J Real-Time Image Proc (2014) 9:379–392
DOI 10.1007/s11554-011-0236-3

ORIGINAL RESEARCH PAPER

Fast electrostatic halftoning

Pascal Gwosdek · Christian Schmaltz · Joachim Weickert · Tanja Teuber

Received: 20 June 2011 / Accepted: 29 November 2011 / Published online: 24 December 2011

Abstract Electrostatic halftoning is a high-quality method for stippling, dithering, and sampling, but it suffers from a high runtime. This made the technique difficult to use for most real-world applications. A recently proposed minimisation scheme based on the *non-equispaced fast Fourier transform* (NFFT) lowers the complexity in the particle number M from $\mathcal{O}(M^2)$ to $\mathcal{O}(M \log M)$. However, the NFFT is hard to parallelise, and the runtime on modern CPUs lies still in the orders of an hour for about 50,000 particles, to a day for 1 million particles. Our contributions to remedy this problem are threefold: we design the first GPU-based NFFT algorithm without special structural assumptions on the positions of nodes, we introduce a novel nearest-neighbour identification scheme for continuous point distributions, and we optimise the whole algorithm for n-body problems such as electrostatic halftoning. For 1 million particles, this new algorithm runs 50 times faster than the most efficient technique on the CPU, and even yields a speedup of 7,000 over the original algorithm.

P. Gwosdek (✉) · C. Schmaltz · J. Weickert
Mathematical Image Analysis Group, Faculty of Mathematics and Computer Science, Saarland University, Campus E1.1, 66041 Saarbrücken, Germany
e-mail: gwosdek@mia.uni-saarland.de

C. Schmaltz
e-mail: schmaltz@mia.uni-saarland.de

J. Weickert
e-mail: weickert@mia.uni-saarland.de

T. Teuber
Mathematical Image Processing and Data Analysis, Department of Mathematics, Technical University of Kaiserslautern, Erwin-Schrödinger-Straße, 67663 Kaiserslautern, Germany
e-mail: tteuber@mathematik.uni-kl.de

Keywords NFFT · GPU · Fast Summation Algorithm · Sampling · Halftoning

1 Introduction

Digital image halftoning is the task of approximating images with continuous tones using a finite number of small discs. It is frequently used to binarise grey-scale images for printers or fax machines, or to create non-photorealistic renderings. Halftoning algorithms distribute points in such a way that, within each image part, the appearance of the binary image is close to that of the continuous original. These methods can be used for a variety of applications. Besides classic halftoning applications such as stippling [30], these algorithms even cover tasks from completely different contexts such as object placement [8], multi-class sampling [37], ray-tracing [26], image-based lighting using high dynamic range images [21], geometry processing [33], or numerical integration such as Quasi-Monte Carlo methods [7, 17]. A prominent numerical method for many of these applications was recently proposed by Balzer et al. [2], which suggested optimising the point locations iteratively by considering capacity-constrained Voronoi tessellations.

Schmaltz et al. [29] presented a new method called *electrostatic halftoning*, which achieves state-of-the-art results. It sets up on the simple idea to understand circular dots as small charged particles which repulse each other, but which are at the same time attracted by the image they are supposed to approximate. In the arising particle system, the solution is found in an iterative process. Each iteration consists of an evaluation of the interactions between all pairs of particles and the image, and a movement of particles based on the computed forces. As it turns out, this procedure is simple to implement, but very expensive to compute. A straightforward implementation possesses a runtime complexity of $\mathcal{O}(M^2)$ in the number of particles M. Hence, even on a modern CPU, a system with 1 million particles requires almost 2.5 h for one iteration. This article introduces a new parallel algorithm for GPUs, which has runtime complexity $\mathcal{O}(M \log M)$, and which thus reduces this runtime to about 1 s per iteration.

Particle systems based on the Coulomb potential possess a high influence in the near surrounding, but a low effect over large distances. This means that particles which are close together repulse each other significantly, while the force drops off rapidly the further particles are apart. A number of algorithms exploit this fact by approximating far-field interactions by a number of simplifications. This results in a lower runtime complexity, typically $\mathcal{O}(M \log M)$.

