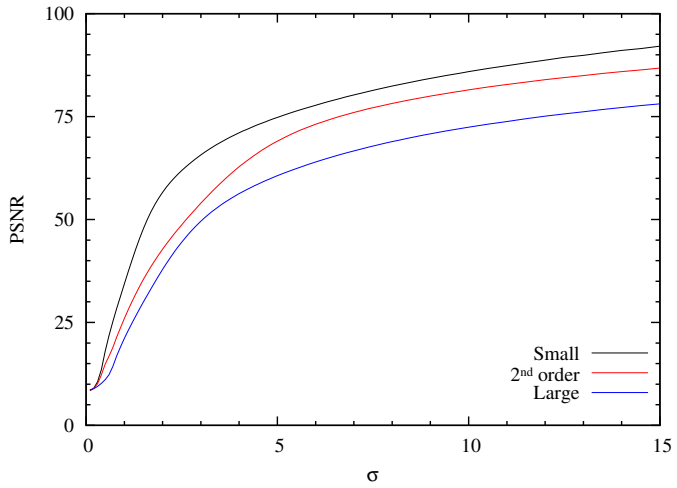



**Figure 7.38**: PSNR of halftones for the skull image smoothed by a Gaussian $K_\sigma$ against the smoothed original. Depicted are halftones with small and large particles, as well as a second order screen with both types in the ratio $^2/_3$ to $^1/_3$.

the graphs for the traditional halftones. This result indicates that second order screening does not introduce errors other than those arising from the particular dot sizes. On medium and large scales, second order screening is closer to halftoning with small dots than to halftoning with large dots. This confirms our impression that fine structures are much better preserved than possible by only using large dots. However, this is different for small standard deviations. In these cases, the local variation introduced by the few larger particles dominates the representation such that the second order screen is only slightly better than a halftone with large dots.

Let us now explore the effect of the priority weight $C$ (see Figure 7.5) on the quality of the result. To this end, we halftone an image of a tonal ramp from black to white and observe the change from $C = 1$ via $C = 2$ to $C = 4$. In addition, we see what happens if we set a similar prioritisation to the small particles, i.e. weight them with $C_\mathrm{S} = 2$. This is shown in Figure 7.39. In order to see the differences between the variants better, it additionally depicts excerpts for small and large particles from each result.

In a direct visual comparison, the version with $C = 2$ and no prioritisation of small particles provides the best quality. The overall result looks smooth, large particles are quite uniformly distributed, and the arising gaps are homogeneously filled. This changes if we decrease $C$ to 1. Here, large particles are no longer forced to distribute evenly, such that they form striking clusters in all parts of the image. The other extreme, increasing $C$, causes small particles to cluster. Because this change happens on a much

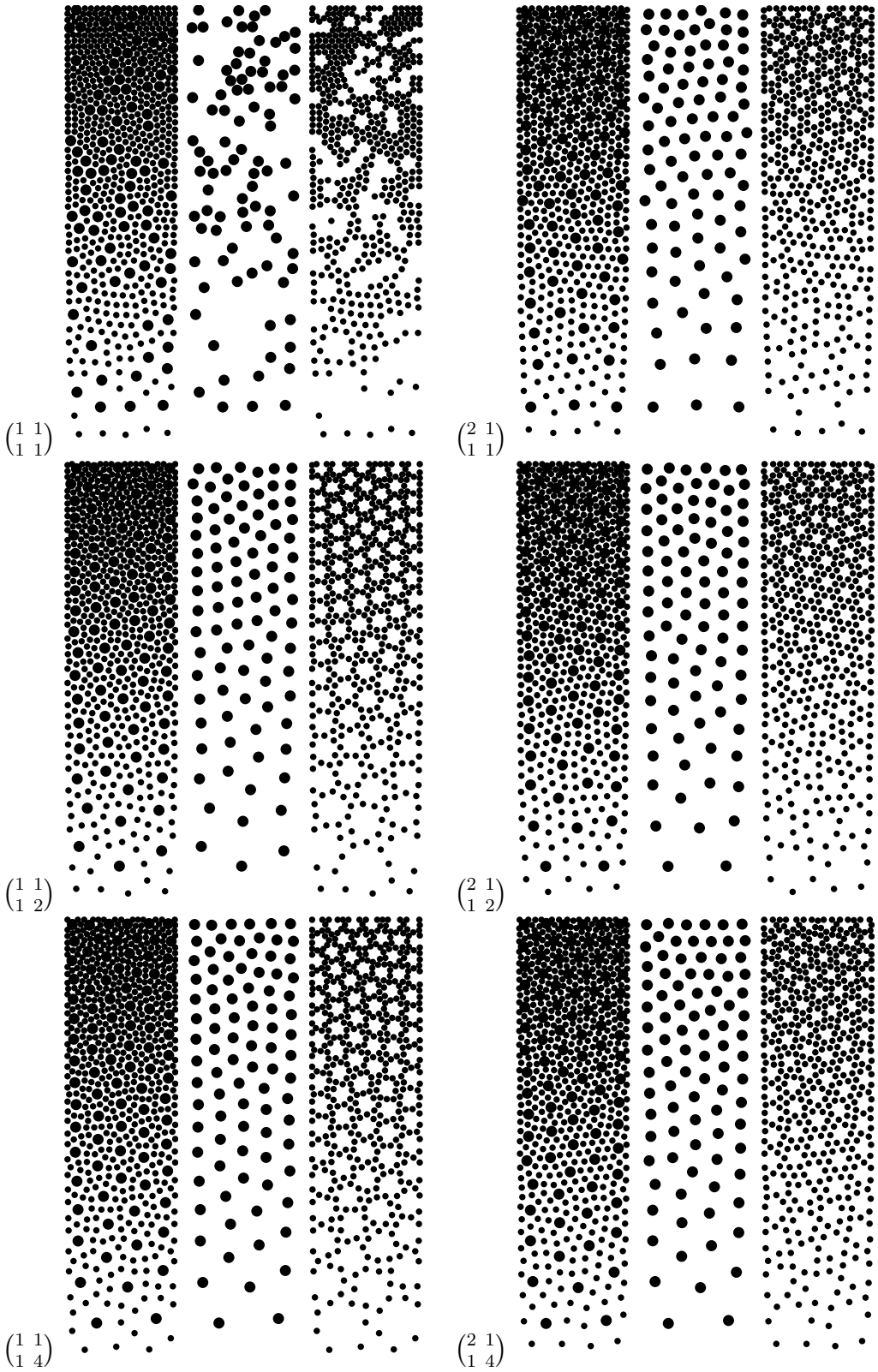



**Figure 7.39**: Effect of weights on the main diagonal of the interaction matrix. Each experiment is depicted with its pointwise multiplication matrix, the full result, and the excerpts for large and small particles, respectively.

smaller scale, these artefacts are not as striking to the human eye. Still, we can observe 'unnatural' structures in the midrange area for $C = 4$. In order to avoid such fine clusters, one could argue that it makes sense to introduce an additional weight to distribute small particles more evenly. This is shown in the right column of Figure 7.39. Although this modification indeed remedies the problem, it introduces new artefacts by aligning small particles too closely around larger ones. Such 'cogwheel'-like structures arise independently of the weight $C$ on the large particles. Albeit this appearance could be desired for specific applications, it creates artificial dot overlaps which cause a grey value to be rendered too bright. Moreover, we see that whenever small particles are priorised, large particles are no longer aligned in an optimal pattern (cf. top right subfigure). Since these larger discs dominate the visual impression of an image, such irregular structures lead to a more noisy appearance.

Finally, let us have a look on multi-class second order screening. As an example, we consider three classes of particles, and apply an interaction matrix as in Figure 7.6. The small particles occupy $w_{\mathrm{S}} = 2/3$ of the overall black area, the medium class $w_{\mathrm{M}} = 2/9$, and the small class $w_{\mathrm{L}} = 1/9$. This ratio was chosen because of aesthetic reasons. A smaller partition of large particles introduces a wrong saliency, while a larger amount does not leave enough freedom for small particles to fill the arising spaces. Still, it remains to evaluate which sizes the particles should have to obtain pleasing results. Figure 7.40 shows examples for the skull image from Figure 7.37. In order to understand the problems arising for small and for large variations, we consider one average choice and two extremal cases. To this end, we use dots with areas $(A_{\mathrm{S}}, A_{\mathrm{M}}, A_{\mathrm{L}}) \in \{(1.0, 1.5, 2.0), (1.0, 2.0, 4.0), (1.0, 3.0, 6.0)\}$. In Figure 7.40, we also see magnifications to the teeth of the skull. This gives us a good basis of evaluation for both uniform and textured regions.

If all particles differ only marginally in size, they assemble results of a comparable quality as in the classical framework (cf. Figures 7.37, 7.40). This property makes such a setup interesting for applications in which a different point size is desired for purely technical reasons and where differences should not be spotted by the observer. However, the true strength of second order screening arises usually for sets of particles with a higher variation in size. Hence, we additionally analyse a setup in which the huge particles have a 6 times larger area than the smallest ones. As it turns out, such a scenario poses new problems, as is shown in Figure 7.40: striking gaps occur around large particles. This is a not an artefact of the algorithm but a fundamental problem which cannot even be remedied by choosing different amounts of particles. In order to represent the average grey value, each particle is surrounded by a white ring whose size depends on the grey value

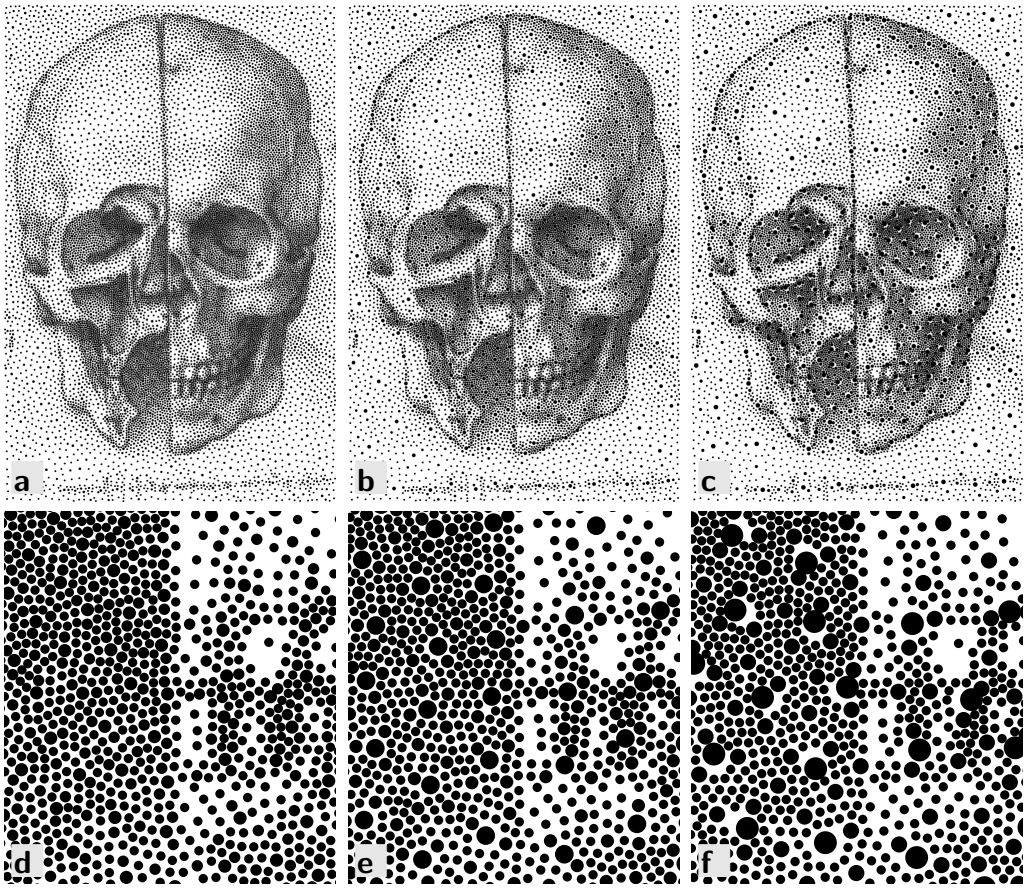



**Figure 7.40**: Second order screening with three classes covering $^2/_3, ^2/_9, ^1/_9$ of the total area. **a.** Little variation in sizes (1.0, 1.5, 2.0). **b.** Medium variation (1.0, 2.0, 4.0). **c.** Large variation (1.0, 3.0, 6.0). **d–f.** Zoom-ins into mouth region of a–c, respectively.

of the underlying image and on the size of the particle. This can easily be understood if we consider a grey value of 0.5 and assume particles to align in a honeycomb pattern. The ring as the difference between the imagined hexagonal cell and the particle must cover the same area as the particle itself. This basic property is the reason for the artefacts we observe. If small particles and large particles meet, the white ring belonging to a large dot touches a dense area of small particles. With respect to them, however, the space is much to wide. This creates the illusion of a supposedly wrong representation of the original image.

Note that this problem is further intensified in dark regions, where different grids for medium and large particles clash. This is well visible in the

left half of the magnification in Figure 7.40. Since these artefacts are very similar to the 'cracks' discussed in Section 7.4.4, we can again apply a perturbation to partly remedy this sub-problem. Alternatively, we can modify the ratios of classes or the priority weights in the interaction matrix. However, any of these changes impairs the measurable quality by introducing additional noise, but cannot solve the underlying problem.

As a consequence of this experiment, the variance between the sizes of the particles should always be kept within a certain range. One example for such a setup is given by the choice $(1, 2, 4)$ in Figure 7.40. On the one hand, we have particles that can clearly be distinguished in size. On the other hand, the overall result still looks smooth and largely free of noise. The insights gathered in this experiment are again of great importance when it comes to second order screening with more than three classes. As we know that we cannot distinguish particles if they do not differ significantly in size, and that the range between smallest and largest is limited, it does not make sense to extend second order screening to an arbitrary number of classes. This holds in particular as each class is required to find a relatively regular grid structure, and these grids are likely to clash. Thus, a higher number of classes usually poses additional challenges rather than creating better results.

### Multi-Class Sampling

Multi-class sampling fundamentally relies on the parameter $\varphi$ (see (7.36) and (7.39)) which describes the balance between a 'good' distribution of the whole point set and a 'good' distribution of each subset. In Section 7.4.8, we have seen that these constraints are mutually exclusive and cannot be fulfilled simultaneously. Instead, we can only find an optimal compromise in which both constraints are least violated. From (7.39) in Section 7.4.8, we have the intuition that we should chose $\varphi = 1$. Then, the weighted block diagonal matrix and the non-overlap map have the same influence such that the whole set should be equally optimised to any of the subsets. Let us confirm this assumption by an experiment. Since $\varphi$ describes a ratio, we account for this fact by choosing values from $\{0, 0.2, 1, 5, 100\}$. In order to keep the system simple, we only regard two subsets, each containing only one size of particles, and consider a uniform image.

The results of this experiment are shown in Figure 7.41. For $\varphi = 1$, we obtain a homogeneous distribution for both the overall set of points and each of the two subsets. However, this converged solution does not reveal the typical hexagonal structures that we know from standard halftoning, and which we expect in this case as well. Instead, the two colours arrange

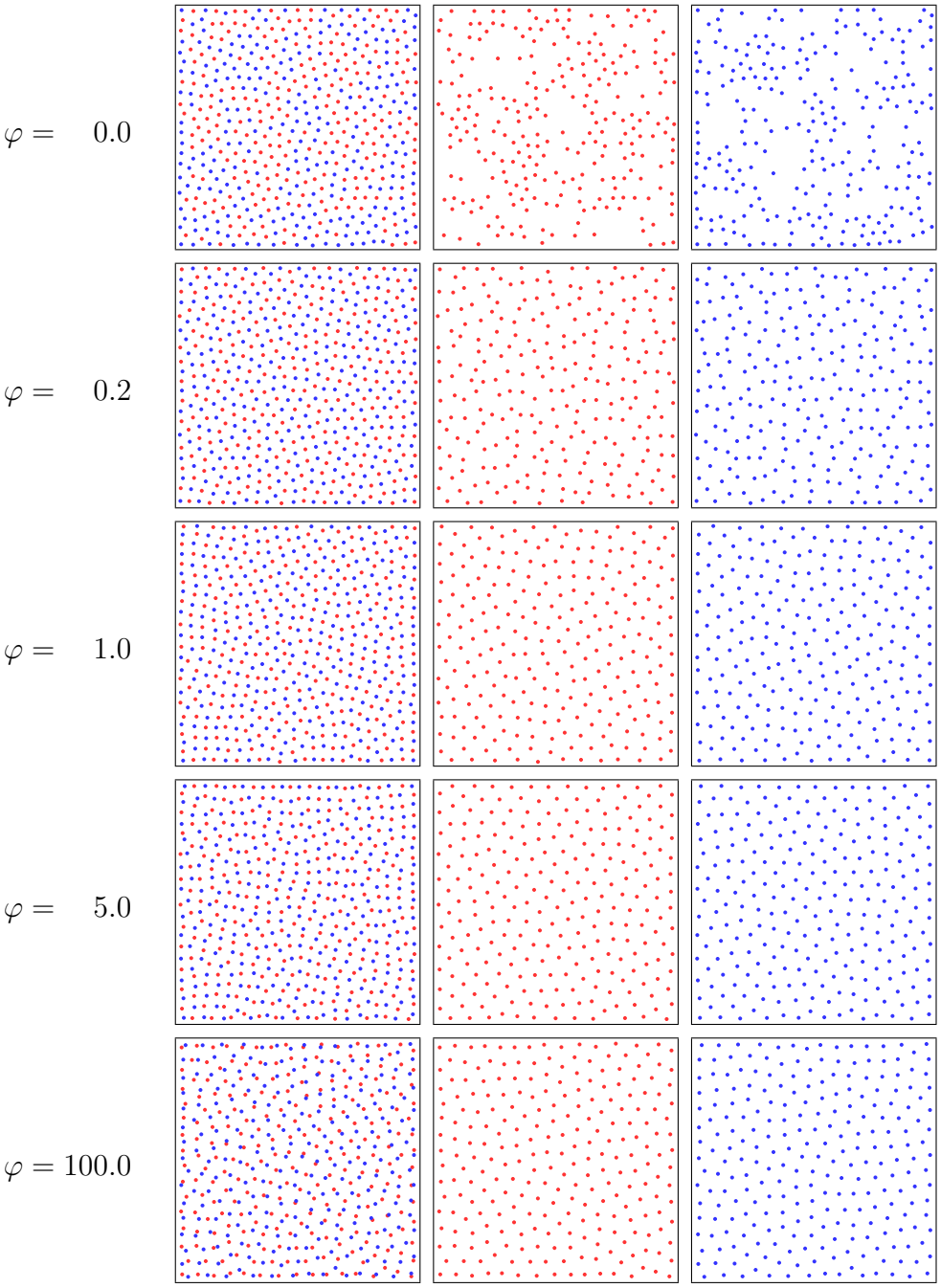



**Figure 7.41**: Influence of the weight $\varphi$ on single order multi-class sampling of a uniform image. Small $\varphi$ cause clusters in individual uniform subsets, while large $\varphi$ generate clusters in the total set.

in a rectangular checkerboard pattern. Such arrangement is neither optimal for either of the individual sets nor the overall set, but it seems to represent the best compromise.

If we chose $\varphi$ much smaller, the overall set is better approximated. Among the set of all particles, the familiar hexagonal structures appear. However, this comes at the cost of a bad representation of the individual sets. In each of these sets, particles form striking clusters which are even well visible in the joint plot. The opposite is the case if we choose $\varphi$ very large. In such cases, the single sets are very well represented, but the total set is clustered. This is because particles of different colours are no longer interested in an interaction with each other such that there are no significant changes in the equilibrium of forces if two particles of different colour move closer to each other. This causes 'worm'-like artefacts in the plot.

To this end, we can draw two conclusions from this experiment. One is that $\varphi \approx 1$ is a good choice if our application requires a good distribution of both the individual sets of points and the total set. Nevertheless, we also see that the model possesses a parameter that lets us choose which of the two requirements is more important for the particular application. Assume a plane is to be textured with different patches of grass. One could include wild flowers, while the other does not. In such cases, we are most probably more interested in covering the whole plane rather than distributing the patches with flowers in a strict checkerboard alternation to those without. Hence, we probably want to chose $\varphi < 1$. If in contrast, we are interested to place two less correlated classes of objects, such as trees and cattle into a meadow, $\varphi > 1$ might be the better choice. This can cause cows to be placed underneath a tree, while still, we require that they are placed with a sufficient distance to the trunk.