The first algorithm of this type was the particle–particle/particle–mesh (P^3M) method for molecular dynamics introduced by Hockney and Eastwood [18]. It performs a subdivision of the domain into cells, and assigns the charge density of each cell to a virtual super-particle in its centre. The interaction between these super-particles can then be efficiently computed by a fast Fourier transform (FFT). A more simple yet efficient method was introduced by Barnes and Hut [3] based on an idea of Appel [1]. It decomposes the image into a quadtree, and uses the accumulated charges in each subregion to estimate the influence of all particles in the region. Although this method has only an accuracy up to about 99%, it is still frequently used in astrophysical simulations due to its simplicity.

Over the last two decades, a new class of algorithms became popular. These algorithms also possess a runtime complexity of $\mathcal{O}(M \log M)$ or even $\mathcal{O}(M)$ for potentials with special characteristics. However, different to the aforementioned methods, these techniques approximate the exact solution up to a predetermined, arbitrarily small error. As shown by Fenn and Steidl [10], these methods are closely related to each other, although they target different applications. One prominent member of this class is the *fast multipole method* [13] which uses multipole expansions to compute interactions between many particles over large distances. This operation can be interpreted as a matrix-vector multiplication, where the advantageous runtime of the method follows from a sparsification of the huge operator [32]. Since this technique is data-parallel, it can be implemented for a variety of massively parallel architectures, such as the Intel Paragon [22], or GPUs [15]. In the literature, one finds algorithms based on similar ideas under the names *fast mosaic-skeleton approximations* [35] and *fast $\mathcal{H}$-matrix multiplications* [16].

Besides these, there are a number of related methods for the fast computation of matrix-vector products that differ from the aforementioned class only by details. Beylkin et al. [5] proposed to compute the interactions in the wavelet domain. This even allows to use kernels whose analytical properties are unknown. Beylkin and Cramer [4] suggested an algorithm which uses an approximation step followed by a correction step. While the approximation can be computed efficiently by fast Fourier transforms, the correction takes only place in a local neighbourhood around a particle. This strategy makes this approach very efficient.

For this paper, we use a related technique that exploits the convolution theorem to efficiently compute interactions between particles [11, 15]. Instead of evaluating the radial potential function for all pairwise interactions, it transfers the problem into the frequency domain where this expensive convolution reduces to a simple point-wise multiplication. An efficient way to handle points that do not reside on a regular grid is given by non-equispaced fast Fourier transforms (NFFTs) (see Dutt and Rokhlin [9], Potts et al. [27]). Recently, Teuber et al. [34] showed that this method can be used to significantly accelerate electrostatic halftoning on the CPU. However, a parallelisation of this technique for GPUs is not straightforward and involves many algorithmically extensive and time-critical operations.

In the following, we propose a novel parallel GPU algorithm based on the sequential algorithm from Teuber et al. [34]. Our work addresses several bottlenecks and shortcomings that make the original algorithm inefficient on parallel hardware, and introduces new concepts to handle these issues. The contributions are the following:

1. We design a new parallel NFFT algorithm that does not suffer from the limitations present in related parallel NFFT schemes from the literature [14, 31], and which is thus better suited for applications like electrostatic halftoning.
2. We introduce a novel nearest-neighbour identification scheme for continuously placed particles that allows highly efficient near-field computations on parallel devices such as GPUs. Unlike previous methods such as the one by Gumerov and Duraiswami [15], our method does not require to sort the particle vector after every iteration.
3. By a careful manual optimisation of the whole system, we obtain a speedup of about 50 over the CPU-based method from Teuber et al. [34], and a speedup of more than 7,000 over the algorithm from Schmaltz et al. [29] when using 1 million particles. This performance is remarkable considering that this algorithm is highly memory-bound.

While there are a number of other GPU-based algorithms for halftoning and importance sampling in the literature, all of them differ quite significantly from our work.