As a next experiment, we extend the previous case by objects with different sizes. To this end, we sample the same density as before by particles in two colours and two sizes. The corresponding interaction matrix for this case is shown in Figure 7.8. Since we expect the choices for the multi-colour weight $\varphi$ and the size priority $C$ to carry over, we only consider the case for $\varphi = 1$ and $C = 2$. From the result, we expect

- the overall set,
- its partition of large particles,
- the individual sets, and
- their partition of large particles

to be 'well' distributed. This evaluation is shown in Figure 7.42. As expected, all renderings look very smooth. In particular, neither of the ex-

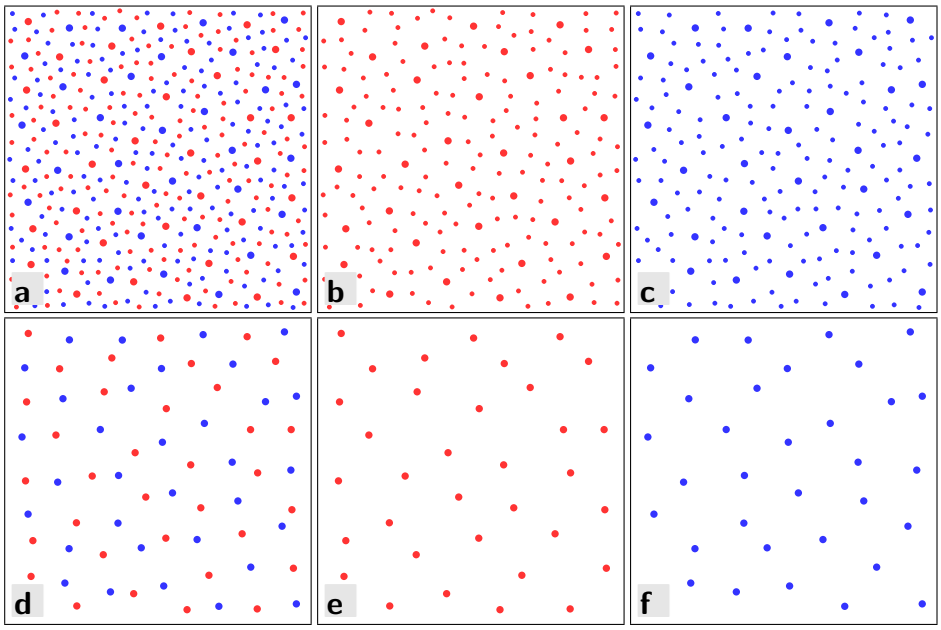



**Figure 7.42**: Second order multi class sampling of a uniform density. **a.** Complete result, and **d.** its partition of big dots. **b–c, e–f.** Red and blue subsets of a and d, respectively.

cerpts reveals striking artefacts that hint at the presence of a second class, or size of particles. Finally, let us remark that we can still have $\varphi$ and $C$ as free parameters if we are interested in results that are specially biased towards particular applications. The effects of either of the parameters on the result are then perceptually similar to those evaluated for the single parameters in terms of the previous experiments.

Figure 7.43 shows a first-order sampling of two overlapping Gaussians, sampled with 435 particles per colour. This experiment is very similar to the one performed by Wei [Wei10] but uses a larger image domain to better demonstrate the behaviour towards small grey values. Compared to the original work, electrostatic halftoning creates much more homogeneous results, as can particularly be seen in the centres of the Gaussians and in the transition region between both colours. Moreover, both sampling clouds possess an almost circular shape, regardless of the intersection with the differently coloured cloud. This is a clear benefit over the approach of Wei, which places several outliers in the region dominated by the other colour (see [Wei10, Figure 1]).

Now that we evaluated the principle properties of multi-class halftoning, let us see some examples in which rendering has been exchanged by some

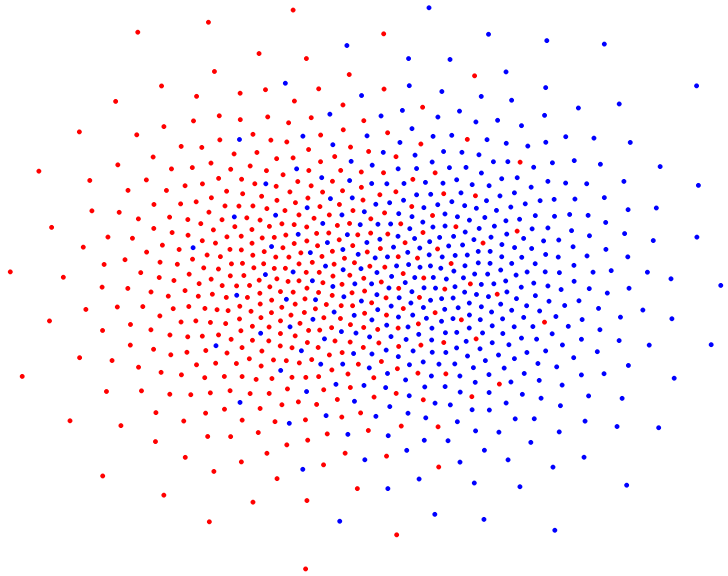


**Figure 7.43**: First-order sampling of two overlapping Gaussians.

more advanced operator. As a first application, we take a look on automatic texture generation. To this end, we distribute sprites on a 2-D image plane. The arising new image can then be used to texture the floor of large outdoor scenes or as an individual texture to objects such as clothing.

We begin with a simple setup that has a similar motivation as the last experiment but uses more complex sprites in two sizes. Figure 7.44 shows two intersecting density ramps from fully populated to empty. In the horizontal direction, this ramp is sampled with dandelions, in the vertical direction with daisies. The result shows that the properties we have seen for simple circles as elements carry over to more complex, yet almost circular real-world objects. In the top left corner, both flower types jointly fill the region such that both individual types are well visible, but the grass does not show through. On the main diagonal, we still have an equal distribution of both types, but the more we go to the bottom right corner, the more grass becomes visible. Finally, we observe a smooth transition from one class into the other while our view approaches the bottom left or top right corners. These characteristics indicate that electrostatic halftoning is profoundly applicable to sampling tasks with real-world objects.

As a next experiment, we want to explore the behaviour of this system for sprites with more complex, i.e. not necessarily circular, shapes. To this end, we generate images of pizzas with different covering patterns made up

**Figure 7.44**: Two intersecting density ramps with daisies (vertical) and dandelions (horizontal) in two sizes, each.

from salami, mushrooms, and ham in three different sizes, each. This is shown in Figure 7.45. The corresponding input density is always given in the bottom left corner of the results, with cyan indicating salami, magenta denoting mushrooms, and yellow representing ham. In the first part, we use a constant density for all three channels. Its rendered result looks very realistic and can hardly be distinguished from a real photograph. In particular, one does not immediately recognise that each type of ingredient is only made up from at most three different sprites which are rotated and scaled. Moreover, there is only a small amount of the pastry visible, which is mostly due to the imperfectly round shape of the used objects. This is a clear criterion of quality. The next example shows the rendering of a colour circle. It demonstrates how well each pair of colours mixes, and how well each individual colour can cover the base. While the circular slices of salami almost densely cover the designated area and also interleave well with other ingredients, this is different for the two other classes. As it turns out, mush-

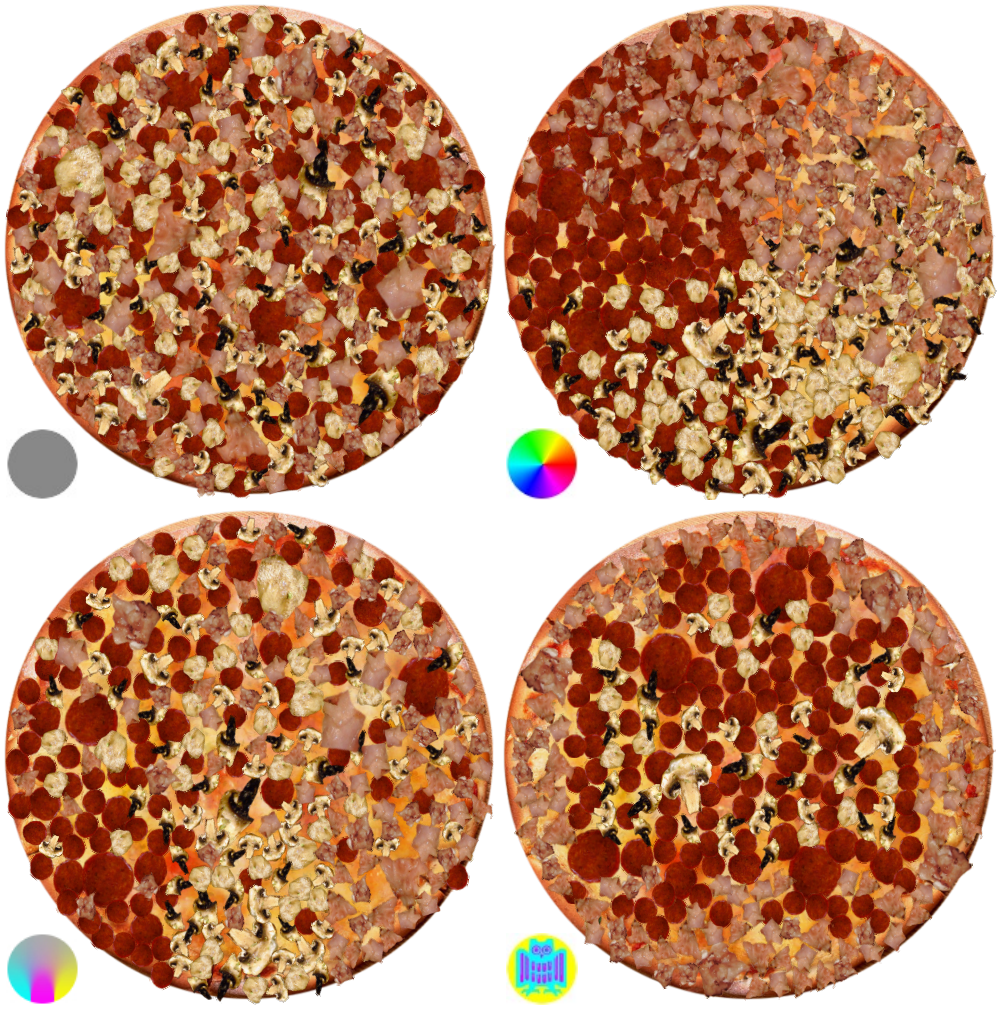


**Figure 7.45**: Multi-class sampling: Putting ingredients on Pizza. All pizzas are generated from sprites of salami (User:vagabondvince310, flickr.com, CC-BY), mushrooms (User:cyclonebill, flickr.com, CC-BY-SA), and ham (User:fugutabetai_shyashin, flickr.com, CC-BY-NC-SA) in three different sizes. The input densities in CMY coding are given in the bottom left corners.

rooms perform particularly bad in this respect which is a direct consequence of their complex shape. The sampling process is unaware of the orientation and coverage of each sprite and can thus not optimise and adapt these parameters. Apart from this principle issue, the overall result again looks photo-realistic, although the pattern is more artificial than in the previous rendering. Thirdly, we use an input density that fades from a state in which all classes of sprites are separated into a uniform mixture of all classes. To this end, we observe sharp transition edges between the different parts in the lower quarter, which dissolve more and more to form a distribution similar to the one in the first example. Because of the constraint that favours a good distribution of the largest instance of each class, we spot even some of these large objects in the upper half. To conclude this experiment, let us briefly focus on the fourth result from Figure 7.45. Depicting the logo of Saarland University, it demonstrates an extreme example for the multitude of possibilities of multi-class sampling. This particular application is simple from the perspective of sampling, since classes are not required to mix. Nevertheless, it is impressive to see that the depicted owl is clearly recognisable, although the sprites are relatively large. This gives rise to a series of new applications, such as the creation of photo-realistic print ads with the logo of the company embedded into an image of the product that is being advertised.

Finally, let us perform an outlook to the multitude of applications of electrostatic multi-class sampling for digital object placement. This requirement arises in computer graphics when it comes to the generation of a large heterogeneous cloud of 3-D meshes which still follow a certain distribution. Using this method, we can generate simple scenes such as realistic natural ecosystems or even highly complex distributions such as spectators of a huge rock concert with multiple stages. As a simple example from the perspective of sampling, we want to construct a forest consisting of beeches and pines in different sizes which are homogeneously distributed on the image plane. Figure 7.46 shows the results of this experiment. We start with a two-class two-size distribution of points as in Figure 7.42. Using a custom import script to the modelling software *Blender* [Hes07], a tree mesh is placed at each of these points. Each mesh is then scaled to its desired size and rotated around the Z axis to avoid reoccurring patterns from any perspective. Figure 7.46 shows screenshots of the untextured but shaded model during the layout phase. The first view shows the same bird view perspective as the point plot, while the other depicts the forest as seen from the bottom edge. The last picture in this series shows a possible result of how this forest can look like in an animation or video game. It has been obtained by the built-in raytracer from Blender using a slight additional

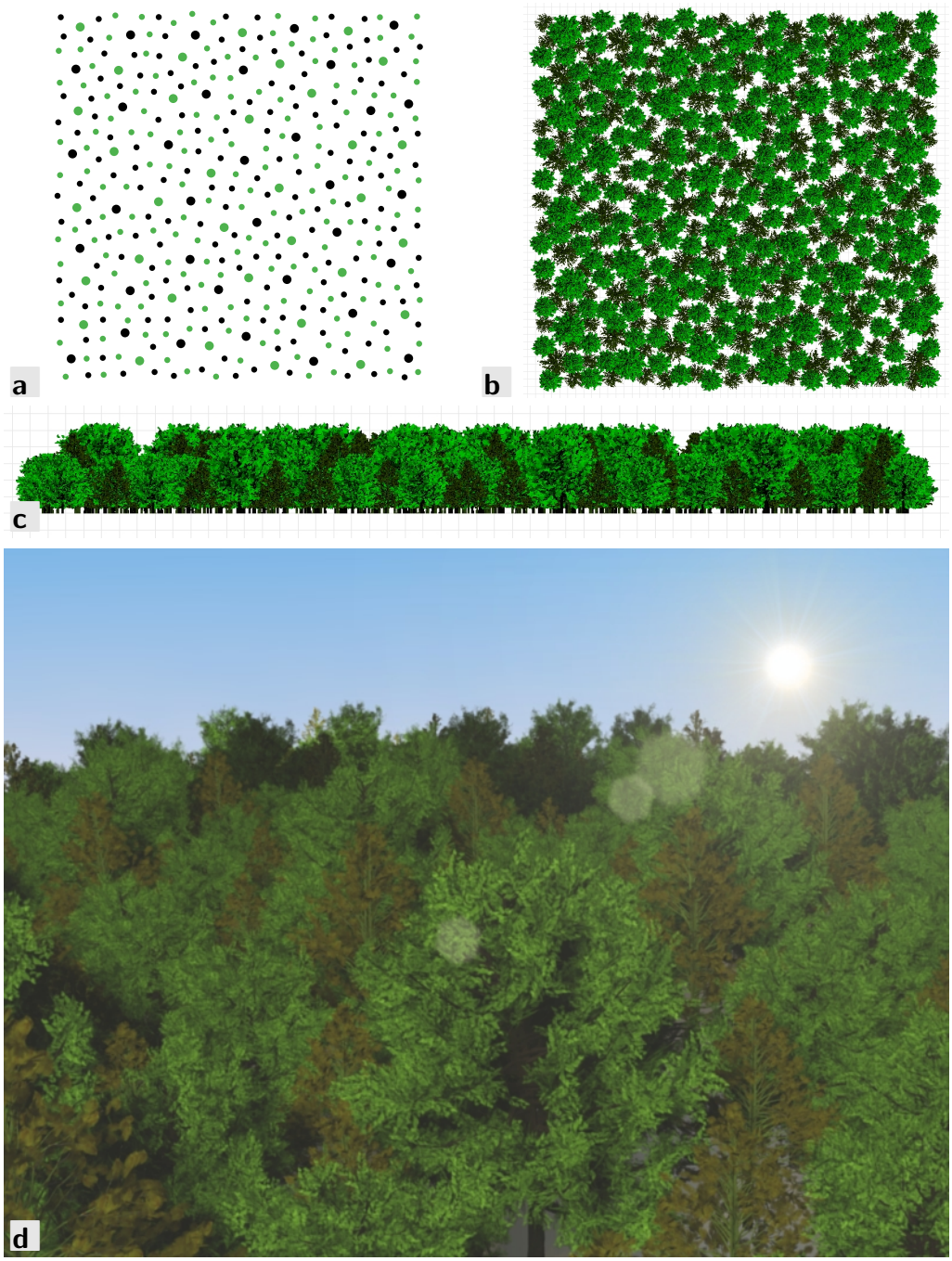



**Figure 7.46**: Object placement by electrostatic multi-class sampling. **a.** Locations. **b.** Sampled with 3-D meshes of pines and beeches (CC-BY, Yorik van Havre). **c.** Ditto, viewed from the side. **d.** Rendered with *Blender* using realistic textures and illumination.

post-processing such as tonemapping. Although only two different meshes are used, the forest looks very much photo-realistic. It fills the designated area homogeneously, but does not show any artefacts that could reveal its nature of being automatically generated.

### 7.7.4 Runtime

Let us now evaluate and compare the runtimes of electrostatic halftoning algorithms. Since hundreds to thousands of iterations are required to have the system converge to a very good result, the overall runtimes of all algorithms depend most crucially on the time required for one iteration. In turn, this particular runtime depends only on the configuration of particles, since attractive forces are already precomputed in the initialisation phase, and their values can be accessed directly. To this end, we take a varying number of particles, distribute them uniformly on a sufficiently large image, and compute the runtime of direct summation on CPU and GPU, and of NFFT-based fast summation on both CPU and GPU. As motivated by the experiments in Section 7.7.2, the latter were run with a quality parameter $\tilde{m} = 5$. A runtime comparison with different quality settings is given in the next section. In order to perform a fair comparison, all algorithms were hand-optimised and compiled using optimisation level 3 with auto-vectorisation (SSE). CPU algorithms are executed on one core of an Intel Core2 Duo E8200 clocked with 2.66GHz, while GPU algorithms are run on an NVidia GeForce GTX 480 on the same machine. Throughout the experiments, no swapping occurred such that all benchmarks were obtained on physical memory.

Table 7.2 shows the absolute runtime of the algorithms for one iteration, as well as their relative speedups. The 'Speedup' rows correspond to the parallelisation gain within a particular method, while the 'Speedup' column refers to the acceleration by choosing the fast summation technique over the direct summation method. The 'Total Speedup' denotes the overall gain from the CPU-based direct summation approach to the GPU-based fast summation technique.

Due to the high number of particle-particle interactions, the CPU-based direct summation-algorithm is relatively slow. On about 1 million particles, it takes almost 2.5 hours to perform 1 iteration — which is much too long for practical applications, considering that more than 100 iterations are required to obtain a good solution. A full run of the algorithm requires several weeks and can approach the order of several months if a very high quality is required.

If we parallelise this algorithm for execution on the GPU, this already

**Table 7.2**: Runtime comparison for 1 iteration with different numbers of particles and corresponding speedup factors. All times are given in seconds. Speedup factors describe the parallelisation benefit (vertical), the difference between runtime classes (horizontal), and the overall improvement of the fast summation GPU method over the original CPU direct summation technique (total).

| Particles | | Direct | Fast | Speedup | Total Speedup |
|---|---|---|---|---|---|
| | **CPU** | 2.12 | 0.84 | **2.54** | |
| **16 384** | **GPU** | 0.03 | 0.02 | **1.62** | |
| | **Speedup** | **70.67** | **41.12** | | **104.38** |
| | **CPU** | 33.90 | 3.65 | **9.28** | |
| **65 536** | **GPU** | 0.43 | 0.07 | **6.69** | |
| | **Speedup** | **78.84** | **56.00** | | **519.85** |
| | **CPU** | 542.64 | 14.74 | **36.81** | |
| **262 144** | **GPU** | 6.62 | 0.28 | **24.06** | |
| | **Speedup** | **81.97** | **53.58** | | **1 972.48** |
| | **CPU** | 8 853.46 | 57.95 | **152.78** | |
| **1 045 876** | **GPU** | 103.61 | 1.20 | **86.17** | |
| | **Speedup** | **85.45** | **48.20** | | **7 363.50** |

yields a speedup of about 80. The runtime for one iteration is reduced to less than 2 minutes, which allows for a convenient use of the algorithm for real-world images. High-quality results can be obtained in only a few hours. This result is particularly interesting if we recall that the structure of the algorithm remains very simple, even after the parallelisation. It allows to easily incorporate new concepts such as colour halftoning, second-order screening, or multi-class sampling without increasing the runtime by much.

For large numbers of particles, we can exploit the lower runtime complexity of fast summation algorithms. Compared to direct summation on the CPU, the CPU-based fast summation method yields speedups from about 2.5 for 16 thousand particles, to 150 for 1 million particles. By this, it even outperforms the GPU direct summation algorithm for more than about 600 000 particles.

As is expected, the best performance can be obtained by using a parallel GPU-based fast summation algorithm. It obtains a speedup of about 50 over the CPU-based variant. Unsurprisingly, this factor is lower than for direct summation. While for the direct summation approach, many operations are exclusively executed in fast shared memory, the fast summation

approach works largely on the slow global memory. Unlike direct summation, it even consists of parts with highly incoherent memory patterns such as they arise for the computation of the operators $\boldsymbol{B}$ and $\boldsymbol{B}^\top$ (cf. Section 7.6). In these cases, single values spread over the whole image domain are read or written. In these premises, a factor 50 is a substantive and also unexpected parallelisation gain.

Finally, let us consider the overall improvement in runtime between the CPU-based direct summation technique and the GPU-based fast summation technique. Given 1 million particles, one iteration requires only little more than one second, compared to more than 8 800 seconds for the simple algorithm. This corresponds to a speedup factor higher than 7 000. Even for moderate particle numbers, astonishing speedups can be obtained, such as almost 2 000 for a quarter million particles, or still 100 for as few as 16 000 particles.

The absolute runtime of the fast summation approach is also much lower than reported runtimes for many competing methods from the literature. The technique of Balzer *et al.* in its standard parametrisation with 1 024 points per site takes 17 minutes until convergence for 20 000 particles [BSD09]. In the same time, we can perform 34 000 iterations with electrostatic halftoning, which leads to much more accurate results. The approach of Secord computes a set of 40 000 dots in 20 minutes [Sec02]. Since this corresponds to 24 000 iterations with electrostatic halftoning, and because its results are worse than the ones obtained with the capacity-constrained approach of Balzer *et al.*, the method presented in this chapter performs much better. Finally, Li *et al.* proposed a fast parallel implementation of the capacity-constrained voronoi tesselation, which is about 10 times faster than the original algorithm [LNW+09]. Since the quality of the results is already surpassed by several hundreds of iterations of electrostatic halftoning, we obtain a much better quality with the 3 600 iterations of the fast summation technique that are possible in the same time.

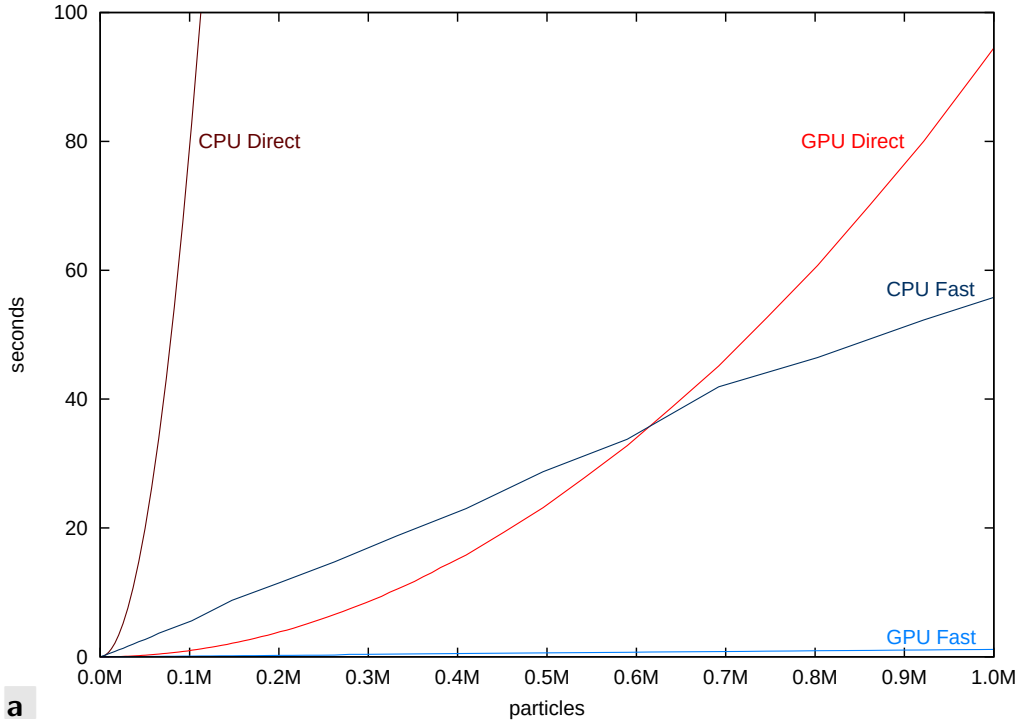
Recently, Fattal presented a method which yields reasonably good results in a much lower runtime than electrostatic halftoning [Fat11]. Even on a CPU this method is about 10 times faster than the fast summation method for electrostatic halftoning on the GPU. This comes at a higher approximation error, which is comparable to the one of the capacity-constrained method by Balzer *et al.* With these characteristics, Fattal's method might currently be more interesting for certain real-time applications than electrostatic halftoning, in particular when such applications do not require results in a particularly high quality. On a long term, however, this multi-scale approach can also turn into a great chance for electrostatic halftoning. We have seen earlier in this chapter that a good initialisation helps to accelerate

the runtime of our algorithms. Hence, we could apply a similar strategy as in [Fat11] to break down the problem hierarchically and to initialise with the solution from coarser scales. By this, it is likely that electrostatic halftoning manages to compute much more accurate results than the approach from [Fat11] in the same time.

If we now have a look on the scaling behaviour of the algorithms as shown in Figure 7.47a, we can clearly distinguish the complexity classes of $\mathcal{O}(M^2)$ in the number of particles $M$ for the direct summation approaches and $\mathcal{O}(M \log M)$ for fast summation. Because of a strong dominance of linear contributions such as the convolutions involved in the operators $B$ and $B^\top$, the latter graphs even appear linear. In the next section, we go more into detail about this issue. However, the previously mentioned computational overhead for the fast summation algorithms also causes them to perform worse for a small amount of particles. This is shown in a zoom-in to the bottom left part of the graph. Up to about 11 500 particles, the much simpler direct summation algorithms require clearly less time for the same number of interactions. The characteristic staircasing in the graph of the GPU direct summation algorithm is caused by its parallelisation layout. 4 096 particles are clustered in one block grid, such that each edge corresponds to additional kernel launches by the CPU.

A second interesting measure apart from the runtime per iteration is the initialisation time. All presented algorithms must precompute the field of attractive forces. Besides this, the fast summation approaches additionally precompute the kernel and transform it to the frequency space. Moreover, the CPU variant also precomputes samples for the functions $\varphi$ and $\psi$ which the GPU variant evaluates on the fly during the iterations. The time requirements for this initialisation are shown in Figure 7.48. Overall initialisation times are depicted with solid lines. Dashed lines (for fast summation) indicate the time required to compute the attractive forces. We should be aware that the latter procedure is fully exchangeable with the initialisation of a direct summation approach. It does not matter if we precompute the attractive field by direct or fast summation, the outcome is always the same. The surplus initialisation times for the fast summation methods, in contrast, are an integral component of this type of algorithm. They depend only on the number of pixels, and must always be performed if we want to start fast summation iterations later on.

In general, the runtime of the initialisation confirms our insights obtained in the runtime evaluation for one iteration. This is not surprising, as it does not matter for the runtime if two pixels or if two particles interact with each other. As before, it is thus advisable to compute attractive forces by FFTs unless the image contains less than about 11 500 pixels. As



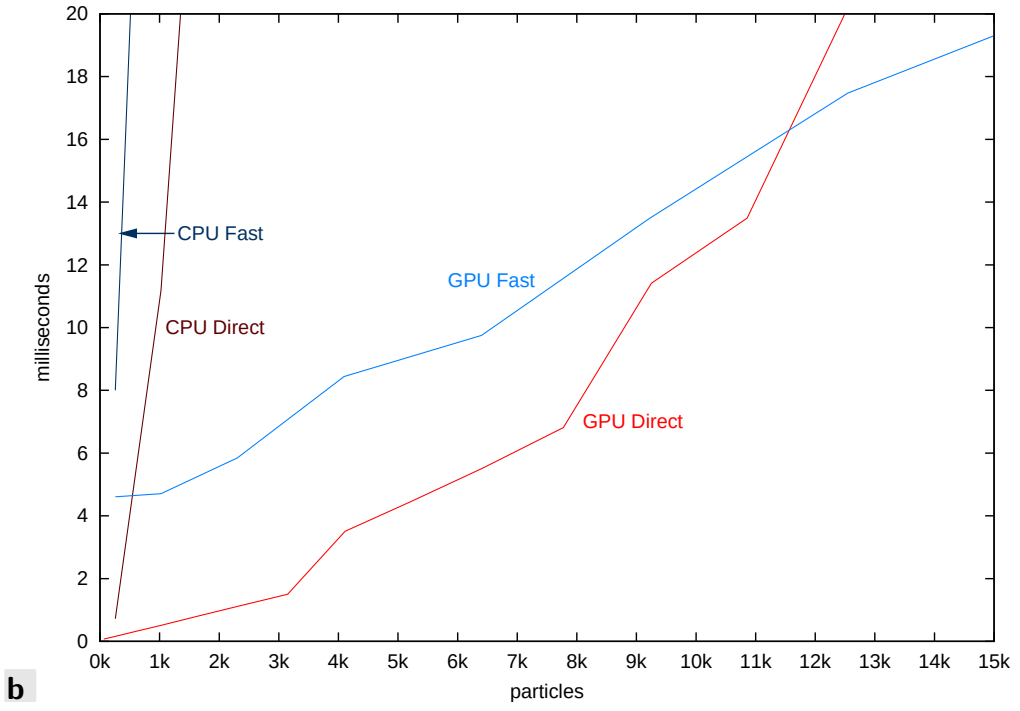




**Figure 7.47**: Runtime comparison for different halftoning algorithms. **a.** Full plot up to 1 million particles, and **b.** zoom into the bottom left corner of a.

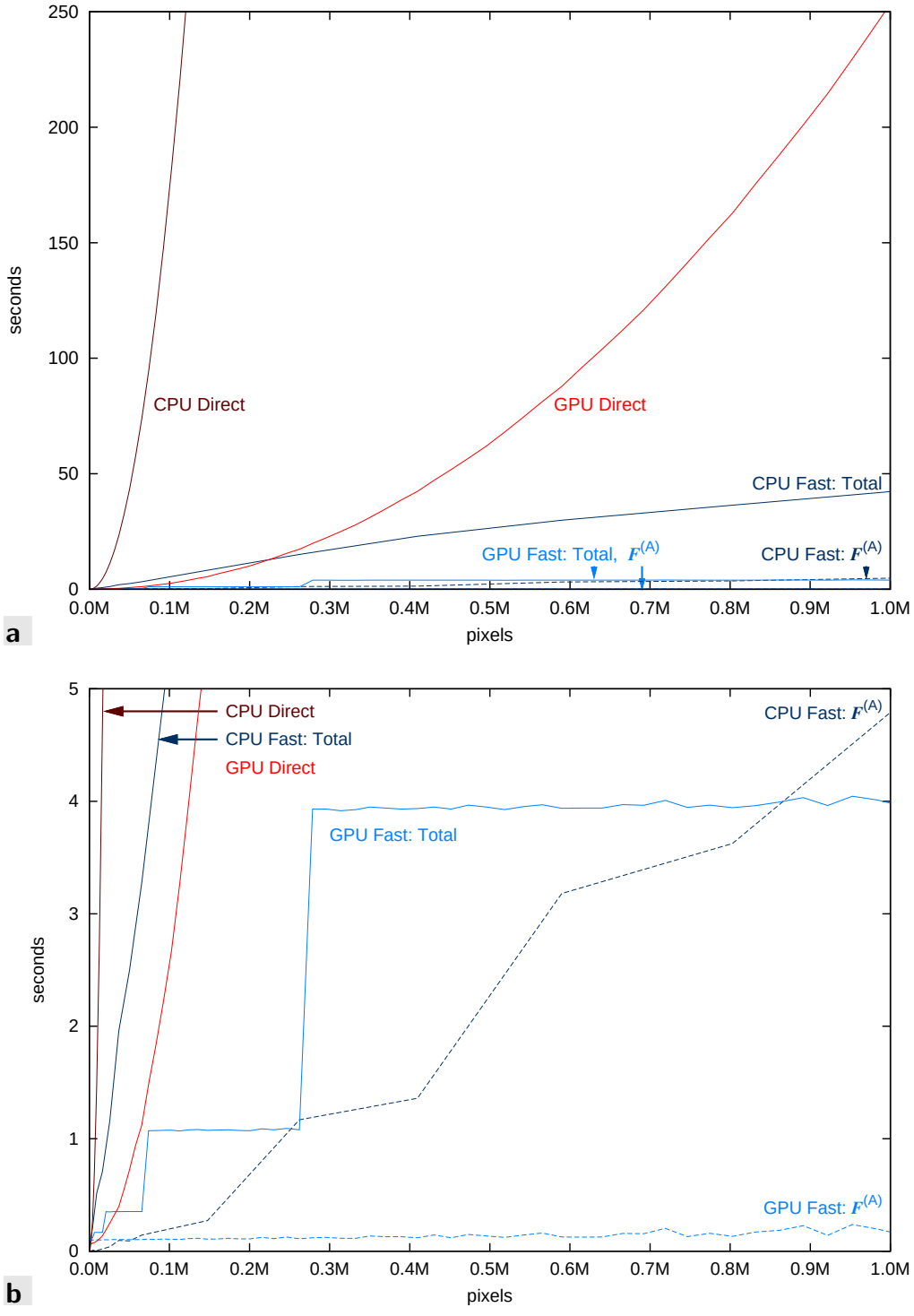



**Figure 7.48**: Comparison of the initialisation times for different halftoning algorithms. Solid lines denote total initialisation times, dashed lines (NFFT only) show the partition required to compute attractive forces. **a.** Complete plot, **b.** Zoom-in into lower part.

**Table 7.3**: Runtime of fast summation on the GPU in milliseconds depending on the quality parameter $\tilde{m}$, with 1 048 576 pixels and particles, each.

| $\tilde{m}$ | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| **1 Iteration** | 197.92 | 342.54 | 546.23 | 829.77 | 1 195.28 | 1 642.11 |
| **Initialisation** | 821.10 | 1 269.01 | 1 905.80 | 2 874.26 | 3 954.40 | 5 308.70 |

a new component, however, we observe significant time overheads for fast summation algorithms which is needed to prepare the following iterations. For the fast summation algorithm, this time even increases significantly as soon as a power of 4 in the number of pixels is reached. This is because the image plane in the frequency space grows likewise, but the radial kernel cannot be efficiently evaluated and sampled in parallel. A large array of either of these sizes is thus still filled on the CPU and then uploaded to the GPU, which in turn creates a runtime behaviour as can be seen here.

Finally, let us remark that physical memory limitations pose a fundamental problem to the fast summation algorithms. On 32-bit CPUs and modern graphics cards, neither of these algorithms can currently be applied to images larger than $1\,024^2$ pixels, or to images that contain more than about 1 million particles. This problem is partly solved already by the rise of 64-bit architectures such that we can expect machines and graphics cards to come with much more memory in the near future, However, there is still the need for an improved memory management of fast algorithms. This is particularly true because the direct summation approaches are capable of much larger sets of particles and pixels. Apart from a few intermediate variables, they only require three vectors, each containing elements consisting of two `float` positions. Two of these vectors contain the old and the new particle locations, while the third vector contains the attractive forces of the image. If we assume as many dots as pixels, a direct summation algorithm can handle up to 65 million particles. However, it would also require about 5 days to perform one iteration on our GPU. On the CPU, it even requires more than one year per iteration. This quick comparison again shows that the direct summation algorithm is infeasible for larger amounts of particles. Hence, the high memory requirements of fast summation methods can certainly be considered one of the most needed features to be addressed in future work.

### 7.7.5 Quality-Based Runtime for Fast Summation

The runtime of the NFFT-based fast summation algorithm depends on the choice of the quality parameter $\tilde{m}$. Hence, it is interesting to compare the runtimes for different quality settings to obtain a feeling for the tradeoff between accuracy and runtime. Table 7.3 depicts the times for one iteration and for the initialisation phase. It shows that the runtime for $\tilde{m} = 3$, which corresponds to a 'good' solution, is more than two times lower than for $\tilde{m} = 5$, the 'perfect' solution. Depending on the application, it can thus make sense to admit a small error in the result to obtain results much faster. This holds in particular if we recall that the PSNR for $\tilde{m} = 3$ is not significantly worse than for the direct summation approach (cf. Figure 7.19). Consequently, the much lower absolute runtime also affects the speedup factor compared to the naive approach. Given $1\,024^2$ particles, it grows from 7 363.50 for $\tilde{m} = 5$ to 16 208.30 for $\tilde{m} = 3$.

### 7.7.6 CUDA Performance Profiling

Finally, let us briefly measure the time consumption of individual sub-tasks during the execution of both presented GPU algorithms. This analysis reveals typical bottlenecks of the algorithms and helps to identify and approach these issues with future developments. To this end, we use the CUDA profiler with halftoning runs on both the GPU direct summation and fast summation methods. Although this data was obtained on 100 iterations and 10 shaking procedures, all graphs are normalised to 1 iteration and 1 shaking step. Contributions are coloured red if they scale with the number of iterations, blue if they are a one-off expense, yellow if they concern shaking, and green if they cover the one-off data transfer between GPU and CPU. As a sample problem, we use the *Trui* image from Figure 7.11, which contains 65 536 pixels and 30 150 particles.

Figure 7.49 shows the profiling of the direct summation run. Due to the quadratic scaling in the numbers of pixels and particles, the double as high number of pixels compared to particles, and the slightly higher time requirements per pixel, the initialisation phase dominates the time needed for one iteration. While these two numbers are one order of magnitude apart, the times for shaking and memory copy operations even vanish in this scale. They are between 3 and 5 orders of magnitude below the one for initialisation and become only visible if we rescale the time axis. One should note that shaking incorporates two kernels, the randomiser and a shift unit, which are coalesced in this plot. Hence, this plot again nicely shows both the simplicity and the computational complexity of this algorithm. In par-

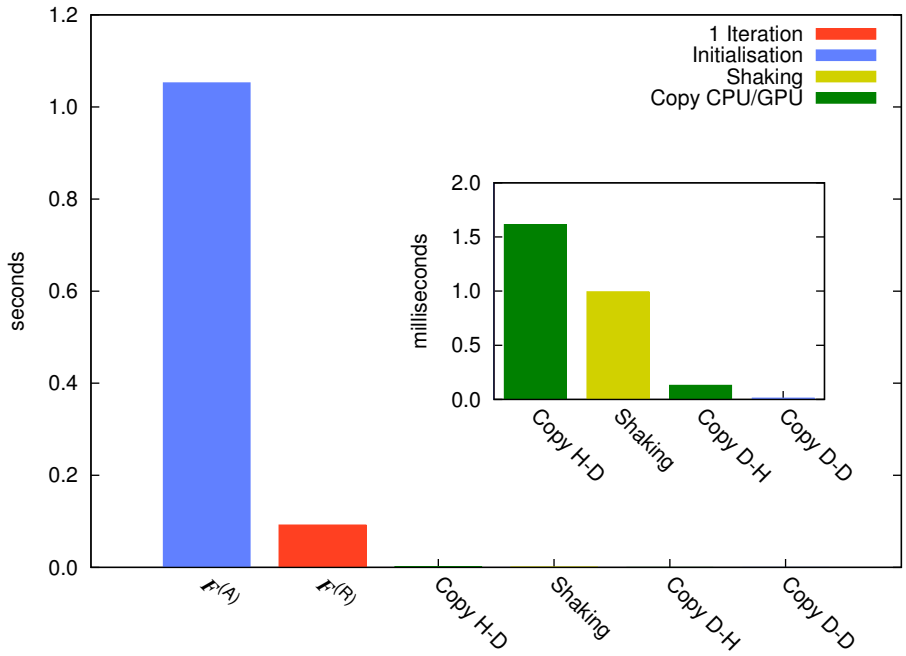



**Figure 7.49**: Module comparison for the GPU direct summation algorithm on *Trui*. Kernels are grouped to show runtime requirements of each operator. The inset shows zoom-ins to the bottom right corner.

ticular, we see that the overall runtime of the process is consumed by the two essential kernels for repulsion and attraction, which are already highly optimised and consist of only very few instructions. Since these kernels are highly memory-bound, there is no significant performance gain to be expected unless the memory bandwidth of graphics cards changes fundamentally.