In the context of importance sampling for rendering, the algorithm presented by Nguyen [23, Ch. 20] uses the fact that the underlying density function is constant and that is possesses characteristic mathematical properties. However, this is not the case for halftoning where the underlying image may be completely random. The greedy algorithm proposed by Chang et al. [6] performs importance sampling on arbitrary density functions, but does not preserve so-called blue noise properties (see Ulichney [36]). As a consequence, the resulting samples reveal striking regularity artefacts. Most closely related to our work is the direct summation approach for electrostatic halftoning by Schmaltz et al. [29]. As mentioned before, however, this algorithm suffers from inadmissible runtimes for large images.

Our article is organised as follows: in Sect. 2, 3, and 4, we start with a short recapitulation of electrostatic halftoning [29], the fast summation technique introduced in Teuber et al. [34], and the NFFT [27], respectively. Section 5 gives details about our GPU-based implementation of NFFT-based electrostatic halftoning, which we evaluate in Sect. 6. The article is concluded with a summary in Sect. 7.

2 Electrostatic halftoning

A good halftoning method distributes points homogeneously over flat image regions. The key idea behind electrostatic halftoning is thus to maximise distances between inkblots by means of electrostatic forces [29]. In this model, inkblots take the role of small, massless particles with negative unit charge. Since their charges have the same sign, they repel each other such that the distances between particles are mutually maximised. In contrast, the underlying image is regarded as a positive charge density which attracts particles proportionally to the image darkness at the respective point. As a result, the converged state of the particle system forms a good halftone of the original image.

Given a grid $\Gamma = \{0,\ldots,n_x-1\} \times \{0,\ldots,n_y-1\}$ and a grey-valued input image $u : \Gamma \to [0,1]$, we thus search for a finite set of points $\{\mathbf{p}_\alpha\}$ with $\alpha \in \{1,\ldots,M\} =: \mathcal{P}$ that best approximates the density described by u. As it was shown by Teuber et al. [34], this problem can be formulated in terms of an energy which has to be minimised by the sought solution. In Schmaltz et al. [29], the authors propose a halftoning scheme which minimises such an energy by an iterative approach driven by the forces acting on a particle at a certain time step. By an abstraction from underlying physical properties such as velocity and acceleration, this simplified minimisation strategy finds a steady state of the particle system in which the forces are in an equilibrium. It does so by transporting particles a small time step along the vector of force acting on them. This yields the update equation

$$\begin{aligned}\mathbf{p}_\alpha^{k+1} &= \mathbf{p}_\alpha^k + \tau\left(\sum_{\substack{\mathbf{x}\in\Gamma\\ \mathbf{x}\neq\mathbf{p}_\alpha}} \frac{1-u(\mathbf{x})}{|\mathbf{x}-\mathbf{p}_\alpha|}\mathbf{e}_{\alpha,\mathbf{x}} - \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{1}{|\mathbf{p}_\beta-\mathbf{p}_\alpha|}\mathbf{e}_{\alpha,\beta}\right)\\ &= \mathbf{p}_\alpha^k + \tau\left(\mathbf{F}_\alpha^{(A)} - \mathbf{F}_\alpha^{(R)}\right), \qquad (1)\end{aligned}$$

where $\mathbf{p}_\alpha^k$ denotes the location of particle α at time level k, and τ represents a small time step which is typically chosen as $\tau = 0.1$. Moreover, $\mathbf{e}_{\alpha,\beta}$ and $\mathbf{e}_{\alpha,\mathbf{x}}$ are the unit vectors from $\mathbf{p}_\alpha$ to $\mathbf{p}_\beta$ and from $\mathbf{p}_\alpha$ to $\mathbf{x}$, respectively:

$$\mathbf{e}_{\alpha,\beta} := \frac{\mathbf{p}_\beta-\mathbf{p}_\alpha}{|\mathbf{p}_\beta-\mathbf{p}_\alpha|}\ , \quad \mathbf{e}_{\alpha,\mathbf{x}} := \frac{\mathbf{x}-\mathbf{p}_\alpha}{|\mathbf{x}-\mathbf{p}_\alpha|}. \qquad (2)$$

The minuend $\mathbf{F}_\alpha^{(A)}$ in (1) describes the attractive forces originating from the discrete grid points, and the subtrahend $\mathbf{F}_\alpha^{(R)}$ denotes the repulsive forces between particles. For more details about this approach, we refer to Schmaltz et al. [29].