If we perform the same experiment for the fast summation algorithm, this general impression changes. For this algorithm, we have a total of 34 different kernels that are involved in this process. In order to make Figure 7.50 more intuitively readable, however, these kernels are collapsed to 15 meaningful units. As an interesting side remark, we should note that this happens only for the right half of the diagram anyway such that the overall statement of this visualisation remains valid. As can be seen in the plot, the operators $\boldsymbol{B}$ and $\boldsymbol{B}^\top$ make up about 79% of the runtime for one iteration. This is due to the complex memory patterns involved in this method: Data loads from random positions can only be efficiently achieved if the data already resides in the texture cache. For random stores, this is even more severe as writes are potentially coinciding between threads. Thus,

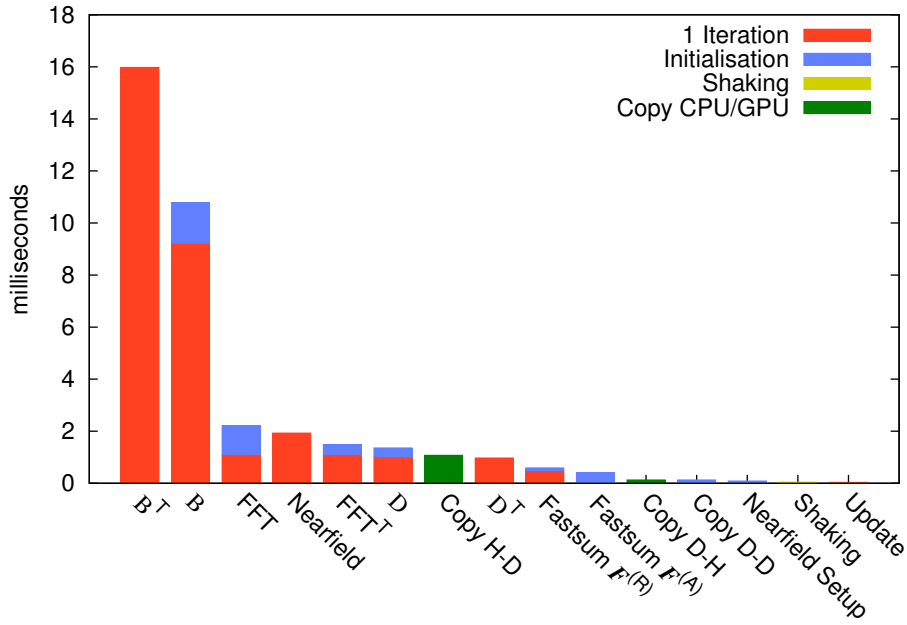



**Figure 7.50**: Module comparison for the GPU fast summation algorithm on *Trui*. Kernels are grouped to show runtime requirements of each operator.

writes can not efficiently be cached. To this end, this major bottleneck can only be resolved if either the support for random loads and stores is significantly enhanced by graphics cards manufacturers or if additional regularity assumptions on the data hold. While the first could for instance be improved by larger caches on the device, the latter was already addressed for special cases in the literature [SSNH08, Gre08, GD08].

Compared to the direct summation approach, the initialisation of the process requires significantly less time than one iteration. The main reason for this observation is given by the use of FFTs which are much faster than NFFTs. Moreover, the FFTs applied in this context work on a much smaller rectangular domain than the ones used by an NFFT. This is because of the oversampling $\tilde{n} = \alpha N$ with $2 \leqslant \alpha < 4$ that is necessary to keep the approximation error low (see Section 7.6.2). However, even in the initialisation phase requires one NFFT, as is indicated by the blue runtime partition for $\boldsymbol{B}$. It relates to the fact that the coefficients for the far-field kernel must be transformed into the frequency domain to be later used during convolution.

A surprising result can be observed for the measurements for the nearfield computation. Both the setup and the evaluation phase for the nearfield map possess a fairly low workload compared to the full process. This exceeds our expectations since setting up a nearfield map requires many,

potentially conflicting, random writes into the image, while the evaluation consists of an integration over a neighbourhood in a texture. Still, it seems that this good performance is related to the medium charge density for *Trui* and to the fact that each particle reads a 2-D patch which is in large parts cached. To this end, the heuristic approach with a data-parallel hash map can clearly be seen to pay off in the overall runtime.

## 7.8 Summary

In this chapter, we developed and explored a new method that uses electrostatic forces between particles to create halftones in a very high quality. In its basic form, it finds a continuous distribution of uniform points that approximates a given image. By simple modifications, this system can be adapted to output points on a discrete grid, to use points with different sizes or colours, and to optimise the output with respect to visual preferences such as an artificial irregularity, or an enhancement of edges. Moreover, the method can easily be applied for importance sampling purposes, where it serves as a highly accurate tool for object placement or texture generation.

Today, electrostatic halftoning constitutes the best halftoning technique in the literature, both in the discrete and in the continuous domain. This success is unimaginable without the highly efficient algorithms presented in this chapter. A simple direct-summation CPU implementation is not only too slow for real-world applications, it is also too slow to perform research on this topic. If one iteration on 1 million particles requires about 2.5 hours, 1 000 iterations take 100 days. With new efficient numerics and a fast parallel GPU implementation, however, the same results can be obtained within 20 minutes. This corresponds to a speedup of more than 7 000. By this, the algorithm even outperforms less accurate methods such as the capacity-constrained approach of Balzer *et al.* [BSD09] by several orders of magnitude in runtime. If we consider applications which do not necessarily require optimal solutions, but which work with very good approximations as well, the speedup over the naive approach even grows to more than 16 000. One important ingredient to obtain this speedup is given by the newly developed massively parallel NFFT algorithm. It is the first GPU-based method for this purpose which does not require special structural arrangements of nodes. Because the NFFT is nowadays a widely used tools for many areas of research and applications, the new algorithm has a broad applicability far beyond the field of halftoning.

Still, there seems to be space for improvements. Fast summation methods still require comparably much memory, which is a problem for really

large datasets. Online data compression schemes or data localisation methods could improve this bottleneck. Besides, it can make sense to neglect the mutual influence of particles if they are very far apart, in particular if we are considering domain sizes beyond those captured in this chapter. This simplification can open doors for the application of electrostatic halftoning on large scales such as on full DIN A4 pages in high resolution. Moreover, it might be interesting to extend the model by additional terms or constraints that extend its range of applicability. One interesting feature that gained importance in recent times is the support of structure-aware primitive orientations. Such extension allowed to use rotationally asymmetric rendering primitives, which enables electrostatic halftoning to be used for hatching, or for the placement of objects such as peppers on the pizza [Hae90, SHS02]. A second feature from which the model would benefit is a modification that allows the temporally stable processing of video data, potentially with prior knowledge of the underlying geometry of the scene [USS11]. In the moment, image sequences can only be processed on a per-frame basis, which causes primitives to be placed to new positions in every frame, and thus to make such videos appear noisy. Thirdly, it would be nice to extend the model to arbitrary manifolds [MGW05, GPS11]. Such an extension enables electrostatic halftoning to be used for texture generation for all kinds of complex shaped objects. Finally, the runtime of the proposed algorithms can potentially be reduced if they are embedded into a hierarchic framework as proposed in [Fat11]. This change could allow to carry over the quality of electrostatic halftoning to real-time applications. Given the flexibility and intuitive design of our algorithms, we can be confident that all these ideas are realisable with moderate effort.

To this end, the work presented in this chapter lays the practical foundations for a whole class of algorithms with applications in image processing, computer vision, and graphics. It shows that hardware-accelerated algorithms do not only offer a way to accelerate problems that are already sufficiently developed and evaluated, but they are also very well suited to accelerate the modelling and testing processes. The speedup provided by GPU-based algorithms reimburses for all of their shortcomings such as being more tedious to design and modify. It motivates to move GPU algorithms from their niche existence at the end of the modelling and design process right into its centre, and to use them as tools for our everyday work. Once they are designed, they can even be comfortably used and customised by users without a profound training in low-level programming. Since powerful graphics cards are relatively cheap compared to an equally efficient cluster of CPUs, and because they are today included in many modern desktop computers, GPU-based algorithms no longer constitute a pet project for a

selected community. Instead, they provide an efficient way to use all computing resources of a computer. Designing data-parallel algorithms for the GPU should thus be experienced as an ordinary style of scientific programming, just as it is with CPU-based algorithms today.

# Chapter 8

# Summary and Outlook

*A horse never runs so fast*
*as when he has other horses to catch up and outpace.*
Ovid

## 8.1 Overview

In this thesis, we have explored many different facets of GPU-accelerated algorithms for visual computing. Starting with the selection of suitable numerics, we have developed fast GPU-based algorithms for PDE-based image processing and computer vision, as well as for halftoning and sampling. This design process has been driven by the selection and development of efficient but intuitive concepts. As a result, our collection of algorithms is not specialised to limited applications, but creates a general framework for essential techniques in visual computing. This allows to carry over the good runtime of our algorithms even to new, potentially more complex models.

Let us first summarise the contributions of this thesis. In the next section, we are then going to draw conclusions from this work, and examine these contributions critically with respect to their influence on the area of visual computing.

We have started our work with the analysis of the simplest PDE in this field: *Homogeneous diffusion* [Iij59]. Although this method is very well understood and there are a multitude of different numerical approaches to its solution in the literature, there does not seem to exits a comprehensive comparison between these methods so far. This makes it hard to find an optimal GPU-based algorithm for this purpose. As a step towards the

solution of this deficiency, we have selected six important representatives, and have complemented them with two new approaches which are tailored for the application to parallel hardware. For all of these techniques, we have designed highly optimised GPU-based algorithms, which we have evaluated with respect to their quality and runtime performance.

While there are significant differences among the algorithms, we can clearly determine our parallel recursive filter based on [Hal06b] to yield the best performance on the GPU for moderate and large stopping times. Compared to the CPU, our GPU-based method obtains speedups of more than 130 which allows to process images up to a size of $4\,096 \times 4\,096$ pixels in realtime – regardless of the chosen stopping time. For very small stopping times, the discrete convolution with a Gaussian kernel turns out to be even more efficient than this technique.

Despite its simplicity, our novel *extended box filter* [GGBW11] is almost as fast as recursive filtering, although the quality of its results is slightly worse. In return, it is much easier to implement and optimise, such that it provides a convenient alternative whenever the implementation effort should be kept small.

In context of edge and coherence enhancing *anisotropic diffusion* [Wei98, Wei11a], we have discovered the fast explicit diffusion (FED) technique by Grewenig *et al.* [GWB10] as an efficient but simple solver for GPU-based algorithms [GZG+10]. Even with a very data-intensive discretisation scheme, our algorithm obtains speedups up to a factor 16. Assuming a stopping time $T = 500$, our algorithm still obtains real-time performance up to an image size of about $512 \times 512$ pixels. If we disregard approximative algorithms such as the parallel bilateral filter from [SKB+11], our technique seems to represent the fastest algorithm for anisotropic diffusion.

This performance of FED can be carried over to *PDE-based image inpainting*. Despite the raising importance of fast algorithms in this field, our algorithm seems to be the first anisotropic diffusion inpainting technique on the GPU. Its cascadic FED implementation runs more than ten times faster than the most efficient method on the CPU [SWB09]. While our algorithm still requires more than one second to solve a typical anisotropic image-based inpainting problem in the size $1\,024 \times 1\,024$ pixels, it is an important step towards applications such as real-time anisotropic image inpainting. Moreover, its flexibility allows our algorithm to be specialised to homogeneous diffusion inpainting, where it is among the fastest GPU-based approaches from the literature.

In the field of *optic flow*, we have designed one of the fastest yet very accurate algorithms in the literature. At the time of the first publication, it ranked on sixth place with respect to the approximation quality in the Mid-

dlebury benchmark, and obtained the lowest runtime among the top 10 of most accurate algorithms. Although it lost this position to newer methods in the meantime, it still represents the fastest method among the top 30 techniques. Compared to the original CPU-based method from [ZBW+09], our implementation [GZG+10] obtains a speedup of more than 40 without a significant decrease in quality.

One of the most significant impacts of the work conducted in this thesis are given for *electrostatic halftoning*. Our GPU-based algorithms [SGBW10, GSWT11] run 50–80 times faster than the corresponding algorithms on the CPU. In this context, we have developed the first unconstrained algorithm for the non-equidistant Fourier transform (NFFT) on GPUs, and have proposed the first GPU-based fast summation technique that sets up on this numerical scheme. Compared to a straightforward reference implementation of electrostatic halftoning on the CPU, our NFFT-based algorithm obtains a runtime improvement of more than 7 000. By that, it is not only faster than other, less accurate methods from the literature, but also lays the foundation for the extensive research that we have performed in this field.

Our dithering outperforms all other discrete halftoning methods from the literature when it comes to the approximation quality of the original image. In the continuous setting, electrostatic halftoning can be individually tuned towards individual sizes of rendering primitives. A fundamental problem to all point-based halftoning processes, the saturation of dark images regions, has been solved by a theoretically justified dot-overlap model. Electrostatic halftoning is also the first method in the literature which allows an freely adjustable trade-off between the mutually exclusive requirements of a good approximation quality on the one hand, and of a high visual quality on the other hand. For a further visual improvement, it is easily possible to enhance edges or other coherent structures in the halftone. This allows our approach to emulate the visual appearance of other methods from the literature, while still providing lower approximation errors than those techniques.

The high quality of electrostatic halftoning also carries over to settings with multiple colours, sizes of inkblots, or both. Due to technical reasons, such scenarios inevitably lead to new shortcomings such as a reduced resolution of the results. However, our flexible model allows to counterbalance the influences of all contributions. This guarantees a high quality, even under complicated conditions. As it turns out, our concepts carry over to the application in multi-class sampling. In this respect, our method is the first technique to generate an optimal distribution of objects of different kind and colour according to a transparent set of overlap-and-exclusion rules.

## 8.2 Conclusions

The runtimes and speedups of our algorithms are impressive, but they also represent a relatively volatile measure. For applications such as optic flow, we have already observed a series of tremendous improvements in the literature which also affect the runtime of the process. Moreover, the development of new hardware will soon allow for even faster algorithms, or larger problem sizes.

Instead, there are many general ideas, insights, and impulses which may remain valid on a longer term. One important contribution seems to be the introduction of FED and FJ to the field of general purpose computing on GPUs. Because of their favourable scaling behaviour and algorithmically simple structure, these solvers can efficiently be applied even to complicated problems. This is not only frequently reflected in our own research, but is also confirmed by other works from the literature [BAR11].

The high flexibility and transparency to the modeller is a general concept that we find throughout this thesis. It does not only hold for our FED- and FJ-based algorithms for PDE-based models, but also for our highly optimised linear diffusion algorithm which can efficiently be used for pre-smoothing purposes in this context, and for the GPU-based fast summation algorithm described in Chapter 7. All our efficient GPU-based algorithms are compatible to each other, and can be combined to more complex frameworks. Even for people that are inexperienced with GPU programming, it is easy to write a CPU-sided program which calls the respective GPU kernels instead of performing tasks directly on the CPU. This is also a result of our design decisions to adapt our algorithms to common and versatile concepts such as the energy minimisation via the Euler-Lagrange framework. Hence, our new abstraction layer offers significant performance improvements without the need to learn new programming concepts.

This is also interesting for a faster pace in research. There are many modern algorithms in visual computing which are much more accurate than their predecessors, but which also take considerably more time to find a solution. Where it is not possible to optimise parameters on a smaller test example, it takes a long time for a researcher to judge the quality of a new method, and to compare it against other techniques in the field. Because our GPU-based methods all provide a quality which is almost indistinguishable from their corresponding method on the CPU, it is convenient to use the faster GPU-based algorithms for rapid prototyping.

One particular example for the use of fast algorithms during the design of algorithms is our novel technique of electrostatic halftoning. Given the CPU-based hardware at the time, it would not have been possible to de-

sign the multitude of extensions which established electrostatic halftoning as one of the most accurate techniques for dithering, non-photorealistic rendering, and sampling. This includes in particular recent developments such as multi-class and multi-size sampling. The availability of algorithms which solve the arising problems in few minutes instead of several days helped to open this entirely new field of research. The results of electrostatic halftoning can again be used to improve many other fields of visual computing with respect to the quality of their results: Exciting examples for this multitude off applications are image compression, non-photorealistic rendering with arbitrary primitives, re-lighting, high dynamic range imaging, texture generation, or object placement.

Another important result with respect to further research is the scaling behaviour of our new algorithms. Although all of them obtained a speedup over their respective CPU counterparts, this performance gap is particularly high for some algorithms such as for linear diffusion or halftoning. This information helps to estimate the potential that the design of a GPU-aided algorithm bears, and allows to set it into relation with the workload accompanied with this process.

Besides the aforementioned general insights, this work has presented a high number of smaller contributions to the various fields of visual computing. One dominant example are our thorough surveys of modern algorithms for linear diffusion and halftoning. While in the latter case, there exist several overviews on traditional methods [Uli87, Kip01], our comparison seems to be the first that also considers a large range of recent techniques. Moreover, there does not even seem to be such comparison for homogeneous diffusion algorithms in the literature so far. This is very surprising if we consider the omnipresence of this technique and the multitude of different algorithms for this purpose. The survey performed in this thesis should be understood as the first step towards a comprehensive analysis, although the statement of such evaluation may finally look different on other architectures. Our CPU-based experiment suggests that our novel extended box filter may play a more important role there.

## 8.3 Future Work

Although this work gives many insights and answers various questions, there are several aspects which may substantially benefit from future research. In Chapters 4–6, we have designed several parabolic and elliptic solvers for PDE-based image processing and computer vision. Our FED- and FJ-based algorithms are highly optimised to the particular problem, while they still

preserve their flexibility. However, it would be presumptuous to assume that they represent the *optimal* solution to each of these problems. This becomes particularly obvious if we look back to the much simpler problem of homogeneous diffusion, were there is a large number of fundamentally different approaches. Given the higher parameter-dependency and memory-boundedness of nonlinear techniques, it is likely that a similar variety of methods also exists for these methods. However, a sound analysis and evaluation of the most promising alternatives to FED and FJ is time-consuming. On the one hand, it is possible to focus further investigations on fast numerical solvers such as (preconditioned) conjugate gradient schemes [Mei05] or primal-dual approaches for diffusion-reaction processes [ZPB07, WTP+09]. On the other hand, it is also worthwhile to evaluate the potential of fast approximative solutions such as it is obtained for a dynamical adaptation of the stencil size during the iteration [JCW09].

The Fast Jacobi solver [Wei11b] may play an important role in this process. Its characteristics proved to yield much better results than FED for the elliptic equations arising in optic flow computations. It is likely that this performance carries over to other elliptic processes, where FJ or cascadic FJ on the GPU could provide an efficient alternative to sequential CPU-based schemes such as cascadic successive over-relaxation. In this context, it should be evaluated how the coupling weight $c$ between the FED time steps and the FJ relaxation parameters can be chosen such that the process provides an optimal runtime while still guaranteeing the numerical stability of the process.

A similar discussion should also be started for homogeneous diffusion. While we have already evaluated numerous techniques in Chapter 3, it is not said that the optimal method with respect to approximation quality and runtime is among these selected representatives. In general, we have found that the key to a good runtime on GPUs is given by a reduction of the algorithmic structure to only a few sweeps. This suggests to experiment with more techniques which have similar characteristics. Besides recursive filters with less processing steps [YvV95], this includes techniques such as integral images [BSB10]. A new, potentially faster approach than our recursive filter that sets up on such ideas was recently proposed in [NMLH11].

In context of homogeneous diffusion, it could also be interesting to investigate the applicability of our novel extended box filter further. While it performs worse than recursive filters on the GPU, it seems to be equally well suited for application on the CPU. Hence, it is also interesting to see whether there are other hardware platforms on which similar ideas could lead to a significant performance gain. This affects in particular hybrid architectures such as the Cell processor, and vector processors.

Despite this large number of suggestions, it seems that the new electrostatic halftoning method still provides the highest potential for exciting improvements. One property that can not be expressed by our halftoning approach yet is a truly anisotropic behaviour. This prevents a use of our new technique for stroke-based non-photorealistic rendering, so-called hatching, and restricts its applicability for object placement or texture generations. Once it is possible to rotate primitives according to their shape, it will be possible to optimally place peppers on pizza, or flower motives with stalk onto cloth. Moreover, it might become possible to tap fields of artistic rendering, such as the visualisation of faces with objects such as fruits [Ghi11].

The work described by this thesis should also be complemented by a comprehensive evaluation of potential application fields for electrostatic halftoning. There are many potential applications, but most of them are today governed by competing methods. Even though our method requires more time to find a solution, the quality of its results is usually much higher. This advantage can be the key to significant improvements in PDE-based image compression [BBBW09], in high dynamic range imaging and realistic illumination [CD01, ARBJ03], as well as for automatic texture generation [LD05].

Future work on electrostatic halftoning should also be concerned with lower memory requirements on the one hand, and with efficient numerics or approximative solutions on the other hand. Our work has defined new levels for the accuracy of halftoning and blue noise sampling, but suffers still from a high computational workload and a massive memory consumption. Different numerical schemes such as the fast multipole method (FMM) [GD08] can slightly change its absolute runtime, but not the runtime class or the memory requirements. Instead, it may be interesting to see if the quick drop-off of the potential function can be exploited to perform a real decomposition of the image domain. This may lead to fully separable approximative solutions, and could open doors for the application of electrostatic halftoning to huge point sets as they occur for the printing of large pages and posters. An alternative solution can be fast approximative multigrid techniques such as the one from [Fat11], which is also much more conservative with respect to its memory footprint. It remains to see if such approaches can yield a similar quality as electrostatic halftoning.

# Appendix A

# Proofs

*A proof tells us where to concentrate our doubts.*
Morris Kline

## A.1 Linear Diffusion

**Proof 1 (for Theorem 3.1).**

By (3.73), it follows that

$$\sigma_G^2 = \sigma^2 - d\frac{L^2 - h^2}{12}\,. \tag{A.1}$$

We apply (3.72) for $L$ and obtain

$$\sigma_G^2 = \sigma^2 - dh^2\frac{\left(2\left\lfloor\frac{\sqrt{12\frac{\sigma^2}{dh^2}+1}-1}{2}\right\rfloor+1\right)^2-1}{12}\,, \tag{A.2}$$

In order to give a lower bound for the term $\lfloor\cdot\rfloor$, we distinguish two cases, based on whether

$$\frac{\sqrt{12\frac{\sigma^2}{dh^2}+1}-1}{2} \overset{?}{\geqslant} 1 \tag{A.3}$$

$$\Leftrightarrow \qquad \sigma^2 \overset{?}{\geqslant} \frac{2}{3}dh^2 \tag{A.4}$$

$$\Leftrightarrow \qquad L \overset{?}{>} h\,. \tag{A.5}$$

The latter equivalence follows from (3.72).

**Case $L > h$:**

$$\sigma_G^2 = \sigma^2 - dh^2 \frac{\left(2\left\lfloor \frac{\sqrt{12\frac{\sigma^2}{dh^2}+1}-1}{2} \right\rfloor + 1\right)^2 - 1}{12} \tag{A.6}$$

$$< \sigma^2 - dh^2 \frac{\left(2\left(\frac{\sqrt{12\frac{\sigma^2}{dh^2}+1}-1}{2} - 1\right) + 1\right)^2 - 1}{12} \tag{A.7}$$

$$= \sigma^2 - dh^2 \frac{\left(\sqrt{12\frac{\sigma^2}{dh^2}+1} - 2\right)^2 - 1}{12} \tag{A.8}$$

$$= \sigma^2 - dh^2 \frac{12\frac{\sigma^2}{dh^2} + 1 - 4\sqrt{12\frac{\sigma^2}{dh^2}+1} + 4 - 1}{12} \tag{A.9}$$

$$= \frac{dh^2}{3}\left(\sqrt{12\frac{\sigma^2}{dh^2}+1} - 1\right) \tag{A.10}$$

$$\tag{A.11}$$

Since both terms are positive, it follows that

$$\sigma_G < \sqrt{\frac{dh^2}{3}\left(\sqrt{12\frac{\sigma^2}{dh^2}+1} - 1\right)}. \tag{A.12}$$

**Case $L \leqslant h$:**

$$\sigma_G^2 = \sigma^2 - dh^2 \frac{(2\cdot 0 + 1)^2 - 1}{12} \tag{A.13}$$

$$= \sigma^2 \tag{A.14}$$

Hence, $\sigma_G = \sigma$. Finally, let us note that by (3.72), $L \geqslant h$. As a consequence, the latter case is equivalent to $L = h$. This finishes our proof.

■

**Proof 2 (for Theorem 3.2).**

By symmetry considerations, we see that the expectation value of $\boldsymbol{E}_\Lambda$ is

zero. For the variance $\sigma^2(\boldsymbol{E}_\Lambda)$ of one (non-iterated) box kernel, it follows

$$\sigma^2(\boldsymbol{E}_\Lambda) = \sum_{k=-(l+1)}^{l+1} (E_\Lambda)_{hk} \cdot (hk-0)^2 \tag{A.15}$$

$$= \sum_{k=-l}^{l} \frac{h}{\Lambda}(hk)^2 + hw\left(-(hl+h)\right)^2 + hw(hl+h)^2 \tag{A.16}$$

$$= \frac{2h^3}{\Lambda}\sum_{k=1}^{l} k^2 + 2h^3 w(l+1)^2 \tag{A.17}$$

$$\overset{(3.82)}{=} \frac{h^3}{3\Lambda}\left(2l^3+3l^2+l+6\alpha(l+1)^2\right) \ . \tag{A.18}$$

By Bienaymé's formula, we obtain the variance $\sigma^2(\boldsymbol{E}_\Lambda^d)$ for the iterated extended box kernel as the sum of single variances [Kre05]. This concludes the proof.

■

**Proof 3 (for Theorem 3.3).**

Let $L = h(2l+1)$ with $l \in \mathbb{N}_0$. We regard both limiting cases $\Lambda \to L^+$ and $\Lambda \to (L+2h)^-$ separately, and write $\alpha = \frac{\Lambda - L}{2h}$ (by (3.76)).

**Case $\Lambda \to (L+2h)^-$:**

$$\lim_{\Lambda\to(L+2h)^-} \sigma^2(\boldsymbol{E}_\Lambda^d) = \lim_{\Lambda\to(L+2h)^-} \frac{dh^3}{3\Lambda}\left(2l^3+3l^2+l+6\frac{\Lambda-L}{2h}(l+1)^2\right) \tag{A.19}$$

$$= \frac{dh^3}{h(3L+6)}(2l^3+3l^2+l+6(l+1)^2) \tag{A.20}$$

$$= \frac{dh^2}{6l+9}(2l^3+9l^2+13l+6) \tag{A.21}$$

$$= dh^2(\frac{1}{3}l^2+l+\frac{2}{3}) \tag{A.22}$$

$$= d\frac{h^2((2l+3)^2-1)}{12} \tag{A.23}$$

$$= d\frac{(L+2h)^2-h^2}{12} \tag{A.24}$$

$$= \sigma^2(\boldsymbol{B}_{L+2h}^d) \tag{A.25}$$

**Case $\Lambda \to L^+$:**
Let us show that for $\Lambda = L$, the iterated extended box filter $\boldsymbol{E}_\Lambda^d$ is equal to the iterated conventional box filter $\boldsymbol{B}_L^d$. To this end, we denote $L$ by $h(2l'+1)$ in order to distinguish $l'$ from the $l$ in $\Lambda = h(2l+1+2\alpha)$. Note that for $\Lambda = L$, it holds that

$$\begin{aligned} l &= \left\lfloor \frac{\Lambda}{2h} - \frac{1}{2} \right\rfloor && \text{(A.26)} \\ &= \left\lfloor \frac{h(2l'+1)}{2h} - \frac{1}{2} \right\rfloor && \text{(A.27)} \\ &= \lfloor l' \rfloor && \text{(A.28)} \\ &= l' \,, && \text{(A.29)} \end{aligned}$$

and

$$\begin{aligned} w &= \frac{1}{2}\left(\frac{1}{h} - \frac{2l+1}{h(2l+1)}\right) && \text{(A.30)} \\ &= \frac{1}{2} \cdot 0 && \text{(A.31)} \\ &= 0 \,. && \text{(A.32)} \end{aligned}$$

It follows that $\boldsymbol{E}_\Lambda^d = \boldsymbol{B}_L^d$ if $\Lambda = L$, which implies that $\lim_{\Lambda \to L^+} \sigma^2(\boldsymbol{E}_\Lambda^d) = \sigma^2(\boldsymbol{B}_L^d)$. To this end, $\boldsymbol{E}_\Lambda^d$ is a consistent generalisation of $\boldsymbol{B}_L^d$ with respect to $\Lambda$.

■

**Proof 4 (for Theorem 3.4).**

We can deduce an approximation of the continuous setting by computing the limit of $\sigma^2(\boldsymbol{E}_\Lambda^d)$ for the grid spacing $h \to 0$. Since we are interested in the order of consistency, we must consider the variance in (3.83) and rewrite it:

$$\begin{aligned} \sigma^2(\boldsymbol{E}_\Lambda^d) &= \frac{(2hl)^3}{12\Lambda} + dh\frac{(2hl)^2}{4\Lambda} + dh^2\frac{2hl}{6\Lambda} + 2dh^3\frac{\alpha}{\Lambda}(l^2+2l+1) && \text{(A.33)} \\ &= d\frac{(2hl)^3}{12\Lambda} + dh(1+2\alpha)\frac{(2hl)^2}{4\Lambda} + dh^2(1+12\alpha)\frac{2hl}{6\Lambda} + dh^3\frac{2\alpha}{\Lambda}. && \text{(A.34)} \end{aligned}$$

Now we replace $2hl$ by $\Lambda - (1+2\alpha)h$ and get for the first three terms:

$$d\,\frac{(\Lambda-(1+2\alpha)h)^3}{12\Lambda} = \frac{d\Lambda^2}{12} - \frac{dh}{4}(1+2\alpha)\Lambda \tag{A.35}$$

$$+\ \frac{dh^2}{4}(1+2\alpha)^2 + \mathcal{O}(h^3), \tag{A.36}$$

$$dh(1+2\alpha)\frac{(\Lambda-(1+2\alpha)h)^2}{4\Lambda} = \frac{dh}{4}(1+2\alpha)\Lambda - \frac{dh^2}{2}(1+2\alpha)^2 + \mathcal{O}(h^3), \tag{A.37}$$

$$dh^2(1+12\alpha)\frac{\Lambda-(1+2\alpha)h}{6\Lambda} = \frac{dh^2}{6}(1+12\alpha) + \mathcal{O}(h^3). \tag{A.38}$$

The fourth term can be written as

$$dh^3\frac{2\alpha}{\Lambda} = 2dh^3 w\,. \tag{A.39}$$

Because $w$ is bounded to $w \in [0,1)$, we obtain:

$$\sigma^2(\boldsymbol{E}_\Lambda^d) = \frac{d\Lambda^2}{12} - h^2\cdot\frac{d}{12}\left(12\alpha^2-12\alpha+1\right) + \mathcal{O}(h^3). \tag{A.40}$$

Thus, the consistency order is $\mathcal{O}(h^2)$ and we can state that

$$\lim_{h\to 0}\sigma^2(\boldsymbol{E}_\Lambda^d) = d\frac{\Lambda^2}{12} = d\int_{-\frac{\Lambda}{2}}^{\frac{\Lambda}{2}}\frac{1}{\Lambda}\cdot x^2\mathrm{d}x = d\int_{-\infty}^{\infty}B_\Lambda(x)x^2\mathrm{d}x = \sigma^2(B_\Lambda^d). \tag{A.41}$$

■

**Proof 5 (for Theorem 3.5).**

From the dependency of $l$ from $\Lambda$ as given in (3.76), as well as from the additional conditions

- $h < \Lambda < 3h$,
- $l$ being integer, and
- $0 \leqslant \alpha < 1$,

we deduce that $l = 0$. This consequence matches our intuition, since we aim at constructing an extended box kernel with only one central weight. Moreover, since we do not iterate the extended box filter, we have $d = 1$.

Plugging $l = 0$ and $d = 1$ into (3.83) yields:

$$\sigma^2(\boldsymbol{E}_\Lambda) = h^3 \frac{6\alpha}{3h(1+2\alpha)} \tag{A.42}$$

$$\Leftrightarrow \qquad \alpha = \frac{1}{2\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)} - 2}\,. \tag{A.43}$$

Let us now compute the outer weights $(\boldsymbol{E}_\Lambda)_h$ of the kernel. By (3.78) and (3.79), they are given by $(\boldsymbol{E}_\Lambda)_h = hw$ with $w = \frac{1}{2}\left(\frac{1}{h} - \frac{2l+1}{\Lambda}\right)$:

$$(\boldsymbol{E}_\Lambda)_h = \frac{h}{2}\left(\frac{1}{h} - \frac{2l+1}{\Lambda}\right) \tag{A.44}$$

$$= \frac{h}{2}\left(\frac{1}{h} - \frac{1}{1+2\alpha}\right) \tag{A.45}$$

$$= \frac{1}{2}\left(\frac{1+2\alpha-1}{1+2\alpha}\right) \tag{A.46}$$

$$= \frac{\alpha}{1+2\alpha}\,. \tag{A.47}$$

Using (A.43) in (A.47) leads to:

$$(\boldsymbol{E}_\Lambda)_h = \frac{\frac{1}{2\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)}-2}}{1+2\frac{1}{2\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)}-2}} \tag{A.48}$$

$$= \frac{1}{2\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)} - 2 + 2} \tag{A.49}$$

$$= \frac{\sigma^2(\boldsymbol{E}_\Lambda)}{2h^2}\,. \tag{A.50}$$

By (3.6), the operator for the 1-D explicit scheme has off-diagonal entries $\tau/h^2$. This value equals $(\boldsymbol{E}_\Lambda)_h$ if and only if

$$\frac{\tau}{h^2} = \frac{\sigma^2(\boldsymbol{E}_\Lambda)}{2h^2} \tag{A.51}$$

$$\Leftrightarrow \qquad \sigma^2(\boldsymbol{E}_\Lambda) = 2\tau\,. \tag{A.52}$$

Let us now check if the diagonal entries of the second-order finite difference approximation of $(\boldsymbol{I} + \tau\boldsymbol{A})$ equal $(\boldsymbol{E}_\Lambda)_0$ under these conditions. Again

by (3.78) and (A.43), we have

$$(\boldsymbol{E}_\Lambda)_0 = \frac{1}{1+2\alpha} \tag{A.53}$$

$$= \frac{1}{1+\frac{1}{1\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)}-1}} \tag{A.54}$$

$$= \frac{\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)}-1}{\frac{h^2}{\sigma^2(\boldsymbol{E}_\Lambda)}-1+1} \tag{A.55}$$

$$= 1 - \frac{\sigma^2(\boldsymbol{E}_\Lambda)}{h^2} \tag{A.56}$$

$$\overset{(3.6)}{=} 1 - 2\frac{\tau}{h^2}\,. \tag{A.57}$$

Hence, $(\boldsymbol{E}_\Lambda)_0$ equals the diagonal entry of a second-order finite difference approximation for $(\boldsymbol{I}+\tau\boldsymbol{A})$ from (3.6). This finishes our proof.

■

## A.2 Halftoning

**Proof 6 (for Theorem 7.1).**

Without loss of generality, let us assume unit area for particles, i.e. $\pi r^2 = 1$. The radius of each particle is thus given by

$$r = \sqrt{\frac{1}{\pi}}. \tag{A.58}$$

Let us now compute the real distances between particles. Kepler's conjecture tells us that a hexagonally-close packing (HCP) are the densest arrangement that can be obtained. Hence, this setting fulfils the perfectness assumption posed on $P$. Under the special assumption of a regular grid in 2-D, Kepler's conjecture has been proven by Lagrange [Lag73]. As we want to compute the maximal overlap, we fully saturate the image, i.e. put as many particles inside as possible. Assuming an domain $\Omega = \mathbb{R}^2$, this comes down to a regular tiling of the image with hexagons of unit size $A_{\text{Hex}} = 1$. This is visualised in Figure A.1a.

Since the area of a hexagon is given by [BSMM08]

$$A_{\text{Hex}} = \frac{3\sqrt{3}}{2} r_{\text{Hex}}^2\,, \tag{A.59}$$

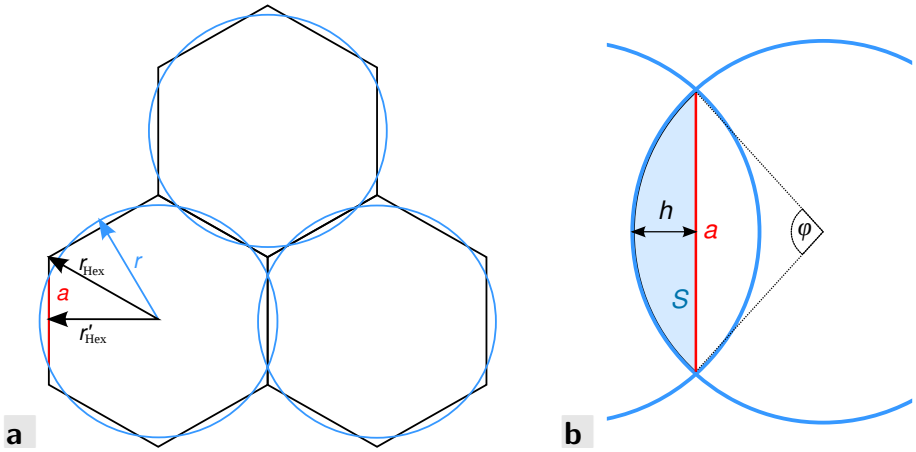



**Figure A.1**: Overlap of particles. **a.** Honeycomb pattern with a particle overlay as arising for dark images. **b.** Zoom on one overlap area.

the halved distance $r'_{\text{Hex}}$ to the neighbouring hexagon is given as the height of the equilateral triangle with edge length $r_{\text{Hex}}$:

$$r'_{\text{Hex}} = \sin 60° \cdot r_{\text{Hex}} \tag{A.60}$$

$$\overset{(A.59)}{=} \frac{\sqrt{3}}{2} \cdot \frac{\sqrt{2}}{\sqrt{3}\sqrt[4]{3}} \tag{A.61}$$

$$= \frac{1}{\sqrt{2}\sqrt[4]{3}} \tag{A.62}$$

We now place two particles in distance $2r'_{\text{Hex}}$ to each other, and draw circles with radii $r$ around them. We call the segment between the two intersection points of the circles $a$ and recognise it as the chord of one of the circles. Since $(a/2)^2 + r^2 = r^2_{\text{Hex}}$, the height $h$ of the circular segment that is cut off by $a$ is given by

$$h = r - \sqrt{r^2 - \frac{a^2}{4}}\,. \tag{A.63}$$

By (A.58) and by knowing that $h = r - r'_{\text{Hex}}$, we can compute $a$ as

$$\sqrt{\frac{1}{\pi} - \frac{a^2}{4}} = \frac{1}{\sqrt{2}\sqrt[4]{3}} \tag{A.64}$$

$$\Leftrightarrow \quad \frac{1}{\pi} - \frac{a^2}{4} = \frac{1}{2\sqrt{3}} \tag{A.65}$$

$$\Rightarrow \quad a = \sqrt{\frac{4}{\pi} - \frac{2}{\sqrt{3}}}\,. \tag{A.66}$$

Let us now use $a$ to compute the angle $\varphi$ of the triangle spanned by the two intersection points and the centre of the circle in the central point. Since

$$\frac{\sin\frac{\varphi}{2}}{\sin 90^\circ} = \frac{\frac{a}{2}}{r}\,, \tag{A.67}$$

we obtain

$$\varphi = 2\arcsin\frac{a}{2\sqrt{\frac{1}{\pi}}} \tag{A.68}$$

$$= 2\arcsin\sqrt{1-\frac{2\pi}{4\sqrt{3}}} \tag{A.69}$$

$$\approx 35.53^\circ \tag{A.70}$$

Finally, we compute the area $S$ of the circular segment (i.e. half-overlap) by [BSMM08]:

$$S = \frac{1}{2\pi}\left(\frac{\pi\varphi}{180^\circ} - \sin\varphi\right), \tag{A.71}$$

and since each particle overlaps with six other particles, we obtain

$$6S \approx 0.0372 = 0.0372(r^2\pi). \tag{A.72}$$

Since $A = A_{\text{Hex}}$, this is the same area which remains uncovered.

It remains to show that above a certain grey value, the rendering is mass-preserving. Given an optimal distribution of particles, the closest arrangement we can find without overlaps is again the HCP. If we decrease the grey value, i.e. move particles closer together, they will overlap and spoil the grey value, but if we move them further away, they will always cover the right area. Thus, we try to compute the grey value corresponding to this setting.

In the HCP, we thus take the triangle between the centres of three neighbouring circles and denote its area by $A_\Delta$. Within this triangle, there are three sectors of $\frac{1}{6}$ the area of a circle, each. We thus compute the coverage in this case, and denote the triangle's height by $h'$:

$$\frac{\frac{3}{6}\pi r^2}{A_\Delta} = \frac{\pi r^2}{2h'r} \qquad = \frac{\pi r^2}{2\sqrt{3}r^2} \tag{A.73}$$

$$= \frac{\pi}{2\sqrt{3}} \qquad \approx 0.9069 \tag{A.74}$$

Thus, above a grey value of $1 - 0.9069 = 0.0931$, the rendering of an ideally distributed point set is mass-preserving.

■

**Proof 7 (for Theorem 7.2).**

The existence of the integral from (7.19) is trivial on $\Omega \setminus \mathcal{N}_\epsilon(\boldsymbol{p}_m)$, where $\mathcal{N}_\epsilon(\boldsymbol{p}_m)$ denotes a square of halved side length $\varepsilon$ around $\boldsymbol{p}_m$, and $0 < \varepsilon \ll 1$. This follows immediately from the boundedness of the integrand and the finite extent of $\Omega$. Using that $u(\boldsymbol{x}) \in [0,1]$ holds for all $\boldsymbol{x}$, let us show that the integral also exists on $\mathcal{N}_\epsilon(\boldsymbol{p}_m)$. This is not self-evident, since the integral could diverge around $\boldsymbol{p}_m$ due to a vanishing denominator of the integrand. Without loss of generality, we substitute $u$ in (7.19) as $u'(\boldsymbol{x}) = (1 - u(\boldsymbol{x}))$ and obtain

$$\int_\Omega \frac{u'(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{p}_m)}{\|\boldsymbol{x} - \boldsymbol{p}_m\|^2} \mathrm{d}\boldsymbol{x} \ . \tag{A.75}$$

The result of this integral yields a force vector acting at a particle residing in $\boldsymbol{p}_m$. It suffices to show the absolute convergence of the integral over the (Euclidean) norm of the integrand, since this property implies the integrand to be absolutely integrable. As a consequence, the integral from (A.75) exists as well and is absolutely convergent.

$$\begin{aligned}
& \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \int_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} \left\| \frac{u'(a,b)\binom{a-\boldsymbol{p}_{m,x}}{b-\boldsymbol{p}_{m,y}}}{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2} \right\| \mathrm{d}b \, \mathrm{d}a && \text{(A.76)} \\
\leqslant & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \int_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} |u'(a,b)| \left\| \frac{\binom{a-\boldsymbol{p}_{m,x}}{b-\boldsymbol{p}_{m,y}}}{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2} \right\| \mathrm{d}b \, \mathrm{d}a && \text{(A.77)} \\
\overset{u' \leqslant 1}{\leqslant} & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \int_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} \left\| \frac{\binom{a-\boldsymbol{p}_{m,x}}{b-\boldsymbol{p}_{m,y}}}{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2} \right\| \mathrm{d}b \, \mathrm{d}a && \text{(A.78)} \\
= & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \int_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} \sqrt{\frac{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2}{\left((a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2\right)^2}} \mathrm{d}b \, \mathrm{d}a && \text{(A.79)} \\
= & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \int_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} \frac{1}{\sqrt{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2}} \mathrm{d}b \, \mathrm{d}a && \text{(A.80)} \\
\overset{(*)}{=} & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \left[ \ln\left( 2\left( \sqrt{(a-\boldsymbol{p}_{m,x})^2 + (b-\boldsymbol{p}_{m,y})^2} + b - \boldsymbol{p}_{m,y} \right) \right) \right]_{\boldsymbol{p}_{m,y}-\varepsilon}^{\boldsymbol{p}_{m,y}+\varepsilon} \mathrm{d}a && \text{(A.81)} \\
= & \int_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \left( \ln\left( \sqrt{(a-\boldsymbol{p}_{m,x})^2 + \varepsilon^2} + \varepsilon \right) \right. \\
& \qquad \left. - \ln\left( \sqrt{(a-\boldsymbol{p}_{m,x})^2 + \varepsilon^2} - \varepsilon \right) \right) \mathrm{d}a && \text{(A.82)}
\end{aligned}$$

$$
\begin{aligned}
{}^{(*)} \quad & \Bigg[(a-\boldsymbol{p}_{m,x})\ln\left(\sqrt{(a-\boldsymbol{p}_{m,x})^2+\varepsilon^2}+\varepsilon\right)+\varepsilon\ln(2) \\
& \quad +\varepsilon\ln\left(\sqrt{(a-\boldsymbol{p}_{m,x})^2+\varepsilon^2}+a-\boldsymbol{p}_{m,x}\right)-a\Bigg]_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \\
- \quad & \Bigg[(a-\boldsymbol{p}_{m,x})\ln\left(\sqrt{(a-\boldsymbol{p}_{m,x})^2+\varepsilon^2}-\varepsilon\right)-\varepsilon\ln(2) \\
& \quad -\varepsilon\ln\left(\sqrt{(a-\boldsymbol{p}_{m,x})^2+\varepsilon^2}+a-\boldsymbol{p}_{m,x}\right)-a\Bigg]_{\boldsymbol{p}_{m,x}-\varepsilon}^{\boldsymbol{p}_{m,x}+\varepsilon} \qquad \text{(A.83)} \\
= \quad & \varepsilon\ln(\sqrt{2}\varepsilon+\varepsilon)+\varepsilon\ln(\sqrt{2}\varepsilon+\varepsilon)-\varepsilon \\
& +\varepsilon\ln(\sqrt{2}\varepsilon+\varepsilon)-\varepsilon\ln(\sqrt{2}\varepsilon-\varepsilon)-\varepsilon \\
& -\varepsilon\ln(\sqrt{2}\varepsilon-\varepsilon)+\varepsilon\ln(\sqrt{2}\varepsilon+\varepsilon)+\varepsilon \\
& -\varepsilon\ln(\sqrt{2}\varepsilon-\varepsilon)-\varepsilon\ln(\sqrt{2}\varepsilon-\varepsilon)+\varepsilon \qquad \text{(A.84)} \\
= \quad & 4\varepsilon\ln\left(\frac{\sqrt{2}\varepsilon+\varepsilon}{\sqrt{2}\varepsilon-\varepsilon}\right) \qquad \text{(A.85)} \\
= \quad & 4\varepsilon\ln\left(\frac{\sqrt{2}+1}{\sqrt{2}-1}\right) \qquad \text{(A.86)} \\
< \quad & 7.06\ \varepsilon\ . \qquad \text{(A.87)}
\end{aligned}
$$

The equalities marked with $(*)$ were obtained by a computer algebra system (WolframAlpha). We conclude this proof with the observation that the integral from (A.76) exists. By 8.2.3.3 (4) in [BSMM08], integrals of type (A.75) exist as well. As a consequence, the attractive force acting on a particle never approaches infinity.

■

**Proof 8 (for Theorem 7.3).**

We consider a circular area of radius $r$ with uniform charge density 1, and a particle $m$ with unit charge at $\boldsymbol{p}_m = \boldsymbol{p}_c - \binom{0}{n}$ in distance $n > r$ from the centre $\boldsymbol{p}_c$ of the circle. This is visualised in Figure A.2. Without loss of generality, we can rotate the system such that the x coordinates of $\boldsymbol{p}_m$ and $\boldsymbol{p}_c$ are equal.

We want to compute the attractive force $\boldsymbol{F}^{(A)}$ acting on this particle. By symmetry of the left and right semi-circles, we see that the $x$ component of $\boldsymbol{F}^{(A)}$ cancels out. We thus need to compute its $y$ component only, and integrate the contributions over the area of the circular disc. This area is given by all points $\boldsymbol{p} \in \{\boldsymbol{p}_c + \binom{x}{y} \mid x^2 + y^2 \leqslant r^2\}$. We thus regard two

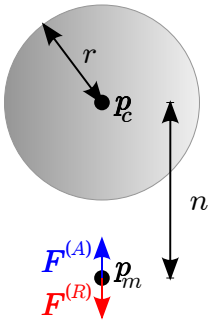



**Figure A.2**: Sketch for Proof 8. A circular area with radius $r$ and a point charge that takes just the negative charge of the area are electrically neutral to external charges.

cascaded integrals, one of which runs in $y$ direction from $-r$ to $+r$, and one which integrates the contributions in $x$ direction on the cut through the circle at the respective $y$ coordinate. The distance vector from $\boldsymbol{p}_m$ to the respective points is given by the difference $(\boldsymbol{p}_c + \binom{x}{y}) - \boldsymbol{p}_m = \binom{x}{n+y}$:

$$\boldsymbol{F}_y^{(A)} = \int_{-r}^{r} \int_{-\sqrt{r^2-y^2}}^{\sqrt{r^2-y^2}} \frac{1}{\left\| \binom{x}{n+y} \right\|} \cdot \frac{n+y}{\left\| \binom{x}{n+y} \right\|} \mathrm{d}x\mathrm{d}y \tag{A.88}$$

$$= \int_{-r}^{r} \int_{-\sqrt{r^2-y^2}}^{\sqrt{r^2-y^2}} \frac{1}{\sqrt{x^2+(n+y)^2}} \cdot \frac{n+y}{\sqrt{x^2+(n+y)^2}} \mathrm{d}x\mathrm{d}y \tag{A.89}$$

$$\overset{(*)}{=} \int_{-1}^{1} 2 \int_{0}^{r\sqrt{1-y^2}} \frac{n+ry}{x^2+(n+ry)^2} \cdot r \; \mathrm{d}x\mathrm{d}y \tag{A.90}$$

$$= 2r \int_{-1}^{1} \int_{0}^{r\sqrt{1-y^2}} \frac{n+ry}{x^2+(n+ry)^2} \mathrm{d}x\mathrm{d}y \tag{A.91}$$

In the step marked by $(*)$, we substitute the interval boundaries for the outer interval by $\varphi(s) = rs$ ('backwards'). The second integral is simplified by using symmetry of the circle. After pulling $r$ out of this integral, it is of form $a \int \frac{\mathrm{d}x}{x^2+a^2}$, and by 21.5.1.3 (57) in [BSMM08], its primitive is given by $\arctan \frac{x}{a}$:

$$\boldsymbol{F}_y^{(A)} = 2r \int_{-1}^{1} \left[ \arctan\left( \frac{x}{n+ry} \right) \right]_0^{r\sqrt{1-y^2}} \mathrm{d}y \tag{A.92}$$

Since $\arctan(0) = 0$, we obtain

$$\boldsymbol{F}_y^{(A)} = 2r \int_{-1}^{1} \arctan\left(\frac{r\sqrt{1-y^2}}{n+ry}\right) \mathrm{d}y \tag{A.93}$$

Using the assumption that $n > r > 0$, and passing the integral above into a computer algebra system (Waterloo Maple 13) yields

$$\boldsymbol{F}_y^{(A)} = 2r \cdot \frac{\pi r}{2n} \tag{A.94}$$

$$= \frac{\pi r^2}{n} \ . \tag{A.95}$$

As a second step, let us assume there is a particle at $\boldsymbol{p}_c$ which has just the (negative) charge of the circular region. Since the area of the circle is given by $A = \pi r^2$ and the area is uniformly charged with density 1, the particle has a charge of $-\pi r^2$. The repulsive force on the test charge in $\boldsymbol{p}_m$ is thus given by

$$\boldsymbol{F}^{(R)} = \frac{-\pi r^2}{\|\boldsymbol{p}_m - \boldsymbol{p}_c\|} \cdot \frac{\boldsymbol{p}_m - \boldsymbol{p}_c}{\|\boldsymbol{p}_m - \boldsymbol{p}_c\|} \ . \tag{A.96}$$

Again, since $\boldsymbol{p}_{m,x} - \boldsymbol{p}_{c,x} = 0$, let us only consider its $y$ component:

$$\boldsymbol{F}_y^{(R)} = \frac{-\pi r^2}{\sqrt{0^2+n^2}} \cdot \frac{n}{\sqrt{0^2+n^2}} \tag{A.97}$$

$$= \frac{-\pi n r^2}{n^2} \tag{A.98}$$

$$= \frac{-\pi r^2}{n} \ . \tag{A.99}$$

We finish this part with the observation that $\boldsymbol{F}_y^{(A)} + \boldsymbol{F}_y^{(R)} = 0$, and thus $\boldsymbol{F}^{(A)} + \boldsymbol{F}^{(R)} = \boldsymbol{0}$. This shows that a particle in the centre of a homogeneous circular region neutralises it with respect to external particles, if their charge equals the negative charge of the area. This finishes the proof.

■

**Proof 9 (for Theorem 7.4).**

In order to compensate for uncovered partitions of the image, let us analyse how large these regions are. Since Theorem 7.4 requires a perfect sampling operator, we can assume particles to be aligned in a honeycomb pattern. In analogy to Proof 6, we note that overlaps occur only if the circular disc that

represents a particle is larger than the incircle. If it equals the circumcircle, the area is rendered in a fully black tone.

To this end, let us use the notation from Figure A.1a. Assuming the interesting case $r > r'_{\text{Hex}}$, we can express the circular segment $S$ cut off by $a$ as the difference between the circular sector spanned by $\varphi$ and the triangle, i.e.

$$S = \frac{\varphi}{2\pi} \cdot \pi r^2 - r'_{\text{Hex}} \cdot \frac{a}{2} \,. \tag{A.100}$$

Since

$$\left(\frac{a}{2}\right)^2 + r'^2_{\text{Hex}} = r^2 \,, \tag{A.101}$$

we can write

$$\frac{a}{2} = \sqrt{r^2 - r'^2_{\text{Hex}}} \tag{A.102}$$

and use (A.102) to rewrite the second term of (A.100) to obtain

$$S = \frac{\varphi}{2\pi} \cdot \pi r^2 - r'_{\text{Hex}} \sqrt{r^2 - r'^2_{\text{Hex}}} \,. \tag{A.103}$$

Moreover, we also know that

$$\frac{a}{2} = r'_{\text{Hex}} \cdot \tan\left(\frac{\varphi}{2}\right) \,. \tag{A.104}$$

Resolving (A.104) for $r'_{\text{Hex}}$ and inserting it into (A.101) yields

$$r'^2_{\text{Hex}} \left(1 + \tan^2\left(\frac{\varphi}{2}\right)\right) = r^2 \tag{A.105}$$

$$\Rightarrow \qquad \varphi = 2 \arctan\left(\sqrt{\frac{r^2}{r'^2_{\text{Hex}}} - 1}\right) \,. \tag{A.106}$$

Hence, we can also rewrite the first term of (A.103) and obtain

$$S = \arctan\left(\sqrt{\frac{r^2}{r'^2_{\text{Hex}}} - 1}\right) r^2 - r'_{\text{Hex}} \cdot \left(\sqrt{r^2 - r'^2_{\text{Hex}}}\right) \tag{A.107}$$

Let us now express the total overlap $6S$ relative to the area of one circle, and call this the relative loss $l$.

$$l = \frac{6S}{\pi r^2} = \frac{6}{\pi}\left(\arctan\left(\sqrt{\frac{r^2}{r'^2_{\text{Hex}}} - 1}\right) - \frac{r'_{\text{Hex}}}{r} \cdot \left(\sqrt{1 - \frac{r'^2_{\text{Hex}}}{r^2}}\right)\right) \tag{A.108}$$

$$= \frac{6}{\pi}\left(\arctan\left(\sqrt{\frac{1}{c^2} - 1}\right) - c\sqrt{1 - c^2}\right) \tag{A.109}$$

Note that we substitute $\frac{r'_{\text{Hex}}}{r} =: c$. This makes sense since $r$ is a free parameter and $r'_{\text{Hex}}$ depends on $r$ and on the grey value of the image. The ratio $c$ thus allows us to regard the problem independent of the scale, i.e. the absolute error.

Let us forget the loss for a moment and consider the case $r \leqslant r'_{\text{Hex}}$ in which the rendering works as expected. Here, we obtain the grey value $g$ as the quotient of the area of the circular disc and the hexagon it covers:

$$g = \frac{A}{A_{\text{Hex}}} \overset{(A.59)}{=} \frac{\pi r^2}{\frac{3\sqrt{3}}{2}\left(\frac{2}{\sqrt{3}} r'_{\text{Hex}}\right)^2} \tag{A.110}$$

$$= \frac{\pi r^2}{2\sqrt{3} r'^2_{\text{Hex}}} \tag{A.111}$$

$$= \frac{\pi}{2\sqrt{3}} \cdot \frac{1}{c^2} \; . \tag{A.112}$$

If the rendering worked well in dark areas, we would also expect such $g$ in these cases. In other words, the relative loss $l$ just describes the deviation of the true grey value $x$ from the desired grey value $g$. The relative remainder $1 - l$ thus denotes the fraction of the area that is still rendered. Since the area and the grey value depend linearly on each other, we can express $x$ as

$$x = g(1-l) = \frac{\pi}{2\sqrt{3}} \cdot \frac{1}{c^2} \cdot \left(1 - \frac{6}{\pi}\left(\arctan\left(\sqrt{\frac{1}{c^2} - 1}\right) - c\sqrt{1-c^2}\right)\right) \; . \tag{A.113}$$

At this point, we can switch the roles of action and reaction: Considering a certain grey value $x$ shall be obtained, which grey value $g$ would we need to sample? Since (A.113) depends only on $x$ and $c$, we can solve it numerically for $c$ assuming a desired $x$. Since the function is not monotonic, we can use that we are only interested in

$$r'_{\text{Hex}} \leqslant r \leqslant r_{\text{Hex}} \tag{A.114}$$

$$\Leftrightarrow \quad 1 \leqslant \frac{1}{c} \leqslant \frac{2}{\sqrt{3}} \tag{A.115}$$

$$\Leftrightarrow \quad 1 \geqslant c \geqslant \frac{\sqrt{3}}{2} \tag{A.116}$$

(the equivalence follows from (A.60)). Finally, we use $c$ in (A.112) to obtain $g$. To finish this proof, let us check the global mass preservation of the process given that a tonemapping as in (7.28) is applied. Regions with a grey value below $\frac{\pi}{2\sqrt{3}}$ are rendered correctly by Theorem 7.1, since $T(x) = x$.

Above this threshold, the grey value preservation follows by construction from (A.113) and (A.112). Using the initial requirement of a perfect sampling operator, this local mass preservation carries over to the global case. ■

# Bibliography

## Own Publications


[BCGV11] M. Breuß, E. Cristiani, P. Gwosdek, and O. Vogel. An adaptive domain-decomposition technique for parallelisation of the fast marching method. *Applied Mathematics and Computation*, 218:32–44, September 2011.

[GBW08] P. Gwosdek, A. Bruhn, and J. Weickert. High performance parallel optical flow algorithms on the Sony PlayStation 3. In O. Deussen, D. Keim, and D. Saupe, editors, *Proc. 13th International Fall Workshop Vision, Modeling and Visualisation 2008*, pages 253–262, Konstanz, Germany, October 2008. AKA, Heidelberg, Germany.

[GBW10] P. Gwosdek, A. Bruhn, and J. Weickert. Variational optic flow on the Sony PlayStation 3 – accurate dense flow fields for real-time applications. *Journal of Real-Time Image Processing*, 5(3):163–177, September 2010.

[GGBW11] P. Gwosdek, S. Grewenig, A. Bruhn, and J. Weickert. Theoretical foundations of Gaussian convolution by extended box filtering. In A.M. Bruckstein and B. ter Haar Romeny, editors, *Scale Space and Variational Methods in Computer Vision*, volume 6667 of *Lecture Notes in Computer Science*, pages 447–458. Springer, Berlin, Germany, 2011.

[GSWT11] P. Gwosdek, C. Schmaltz, J. Weickert, and T. Teuber. Fast electrostatic halftoning. *Journal of Real-Time Image Processing*, December 2011. Online First.

[GZG+10] P. Gwosdek, H. Zimmer, S. Grewenig, A. Bruhn, and J. Weickert. A highly efficient GPU implementation for variational optic flow based on the Euler-Lagrange framework. In *Proc.*

*3rd ECCV Workshop Computer Vision with GPUs*, Heraklion, Greece, September 2010. Springer, Berlin, Germany.

[LZGW11] A. Luxenburger, H. Zimmer, P. Gwosdek, and J. Weickert. Fast PDE-based image analysis in your pocket. In A.M. Bruckstein and B. ter Haar Romeny, editors, *Scale Space and Variational Methods in Computer Vision*, volume 6667 of *Lecture Notes in Computer Science*, pages 544–555. Springer, Berlin, Germany, 2011.

[SGBW10] C. Schmaltz, P. Gwosdek, A. Bruhn, and J. Weickert. Electrostatic halftoning. *Computer Graphics Forum*, 29(8):2313–2327, December 2010.

[SGW11] C. Schmaltz, P. Gwosdek, and J. Weickert. Multi-class anisotropic electrostatic halftoning. *Computer Graphics Forum*, October 2011. Early View.

[TSG+11] T. Teuber, G. Steidl, P. Gwosdek, C. Schmaltz, and J. Weickert. Dithering by differences of convex functions. *SIAM Journal on Imaging Sciences*, 4(1):79–108, January 2011.

## Other References

[AD96] V. Aurich and U. Daub. Bilddatenkompression mit geplanten Verlusten und hoher Rate. In B. Jähne, P. Geißler, H. Haußecker, and F. Hering, editors, *Proc. 18. DAGM-Symposium*, Informatik Aktuell, pages 138–146, Heidelberg, Germany, September 1996. Springer, Berlin.

[Ado99] Adobe Systems Inc. *PostScript Language Reference*. Addison-Wesley, Reading, MA, 3rd edition, February 1999.

[ADSW02] L. Alvarez, R. Deriche, J. Sánchez, and J. Weickert. Dense disparity map estimation respecting image derivatives: a PDE and scale-space based approach. *Journal of Visual Communication and Image Representation*, 13(1/2):3–21, 2002.

[Adv11] Advanced Micro Devices, Incorporated. *AMD Accelerated Parallel Processing OpenCL Programming Guide*, August 2011. Online: `http://developer.amd.com/sdks/AMDAPPSDK/`

`assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf`, Retrieved 11-09-22.

[AHAS08] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan. GpuCV: A GPU-accelerated framework for image processing and computer vision. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. Porikli, J. Peters, J. Klosowski, L. Arns, Y.K. Chun, T.-M. Rhyne, and L. Monroe, editors, *Advances in Visual Computing*, volume 5359 of *Lecture Notes in Computer Science*, pages 430–439. Springer, Berlin, Germany, 2008.

[AL96] J. Allebach and Q. Lin. Fm screen design using dbs algorithm. In *Proceedings of 3rd IEEE International Conference on Image Processing*, volume 3, pages 549–552. IEEE Press, September 1996.

[App85] Andrew W. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(1):85–103, January 1985.

[ARBJ03] S. Agarwal, R. Ramamoorthi, S. Belongie, and H.W. Jensen. Structured importance sampling of environment maps. *ACM Transactions on Graphics (Proc. SIGGRAPH '03)*, 22(3):605–612, 2003.

[AU10] W. Arendt and K. Urban. *Partielle Differenzialgleichungen: Eine Einführung in analytische und numerische Methoden.* Spektrum Akademischer Verlag, Heidelberg, Germany, 2010.

[BA83] P. J. Burt and E. H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31:532–540, 1983.

[BA96] M. J. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, January 1996.

[Bar78] M. S. Bartlett. *An Introduction to Stochastic Processes with Special Reference to Methods and Applications.* Cambridge University Press, Cambridge, UK, third edition, 1978.

[BAR11] R. Ben-Ari and G. Raveh. Variational depth from defocus in real-time. In C. Sagiv, editor, *Proc. 3rd Workshop on GPUs for Vision (in conjunction with ICCV 2011)*, Barcelona, Spain, November 2011. To appear.

[BAS10] R. Ben-Ari and N. Sochen. Stereo matching with Mumford-Shah regularization and occlusion handling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(11):2071–2084, 2010.

[Bay73] B.E. Bayer. An optimum method for two-level rendition of continuous-tone pictures. In *Proc. IEEE International Conference on Communications*, volume 26, pages 2611–2615. IEEE Press, 1973.

[BBBW09] Z. Belhachmi, D. Bucur, B. Burgeth, and J. Weickert. How to choose interpolation data in images. *SIAM Journal on Applied Mathematics*, 70(1):333–352, 2009.

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optic flow estimation based on a theory for warping. In T. Pajdla and J. Matas, editors, *Computer Vision – ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin, 2004.

[BC03] G. Beylkin and R. Cramer. A multiresolution approach to regularization of singular operators and fast summation. *SIAM Journal on Scientific Computing*, 24(1):81–117, 2003.

[BCR91] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms I. *Communications on Pure and Applied Mathematics*, 44(2):141–183, 1991.

[BD96] F.A. Bornemann and P. Deuflhard. The cascadic multigrid method for elliptic problems. *Numerische Mathematik*, 75:135–152, 1996.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[BFB94] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, February 1994.

[BH86] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(n \log n)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[BL77] Jr. B.F. Logan. Information in the zero crossings of bandpass signals. *Bell Systems Technical Journal*, 56:487–510, April 1977.

[BLF+07] J. Beyer, C. Langer, L. Fritz, M. Hadwiger, S. Wolfsberger, and K. Bühler. Interactive diffusion-based smoothing and segmentation of volumetric datasets on graphics hardware. *Methods of Information in Medicine*, 46(3):270–274, 2007.

[BM58] G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, January 1958.

[BPT88] M. Bertero, T. A. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76(8):869–889, August 1988.

[Bra99] R. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, New York, NY, 3rd edition, 1999.

[Bri88] E.O. Brigham. *The fast Fourier transform and its applications*. Prentice-Hall signal processing series. Prentice Hall, 1988.

[Bru10] A. Bruhn. Normalised convolution for efficient image inpainting. Personal communication, 2010.

[BS08] P. Babenko and M. Shah. MinGPU: A minimum GPU library for computer vision. *Journal of Real-Time Image Processing*, 3(4):255–268, April 2008.

[BSB10] A. Bhatia, W.E. Snyder, and G. Bilbro. Stacked integral image. In *Proc. 2010 IEEE International Conference on Robotics and Automation*, pages 1530–1535, Anchorage, AK, May 2010. IEEE Computer Society.

[BSCB00] M. Bertalmío, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proc. SIGGRAPH '00*, pages 417–424, 2000.

[BSD09] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics*, 28(3):86:1–8, 2009.

[BSL+11] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[BSMM08] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig, editors. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, Germany, 2008.

[But05] T. Butz. *Fourier Transformation for Pedestrians*. Springer, December 2005.

[BW05] A. Bruhn and J. Weickert. Towards ultimate motion estimation: Combining highest accuracy with real-time performance. In *Proc. Tenth International Conference on Computer Vision*, volume 1, pages 749–755, Beijing, China, October 2005. IEEE Computer Society Press.

[BWF+05] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr. Variational optical flow computation in real-time. *IEEE Transactions on Image Processing*, 14(5):608–615, May 2005.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

[CAO09] J. Chang, B. Alain, and V. Ostromoukhov. Structure-aware error diffusion. *ACM Transactions on Graphics*, 28(5):162:1–162:8, December 2009.

[CBAB97] P. Charbonnier, L. Blanc–Féraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6(2):298–311, 1997.

[CCL69] F.W. Campbell, R.H.S. Carpenter, and J.Z. Levinson. Visibility of aperiodic patterns compared with that of sinusoidal gratings. *Journal of Physiology*, 204:283–298, 1969.

[CD01] J. Cohen and P. Debevec. LightGen plugin for HDR Shop. Online: `http://gl.ict.usc.edu/HDRShop/lightgen/`, 2001. Retrieved 2010-03-25.

[Cha04] A. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1–2):89–97, 2004.

[Cla61] C. E. Clark. Importance sampling in monte carlo analyses. *Operations Research*, 9(5):603–620, September 1961.

[CLSY93] J.-C. Chen, D. Lu, J.S. Sadowsky, and K. Yao. On importance sampling in digital communications. i. fundamentals. *IEEE Journal on Selected Areas in Communications*, 11(3):289–299, April 1993.

[CN93] T.J. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture hardware. Technical Report TR93-027, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1993.

[Coo86] R. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[Cor11] NVidia Corporation. NVidia Performance Primitives. Online: `http://developer.nvidia.com/npp`, Retrieved: 2011-09-25, 2011.

[CR11] CNet Reviews. Printers. Online: `http://reviews.cnet.com/computer-printers/?sa=1105299&tag=topPanelArea.1`, 2011. Retrieved 2011-04-06.

[Cro84] F.C. Crow. Summed-area tables for texture mapping. *Computer Graphics (Proc. SIGGRAPH '84)*, 4(9):207–212, 1984.

[CSHD03] M.F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics (Proc. SIGGRAPH '03)*, 22(3):287–294, 2003.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[Der87a] R. Deriche. Separable recursive filtering for efficient multi-scale edge detection. In *Proc. International Workshop on Industrial Applications of Machine Intelligence and Vision*, Roppongi, Yokyo, Japan, February 1987. IEEE Computer Society. Out of print, cited in [Der87b].

[Der87b] R. Deriche. Using canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, 1(2):167–187, 1987.

[DHL+98] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. *Computer Graphics (Proc. SIGGRAPH '98)*, pages 275–286, 1998.

[dJ07] J. de Jong. Gaussian blur in GIMP compared to Photoshop. Online: `http://www.gimpusers.com/forums/gimp-developer/6172-gaussian-blur-in-gimp-compared-to-photoshop`, November 2007. Retrieved 2010-02-01.

[EG01] J.H. Elder and R.M. Goldberg. Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):291–296, March 2001.

[EJ10] P. Étoré and B. Jourdain. Adaptive optimal allocation in stratified sampling methods. *Methodology and Computing in Applied Probability*, 12(3):335–360, 2010.

[EK91] R. Eschbach and K. T. Knox. Error-diffusion algorithm with edge enhancement. *Journal of the Optical Society of America A*, 8(12):1844–1850, 1991.

[Els61] L. E. Elsgolc. *Calculus of Variations*. Pergamon, Oxford, 1961.

[EVB01] J. Eberspächer, H-J. Vögel, and C. Bettstetter. *GSM Global System for Mobile Communication*. Teubner, Leipzig, third edition, 2001.

[Fat11] R. Fattal. Blue-noise point sampling using kernel density model. *ACM Transactions on Graphics (Proc. SIGGRAPH '11)*, 28(3):48:1–10, 2011.

[Fel08] M. Felsberg. On the relation between anisotropic diffusion and iterated adaptive filtering. In G. Rigoll, editor, *Pattern Recognition*, volume 5096 of *Lecture Notes in Computer Science*, pages 436–445. Springer, Berlin, 2008.

[FG87] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *Proc. ISPRS Intercommission Conference*

*on Fast Processing of Photogrammetric Data*, pages 281–305, Interlaken, Switzerland, June 1987.

[FHH93] D. J. Field, A. Hayes, and R. F. Hess. Contour integration by the human visual system: Evidence for a local "association field". *Vision Research*, 33(2):173–193, 1993.

[FJ90] D. J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1):77–104, August 1990.

[FJ05] M. Frigo and S.G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[Flo97] L. Florack. *Image Structure*, volume 10 of *Computational Imaging and Vision*. Kluwer, Dordrecht, 1997.

[FMA05] J. Fung, S. Mann, and C. Aimone. OpenVIDIA: Parallel GPU computer vision. In H. Zhang, T.-S. Chua, R. Steinmetz, M.S. Kankanhalli, and L. Wilcox, editors, *Proc. 13th ACM International Conference on Multimedia*, pages 849–852, Singapore, November 2005. ACM.

[FPZ03] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proc. 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Madison, WI, June 2003. IEEE Computer Society Press.

[FS76] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial grey scale. In *Proc. Society of Information Display*, volume 17, pages 75–77, 1976.

[FS02] M. Fenn and G. Steidl. FMM and $\mathcal{H}$-matrices: A short introduction to the basic idea. Technical Report TR-2002-008, Department for Mathematics and Computer Science, University of Mannheim, Germany., 2002.

[FS04] M. Fenn and G. Steidl. Fast NFFT based summation of radial functions. *Sampling Theory in Signal and Image Processing*, 3(1):1–28, 2004.

[Fv06] O. Fialka and M. Čadík. FFT and convolution performance in image filtering on GPU. In *Proc. 10th IEEE Conference on*

*Information Visualization*, pages 609–614, Washington, DC, 2006. IEEE Computer Society Press.

[GBAL06] A.J. González, J. Bacca, G.R. Arce, and D.L. Lau. Alpha stable human visual system models for digital halftoning. *Proceedings of SPIE*, 6057, February 2006.

[GBW09] M. Ghodstinat, A. Bruhn, and J. Weickert. Deinterlacing with motion-compensated anisotropic diffusion. In D. Cremers, B. Rosenhahn, A. Yuille, and F. Schmidt, editors, *Statistical and Geometrical Approaches to Visual Motion Analysis*, volume 5604 of *Lecture Notes in Computer Science*, pages 91–106, Berlin, 2009. Springer.

[GD08] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227(18):8290–8313, September 2008.

[Gen79] W. Gentzsch. Numerical solution of linear and non-linear parabolic differential equations by a time-discretisation of third order accuracy. In *Proc. Third GAMM Conference on Numerical Methods in Fluid Mechanics*, pages 109–117. Vieweg, 1979.

[Ghi11] G. Ghimpeteanu. Non-photorealistic rendering of 2D images. Master's thesis, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany, 2011.

[GHS99] P. Glasserman, P. Heidelberger, and P. Shahabuddin. Asymptotically optimal importance sampling and stratification for pricing path-dependent options. *Mathematical Finance*, 9(2):117–152, 1999.

[GK80] K. Glashoff and H. Kreth. Vorzeichenstabile differenzenverfahren für parabolische anfangswertaufgaben. *Numerische Mathematik*, 35(3):343–354, 1980.

[GLD+08] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High performance discrete Fourier transforms on graphics processors. In *Proc. 2008 ACM/IEEE Conference on Supercomputing*, pages 2:1–2:12. IEEE Press, 2008.

[GO96] G. Golub and J.M. Ortega. *Scientific computing: Eine Einführung in das wissenschaftliche Rechnen und die parallele Numerik.* Teubner, Stuttgart, Germany, 1996.

[Göd05] D. Göddeke. GPGPU–Basic math tutorial. Technical Report 300, Fachbereich Mathematik, Universität Dortmund, November 2005.

[Goo51] W.M. Goodall. Television by pulse code modulation. *Bell Systems Technical Journal*, 30:33–49, 1951.

[Gou85] A.R. Gourlay. Implicit convolution. *Image and Vision Computing*, 3(1):15–23, February 1985.

[GPS11] M. Gräf, D. Potts, and G. Steidl. Quadrature errors, discrepancies and their relations to halftoning on the torus and the sphere. Technical Report 2011-05, Fakultät für Mathematik, Technical University of Chemnitz, Chemnitz, Germany, 2011.

[GR87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, December 1987.

[Gre08] A. Gregerson. Implementing fast MRI gridding on GPUs via CUDA. NVidia Whitepaper, Online: `http://cn.nvidia.com/docs/IO/47905/ECE757_Project_Report_Gregerson.pdf`, 2008. Retrieved 2011-04-11.

[GRS93] R. Geist, R. Reynolds, and D. Suggs. A Markovian framework for digital halftoning. *ACM Transactions on Graphics*, 12(2):136–159, 1993.

[GT08] H. Grossauer and P. Thoman. GPU-based multigrid: Real-time performance in high resolution nonlinear image processing. In A. Gasteratos, M. Vincze, and J. K. Tsotsos, editors, *Computer Vision Systems*, volume 5008 of *Lecture Notes in Computer Science*, pages 141–150. Springer, Berlin, 2008.

[GW08] R. C. Gonzalez and R. E. Woods. *Digital image processing.* Pearson Prentice Hall, third edition, 2008.

[GWB10] S. Grewenig, J. Weickert, and A. Bruhn. From box filtering to fast explicit diffusion. In M. Goesele, S. Roth, A. Kuijper, B. Schiele, and K. Schindler, editors, *Pattern Recognition*, volume 6376 of *Lecture Notes in Computer Science*, pages 533–542. Springer, Berlin, 2010.

[GWL+03] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware (HWWS 03)*, pages 102–111, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[GWW+08] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev, and H.-P. Seidel. Image compression with anisotropic diffusion. *Journal of Mathematical Imaging and Vision*, 31(2–3):255–269, July 2008.

[Hac85] W. Hackbusch. *Multigrid Methods and Applications*. Springer, New York, 1985.

[Hac99] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62:89–108, 1999.

[Hae90] P. Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics (Proc. SIGGRAPH '90)*, 24(4):207–214, 1990.

[Hal60] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[Hal06a] D. Hale. The Mines Java toolkit. Online: `http://inside.mines.edu/~dhale/jtk/`, 2006. Retrieved 2011-05-27.

[Hal06b] D. Hale. Recursive Gaussian filters. Technical Report CWP-546, Center for Wave Phenomena, Colorado School of Mines, CO, 2006.

[Ham07] M.J. Hammel. *The Artist's Guide to GIMP Effects*. No Starch Press, first edition, August 2007.

[Han05] K.M. Hanson. Halftoning and Quasi-Monte Carlo. In K. M. Hanson and F. M. Hemez, editors, *Sensitivity Analysis of Model Output*, pages 430–442. Los Alamos Research Library, 2005.

[Har11] M. Harris. Optimising parallel reduction in CUDA. Talk slides, online: `http://developer.download.nvidia.com/`

`compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`, Retrieved: 2011-09-23, 2011.

[HD07] F. Huguet and F. Devernay. A variational method for scene flow estimation from stereo sequences. In *Proc. 11th International Conference on Computer Vision*, pages 1–7, Rio de Janeiro, Brazil, 2007. IEEE Computer Society Press.

[HE81] R. Hockney and J. Eastwood. *Computer simulation using particles*. McGraw-Hill, New York, NY, 1981.

[Hen07] J. Hensley. Close to the metal. ACM SIGGRAPH Course on GPGPU: General-Purpose Computation on Graphics Hardware. Talk slides, online: `http://http://developer.amd.com/media/gpu_assets/Hensley-Close_to_the_Metal(Siggraph07_GPGPUCourse).pdf`, Retrieved: 2011-09-25, 2007.

[Hes07] R. Hess, editor. *The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender*. No Starch Press, September 2007.

[Hol26] L.A. Holman. *The Graphic Process, a Series of Actual Prints*. Charles E. Goodspeed & Co., Boston, MA, 1926.

[How10] M. Howison. Comparing gpu implementations of bilateral and anisotropic diffusion filters for 3d biomedical datasets. Technical Report LBNL-3425E, Lawrence Berkeley National Laboratory, Berkeley, CA, 2010.

[HS81] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[Iij59] T. Iijima. Basic theory of pattern observation. In *Papers of Technical Group on Automata and Automatic Control*. IECE, Japan, December 1959. In Japanese.

[IKN98] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, November 1998.

[Ilb00] P.W.M. Ilbery. U.S. patent No. 6,124,844, 2000.

[JCW09] S. Jeschke, D. Cline, and P. Wonka. A GPU Laplacian solver for diffusion curves and poisson image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia '09)*, 28(5):116:1–8, 2009.

[JJN76] J. F. Jarvis, C. N. Judice, and W. H. Ninke. A survey of techniques for the display of continuous tone pictures on bilevel displays. *Computer Graphics and Image Processing*, 5:13–40, 1976.

[JLR07] K. Johansson, P. Lundberg, and R. Ryberg. *A Guide to Graphic Print Production*. Wiley, second edition, 2007.

[Joh28] J.B. Johnson. Thermal agitation of electricity in conductors. *Physical Review*, 32(1):97, July 1928.

[JR76] J. F. Jarvis and C. S. Roberts. A new technique for displaying continuous tone images on a bilevel display. *IEEE Transactions on Communications*, 24:891–898, 1976.

[JW07] W.-K. Jeong and T. Whitaker. A fast iterative method for a class of Hamilton-Jacobi equations on parallel sytems. Technical Report UUCS-07-010, School of Computing, University of Utah, UT, April 2007.

[KA02] S.H. Kim and J.P. Allebach. Impact of HVS models on model-based halftoning. *IEEE Transactions on Image Processing*, 11(3):258–269, March 2002.

[Kan99] H.R. Kang. *Digital color halftoning*. SPIE/IEEE series on imaging science & engineering. SPIE Press, 1999.

[KCDL06] J. Kopf, D. Cohen-Or, O. Deussen, and D. Lischinski. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics*, 25(3):509–518, 2006.

[Khr10] Khronos Group. *The OpenCL Specification*, June 2010. Online: `http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf`, Retrieved 11-09-22.

[Kip01] H. Kipphan. *Handbook of print media: Technologies and production methods*. Springer, 2001.

[KK95] S. Krishnan and L. V. Kalé. A parallel adaptive fast multipole algorithm for n-body problems. In K. Gallivan, editor, *Proc. 1995 International Conference on Parallel Processing*, volume 3, pages 46–51, Urbana-Champain, IL, 1995. CRC Press, Inc.

[KK03] T. Kollig and A. Keller. Efficient illumination by high dynamic range images. In P. Christensen and D. Cohen-Or, editors, *EGRW '03: Proceedings of the 14th Eurographics Workshop on Rendering*, pages 45–50, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[KKP09] J. Keiner, S. Kunis, and D. Potts. Using nfft 3 – a software library for various nonequispaced fast fourier transforms. *ACM Transactions on Mathematical Software*, 36(4):19:1–23, 2009.

[KM79] L. Kuhn and R. A. Myers. Ink-jet printing. *Scientific American*, 240(4):162–178, April 1979.

[Kno89] K.T. Knox. Edge enhancement in error diffusion. In *Paper Summaries from The Society for Imaging Science and Technology, 42nd Annual Conference*, pages 310–313, Boston, May 1989.

[Knu87] D.E. Knuth. Digital halftones by dot diffusion. *ACM Transactions on Graphics*, 6(4):245–273, 1987.

[Kre05] U. Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Aufbaukurs Mathematik. Vieweg, 2005.

[KS80] J.F. Kaiser and R.W. Schafer. On the use of the Io-sinh window for spectrum analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):105–107, February 1980.

[KW93] H. Knutsson and C.-F. Westin. Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data. In *Proc. 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 515–523, New York, NY, 1993. IEEE Computer Society Press.

[KZ96] R. Klette and P. Zamperoni. *Handbook of Image Processing Operators*. Wiley, New York, 1996.

[LA08] D.L. Lau and G.R. Arce. *Modern digital halftoning*. Signal Processing and Communications Series. CRC Press, 2008.

[Lag73] J. L. Lagrange. *Œvres de Lagrange*. Gauthier-Villars, January 1873.

[LAG98] D.L. Lau, G.R. Arce, and N.C. Gallagher. Green-noise digital halftoning. *Proceedings of the IEEE*, 86:2424–2444, December 1998.

[LD05] A. Lagae and P. Dutré. A procedural object distribution function. *ACM Transactions on Graphics (Proc. SIGGRAPH '05)*, 24(4):1442–1461, 2005.

[Les06] P.M. Lester. *Visual communication: Images with messages*. Thomson Wadsworth, Belmont, CA, 4th edition, 2006.

[Lim69] J.O. Limb. Design of dither waveforms for quantized visual signals. *Bell Systems Technical Journal*, 48(7):2555–2582, 1969.

[Lin90] T. Lindeberg. Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:234–254, 1990.

[Lin94] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer, Boston, 1994.

[LK73] B. Lippel and M. Kurland. The effect of dither on luminance quantization of pictures. *IEEE Transactions on Communication Technology*, 19(6):879–888, 1973.

[LK81] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Seventh International Joint Conference on Artificial Intelligence*, pages 674–679, Vancouver, Canada, August 1981.

[LK11] W. Lin and C.-C.J. Kuo. Perceptual visual quality metrics: A survey. *Journal of Visual Communication and Image Representation*, 22(4):297–312, May 2011.

[LKC+10] V.W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A.D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on

CPU and GPU. *ACM SIGARCH Computer Architecture News (Proc. ISCA '10)*, 38(3):451–460, June 2010.

[LNW+09] H. Li, D. Nehab, L.-Y. Wei, P. Sander, and C.-W. Fu. Fast capacity constrained voronoi tessellation. Technical Report MSR-TR-2009-174, Microsoft Research, 2009.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[MA03] K. Moreland and E. Angel. The FFT on a GPU. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware (HWWS 03)*, pages 112–119, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[Mar82] D. Marr. *Vision*. Freeman, San Francisco, 1982.

[MBO92] B. Merriman, J. Bence, and S. Osher. Diffusion generated motion by mean curvature. In J. E. Taylor, editor, *Computational Crystal Growers Workshop*, Selected Lectures in Math, pages 73–83. American Mathematical Society, Providence, RI, 1992.

[MBWF11] M. Mainberger, A. Bruhn, J. Weickert, and S. Forchhammer. Edge-based image compression of cartoon-like images with homogeneous diffusion. *Pattern Recognition*, 44(9):1859–1873, September 2011.

[Mei05] A. Meister. *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren*. Vieweg+Teubner, second edition, 2005.

[Mer40] Mergenthaler Linotype Company. *Linotype machine principles*, 1940.

[Mes06] D. Meschede. *Gerthsen Physik*. Springer, 23th edition, March 2006.

[MF92] M. McCool and E. Fiume. Hierarchical Poisson disk sampling distributions. In K.S. Booth and A. Fournier, editors, *Proc. Conference on Graphics Interfaces '92*, pages 94–105, San Francisco, CA, 1992. Morgan Kaufmann.

[MGW05] M.D. Meyer, P. Georgel, and R.T. Whitaker. Robust particle systemsfor curvature dependent sampling of implicit surfaces. In *Proc. 2005 International Conference on Shape Modeling and Applications*, pages 124–133. IEEE Computer Society, 2005.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London, Series B*, 207:187–217, 1980.

[MM98] S. Masnou and J.-M. Morel. Level lines based disocclusion. In *Proc. 1998 IEEE International Conference on Image Processing*, volume 3, pages 259–263, Chicago, IL, October 1998.

[MP95] S. Mann and R.W. Picard. On being 'undigital' with digital cameras: Extending dynamic range by combining dierently exposed pictures. In *Proc. 48th IS&T Annual Conference*, pages 442–448, Washington,DC, May 1995.

[MP08] J. McCann and N.S. Pollard. Real-time gradient-domain painting. *ACM Transactions on Graphics (Proc. SIGGRAPH '08)*, 27(3):93:1–8, 2008.

[MPS+09] D. Mitzel, T. Pock, T. Schoenemann, , and D. Cremers. Video super resolution using duality based tv-$l^1$ optical ow. In J. Denzler, G. Notni, and H. Süße, editors, *Pattern Recognition*, volume 5748 of *Lecture Notes in Computer Science*, pages 432–441. Springer, Berlin, 2009.

[MS00] M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26(3):436–461, September 2000.

[Mur36] K. Murray. Half-tone engravings easily made with ordinary camera. *Popular Science Monthly*, 129(3):76–79, 1936.

[MW09] M. Mainberger and J. Weickert. Edge-based image compression with homogeneous diffusion. In Xiaoyi Jiang and Nicolai Petkov, editors, *Proc. 13th International Conference on Computer Analysis of Images and Patterns*, volume 5702 of *Lecture Notes in Computer Science*, pages 476–483. Springer, Berlin, September 2009.

[Näs84] R. Näsänen. Visibility of half-tone dot textures. *IEEE Transactions on Systems, Man, and Cybernetics*, 14:920–924, 1984.

[NE86] H.-H. Nagel and W. Enkelmann. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:565–593, 1986.

[Ngu07] H. Nguyen, editor. *GPU Gems 3*. Addison–Wesley Professional, August 2007.

[NMLH11] D. Nehab, A. Maximo, R.S. Lima, and H. Hoppe. GPU-efficient recursive filtering and summed area tables. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia '11)*, 30(6), 2011. To appear.

[Nor60] E. Norman. A discrete analogue of the Weierstrass transform. *Proceedings of the American Mathematical Society*, 11(596–604), 1960.

[NVi10a] NVidia Corporation. *NVidia CUDA CUFFT Library*, 3.1 edition, May 2010. Online: `http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf`, Retrieved 10-11-24.

[NVi10b] NVidia Corporation. *NVidia CUDA CURAND Library*, 3.2 edition, August 2010. Online: `http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CURAND_Library.pdf`, Retrieved 10-11-24.

[NVi11a] NVidia Corporation. *Getting started with CUDA SDK samples*, May 2011. Online: `http://developer.download.nvidia.com/compute/DevZone/docs/html/doc/release/Getting_Started_With_CUDA_SDK_Samples.pdf`, Retrieved 11-08-04.

[NVi11b] NVidia Corporation. *NVidia CUDA Programming Guide*, 4th edition, March 2011. Online: `http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf`, Retrieved 11-06-21.

[Nyq28] H. Nyquist. Thermal agitation of electric charge in conductors. *Physical Review*, 32(1):110–113, July 1928.

[OBW+08] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: A vector representation for

smooth-shaded images. *ACM Transactions on Graphics (Proc. SIGGRAPH '08)*, 27(3):92:1–8, 2008.

[OH95] V. Ostromoukhov and R.D. Hersch. Artistic screening. In *Proc. SIGGRAPH '95*, pages 219–228, New York, NY, August 1995.

[ON95] M. Otte and H.-H. Nagel. Estimation of optical flow based on higher-order spatiotemporal derivatives in interlaced and non-interlaced image sequences. *Artificial Intelligence*, 78:5–43, 1995.

[Ope11] OpenCV dev team. *The OpenCV Reference Manual*, August 2011. Online: `https://code.ros.org/svn/opencv/trunk/opencv/doc/opencv2refman.pdf`, Retrieved 11-09-25.

[Ost01] V. Ostromoukhov. A simple and efficient error-diffusion algorithm. In E. Fiume, editor, *Proc. SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 567–572, Los Angeles, CA, August 2001.

[Par62] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[Pat07] N.S. Patil. Fast implementation of anisotropic diffusion using the CUDA technology on the GPU GeForce 8800 gtx. Master's thesis, Graduate Faculty, Rensselaer Polytechnic Institute, Troy, NY, December 2007.

[PB94] Yachin Pnueli and Alfred M. Bruckstein. Dig$^{\text{i}}_{\text{D}}$ürer. *The Visual Computer*, 10(5):277–292, 1994.

[PBH+11] T. Pock, H. Bischof, S. Heber, R. Ranftl, M. Unger, and M. Werlberger. GPU4Vision. Online: `http://gpu4vision.icg.tugraz.at/`, Retrieved: 2011-09-23, 2011.

[Per85] K. Perlin. An image synthesizer. *Computer Graphics (Proc. SIGGRAPH '85)*, 19(3):287–296, 1985.

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan-Kaufmann, second edition, July 2010.

[PM87] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. In *Proc. IEEE Computer Society Workshop on Computer Vision*, pages 16–22, Miami Beach, FL, November 1987. IEEE Computer Society Press.

[Pod07] V. Podlozhnyuk. Parallel mersenne twister. NVidia Whitepaper, Online: `http://developer.download.nvidia.com/compute/cuda/sdk/website/C/src/MersenneTwister/doc/MersenneTwister.pdf`, 2007. Retrieved 2011-04-14.

[PQW+08] W.-M. Pang, Y. Qu, T.-T. Wong, D. Cohen-Or, and P.-A. Heng. Structure-aware halftoning. *ACM Transactions on Graphics*, 27(3):89:1–89:8, August 2008.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[Pra98] R. Prasad. *Universal Wireless Personal Communications*. Artech House, Norwood, MA, 1998.

[PST00] D. Potts, G. Steidl, and M. Tasche. Fast Fourier transforms for non-equispaced data: A tutorial. In J. J. Benedetto and P. J. S. G. Ferreira, editors, *Modern Sampling Theory: Mathematics and Applications*, Applied and Numerical Harmonic Analysis, chapter 12, pages 251–274. Birkhäuser Boston, December 2000.

[PTG94] W. Purgathofer, R. F. Tobler, and M. Geiler. Forced random dithering: Improved threshold matrices for ordered dithering. In *Proceedings of 1st IEEE International Conference on Image Processing*, volume 2, pages 1032–1035, Austin, Texas, November 1994. Also in Graphics Gems 5, p 297–301.

[Pus11] S. Pushkarev. 3d nonlinear diffusion filtering on GPUs using an AOS scheme. Master's thesis, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany, 2011.

[Rad68] C.M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56:1107–1108, 1968.

[RB85] W.T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics (Proc. SIGGRAPH '85)*, 19(3):313–322, 1985.

[RBK98] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, January 1998.

[Ree83] W. T. Reeves. Particle systems — a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, April 1983.

[Rey87] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics (Proc. SIGGRAPH '87)*, 21(4):25–34, 1987.

[RFSB10] J. Rosner, H. Fassold, P. Schallauer, and W. Bailer. Fast gpu-based image warping and inpainting for frame interpolation. In V. Skala and E.M. Hitzer, editors, *Proc. Computer Graphics, Computer Vision and Mathematics (GraVisMa) Workshop*, pages D83:1–6, Brno, Czech Republic, September 2010. UNION Agency Science Press.

[Ril09] D. Riley. GPU processing methods for machine vision. Project whitepaper for the Advanced Computer Graphics class at the University of Maryland, Baltimore County, online: `http://www.cs.umbc.edu/~olano/635s09/d17.pdf`, Retrieved: 2011-07-05, April 2009.

[Rob62] L.G. Roberts. Picture coding using pseudo-random noise. *IRT Transactions on Information Theory*, 8(2):145–154, February 1962.

[RS01] M. Rumpf and R. Strzodka. Nonlinear diffusion in graphics hardware. In *Proc. Joint Eurographics – IEEE TCVG Symposium on Visualization*, Ascona, Switzerland, May 2001. Springer.

[Rum88] S.M. Rump. Algorithms for verified inclusions – Theory and practice. *Perspectives in Computing*, 19:109–126, 1988.

[SA85] R. L. Stevenson and G. R. Arce. Binary display of hexagonally sampled continuous-tone images. *Journal of the Optical Society of America A*, 2(7):1009–1013, March 1985.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, second edition, 2003.

[SAG03] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: A local parameterization approach. In *Proc. 12th International Meshing Roundtable*, 2003.

[SAH91] E. P. Simoncelli, E. H. Adelson, and D. J. Heeger. Probability distributions of optical flow. In *Proc. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 310–315, Maui, HI, June 1991. IEEE Computer Society Press.

[SBD81] R. Shaw, P. D. Burns, and J. C. Dainty. Particulate model for halftone in electrophotography. I. Theory and II. Experimental verification. *Proceedings of the SPIE*, 310:137–150, 1981.

[SC94] R. Szeliski and J. Coughlan. Hierarchical spline-based image registration. In *Proc. 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 194–201, Seattle, WA, June 1994. IEEE Computer Society Press.

[Sch11] C. Schmaltz. Additive choice of the non-overlap weight for multi-class halftoning and sampling. Personal communication, 2011.

[Sec02] A. Secord. Weighted Voronoi stippling. In *Proc. 2nd International Symposium on Non-Photorealistic Animation and Rendering*, pages 37–43. ACM Press, 2002.

[Sen11] A. Senefelder. *The invention of lithography.* The Fuchs & Lang Manufacturing Company, New York, NY, 1911.

[Shi91] P. Shirley. Discrepancy as a quality measure for sample distributions. In *Proc. Eurographics '91*, pages 183–194. Elsevier Science Publishers, 1991.

[SHS02] A. Secord, W. Heidrich, and L. Streit. Fast primitive distribution for illustration. In S. Gibson and P. Debevec, editors, *Proc. 13th Eurographics Workshop on Rendering*, pages 215–226. Eurographics Association, 2002.

[SKB+11] A. Schwarzkopf, T. Kalbe, C. Bajaj, A. Kuijper, and M. Goesele. Volumetric nonlinear anisotropic diffusion on GPUs. In A.M. Bruckstein and B. ter Haar Romeny, editors, *Scale Space and Variational Methods in Computer Vision*, volume 6667 of *Lecture Notes in Computer Science*, pages 555–566. Springer, Berlin, Germany, 2011.

[Smi04] G.D. Smith. *Numerical solution of partial differential equations: Finite difference methods*. Oxford University Press, New York, NY, third edition, 2004.

[SNFJ97] J. Sporring, M. Nielsen, L. Florack, and P. Johansen, editors. *Gaussian Scale-Space Theory*, volume 8 of *Computational Imaging and Vision*. Kluwer, Dordrecht, 1997.

[Soi99] P. Soille. *Morphological Image Analysis*. Springer, second edition, 1999.

[SP01] X. Sun and N. P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2):289–300, February 2001.

[Sri02] R. Srinivasan. *Importance sampling: Applications in communications and detection*. Springer, Berlin, 2002.

[SRLB08] D. Sun, S. Roth, J. P. Lewis, and M. J. Black. Learning optical flow. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008, Part III*, volume 5304 of *Lecture Notes in Computer Science*, pages 83–97. Springer, Berlin, 2008.

[SSG+08] M. Silberstein, A. Schuster, D. Geiger, A. Patney, and J.D. Owens. Efficient computation of sum-products on gpus through software-managed cache. In *Proc. 22nd Annual International Conference on Supercomputing (ICS '08)*, pages 309–318, New York, NY, 2008. ACM.

[SSNH08] T.S. Sørensen, T. Schaeffter, K.Ø. Noe, and M.S. Hansen. Accelerating the nonequispaced fast Fourier transform on commodity graphics hardware. *IEEE Transactions on Medical Imaging*, 27(4):538–547, April 2008.

[Sto66] T. G. Stockham. High speed convolution and correlation. In *Proc. 1966 Spring Joint Computer Conference*, volume 28, pages 229–233, Washington, 1966. Spartan Books.

[Stu81] P. Stucki. MECCA – a multiple error correcting computation algorithm for bilevel image hardcopy reproduction. Technical Report RZ1060, IBM Research Lab, Zurich, Switzerland, 1981.

[Sul03] R. Sullivan. *100 photographs that changed the world*. Life Books, New York, NY, 2003.

[SW87] I. Simpson and L. Wood. *The encyclopedia of drawing techniques*. Headline, 1987.

[SWB09] C. Schmaltz, J. Weickert, and A. Bruhn. Beating the quality of JPEG 2000 with anisotropic diffusion. In J. Denzler, G. Notni, and H. Süße, editors, *Pattern Recognition*, Lecture Notes in Computer Science, pages 452–461, Berlin, 2009. Springer.

[SWD05] S. Schenke, B.C. Wünsche, and J. Denzler. GPU-based volume segmentation. In *Proc. 2005 Conference on Image and Vision Computing New Zealand (IVCNZ '05)*, pages 31:1–6, Dunedin, New Zealand, November 2005. Published online: `http://pixel.otago.ac.nz/ipapers/31.pdf`, retrieved 2011-09-23.

[TA77] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill–Posed Problems*. Wiley, Washington, DC, 1977.

[TK91] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[TOS01] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, San Diego, 2001.

[TP84] O. Tretiak and L. Pastor. Velocity estimation from image sequences with second order differential operators. In *Proc. Seventh International Conference on Pattern Recognition*, pages 16–19, Montreal, Canada, July 1984.

[TS06] B. Triggs and M. Sdika. Boundary conditions for Young - van Vliet recursive filtering. *IEEE Transactions on Signal Processing*, 54(5):1–2, May 2006.

[Tyr96] E.E. Tyrtyshnikov. Mosaic-skeleton approximation. *Calcolo*, 33:47–57, 1996.

[Uli87] R. A. Ulichney. *Digital Halftoning*. Massachusetts Institute of Technology, Cambridge, MA, 1987.

[Uli88] R. A. Ulichney. Dithering with blue noise. *Proceedings of the IEEE*, 76(1):56–79, January 1988.

[USS11] T. Umenhoffer, L. Szécsi, and L. Szirmay-Kalos. Hatching for motion picture production. *EUROGRAPHICS 2011*, 30(2):533–542, 2011.

[VBZ+10] L. Valgaerts, A. Bruhn, H. Zimmer, J. Weickert, C. Stoll, and C. Theobalt. Joint estimation of motion, structure and geometry from stereo sequences. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Computer Vision – Proc. 11th European Conference on Computer Vision*, volume 6314 of *Lecture Notes in Computer Science*, pages 568–581, Heraklion, Greece, 2010. Springer, Berlin, Germany.

[vdBV96] C. J. van den Branden Lambrecht and O. Verscheure. Perceptual quality measure using a spatio-temporal model of the human visual system. *Proceedings of the SPIE*, 2668:450–461, January 1996.

[vMAF+08] J.A. van Meel, A. Arnold, D. Frenkel, S.F. Portegies Zwart, and R.G. Belleman. Harvesting graphics power for MD simulations. *Molecular Simulation*, 34(3):259–266, 2008.

[VO08] D. Vanderhaeghe and V. Ostromoukhov. Polyomino-based digital halftoning. In Pedro Isaías, editor, *IADIS International Conference on Computer Graphics and Visualization 2008*, pages 11–18, July 2008.

[Vog95] C. R. Vogel. A multigrid method for total variation-based image denosing. *Computation and Control IV*, 20:323–331, 1995.

[vR09] D.U. von Rosenberg. *Methods for the numerical solution of partial differential equations*. Elsevier, 5th edition, 2009.

[WA85] A. B. Watson and A. J. Ahumadra, Jr. Model of human visual-motion sensing. *Journal of the Optical Society of America A*, 2(2):322–342, February 1985.

[WB02] J. Weickert and T. Brox. Diffusion and regularization of vector- and matrix-valued images. In M. Z. Nashed and O. Scherzer, editors, *Inverse Problems, Image Analysis, and Medical Imaging*, volume 313 of *Contemporary Mathematics*, pages 251–268. AMS, Providence, 2002.

[WE98] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. *Computer Graphics (Proc. SIGGRAPH '98)*, 169–177, 1998.

[Wei98] J. Weickert. *Anisotropic Diffusion in Image Processing*. Teubner, Stuttgart, 1998.

[Wei10] L.-Y. Wei. Multi-class blue noise sampling. *ACM Transactions on Graphics (Proc. SIGGRAPH '10)*, 29(4):79(1–8), 2010.

[Wei11a] J. Weickert. Edge and coherence enhancing diffusion. Personal communication, 2011.

[Wei11b] J. Weickert. Fast Jacobi. Personal communication, 2011.

[Wei11c] J. Weickert. A novel discretisation for anisotropic diffusion. Personal communication, 2011.

[Wei11d] J. Weickert. Osmosis. Personal communication, 2011.

[Wel86] W. M. Wells. Efficient synthesis of Gaussian filters by cascaded uniform filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):234–239, March 1986.

[Wit83] A. P. Witkin. Scale-space filtering. In *Proc. Eighth International Joint Conference on Artificial Intelligence*, volume 2, pages 945–951, Karlsruhe, West Germany, August 1983.

[Wor08] S.P. Worley. Atomicexch(float) does not work - is it a bug? Online: `http://forums.nvidia.com/index.php?showtopic=79464&pid=451309&mode=threaded&start=#entry451309`, 2008. Retrieved 2011-04-12.

[WS01a] J. Weickert and C. Schnörr. A theoretical framework for convex regularizers in PDE-based computation of image motion. *International Journal of Computer Vision*, 45(3):245–264, December 2001.

[WS01b] J. Weickert and C. Schnörr. Variational optic flow computation with a spatio-temporal smoothness constraint. *Journal of Mathematical Imaging and Vision*, 14(3):245–255, May 2001.

[WSW08] M. Welk, G. Steidl, and J. Weickert. Locally analytic schemes: A link between diffusion filtering and wavelet shrinkage. *Applied and Computational Harmonic Analysis*, 24(195–224), 2008.

[WTP+09] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-$L^1$ optical flow. In *Proc. 20th British Machine Vision Conference*, London, UK, September 2009. British Machine Vision Association.

[XJM10] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. In *Proc. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1293–1300. IEEE Computer Society Press, 2010.

[YvV95] I.T. Young and L.J. van Vliet. Recursive implementation of the Gaussian filter. *Signal Processing*, 44:139–151, January 1995.

[ZBW+09] H. Zimmer, A. Bruhn, J. Weickert, L. Valgaerts, A. Salgado, B. Rosenhahn, and H.-P. Seidel. Complementary optic flow. In D. Cremers, Y. Boykov, A. Blake, and F. R. Schmidt, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, volume 5681 of *Lecture Notes in Computer Science*, pages 207–220. Springer, Berlin, 2009.

[ZBW11a] H. Zimmer, A. Bruhn, and J. Weickert. Freehand HDR imaging of moving scenes with simultaneous resolution enhancement. *Computer Graphics Forum (Proc. EUROGRAPHICS 2011)*, 30(2):405–414, April 2011.

[ZBW11b] H. Zimmer, A. Bruhn, and J. Weickert. Optic flow in harmony. *International Journal of Computer Vision*, 93(3):368–388, July 2011.

[ZF03] B. Zhou and X. Fang. Improving mid-tone quality of variable-coefficient error diffusion using threshold modulation. *ACM Transactions on Graphics*, 22(3):437–444, 2003.

[ZN10] X. Zhuge and K. Nakano. Halftoning via error diffusion using circular dot-overlap model. *International Journal of Digital Content Technology and its Applications*, 4(6):8–17, September 2010.

[ZPB07] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-$L^1$ optical flow. In F. A. Hamprecht, C. Schnörr, and B. Jähne, editors, *Pattern Recognition*, volume 4713 of *Lecture Notes in Computer Science*, pages 214–223. Springer, Berlin, 2007.