3 Fast summation

In Schmaltz et al. [29], the authors evaluate the sum in the subtrahend of (1) by a direct summation approach. This means that, for every particle $\mathbf{p}_\alpha$, the arising repulsive force is accumulated from the interactions with all other particles $\mathbf{p}_\beta \neq \mathbf{p}_\alpha$. Although this algorithm is simple and easy to parallelise, it has a major drawback: its runtime scales quadratically in the number of particles. This makes it infeasible for large images containing many particles, even on modern massively parallel hardware such as GPUs.

As one solution to this problem, Teuber et al. [34] proposed to compute this sum by means of a fast summation technique [34]. By this, the computational complexity in the number of particles M drops from $\mathcal{O}(M^2)$ to $\mathcal{O}(M\log(M))$. However, if we compare the actual runtime of this CPU implementation with the GPU algorithm for the direct summation technique, we see that the better runtime class only pays off if the number of particles is large; see Fig. 6. In this article, we thus develop an efficient parallel algorithm for fast summation-based halftoning on graphics cards, and evaluate its performance against the existing approaches.

Before we go into detail about the parallel design of this algorithm, let us briefly sketch its idea. In Teuber et al. [34], the authors decompose the second sum of (1) into three sums:

$$
\begin{aligned}
&- \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{1}{\mathbf{p}_\beta-\mathbf{p}_\alpha}\mathrm{e}_{\alpha,\beta}\\
&= - \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{\mathbf{p}_\beta-\mathbf{p}_\alpha}{\left|\mathbf{p}_\beta-\mathbf{p}_\alpha\right|^2}\\
&= \mathbf{p}_\alpha \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{1}{\left|\mathbf{p}_\beta-\mathbf{p}_\alpha\right|^2} - \begin{pmatrix}1\\0\end{pmatrix} \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{\mathbf{p}_{\beta,x}}{\left|\mathbf{p}_\beta-\mathbf{p}_\alpha\right|^2}\\
&\quad - \begin{pmatrix}0\\1\end{pmatrix} \sum_{\substack{\beta\in\mathcal{P}\\ \mathbf{p}_\beta\neq\mathbf{p}_\alpha}} \frac{\mathbf{p}_{\beta,y}}{\left|\mathbf{p}_\beta-\mathbf{p}_\alpha\right|^2}.
\end{aligned}
\tag{3}
$$

In this context, the indices x and y refer to the first and second entry of a vector, and the vectors $(1,0)^\top$ and $(0,1)^\top$ are used to process components of a vector separately:

$$
\mathbf{p}_\beta = \begin{pmatrix}\mathbf{p}_{\beta,x}\\ \mathbf{p}_{\beta,y}\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}\mathbf{p}_{\beta,x} + \begin{pmatrix}0\\1\end{pmatrix}\mathbf{p}_{\beta,y}. \tag{4}
$$

Each of these sums can be computed by a convolution of a signal $\gamma(\beta)$ with a radial kernel $K : \mathbb{R}^+ \to \mathbb{R}$ defined by

$$
K(x) = \frac{1}{x^2}, \tag{5}
$$

where $x = \left|\mathbf{p}_\beta - \mathbf{p}_\alpha\right|$. The vectors $\gamma(\beta)$ take the instances

$$
\gamma(\beta) \equiv \mathbf{1}, \quad \gamma(\beta) = \mathbf{p}_{\beta,x}, \quad \text{and } \gamma(\beta) = \mathbf{p}_{\beta,y} \tag{6}
$$

for the three sums from (3), respectively.

By the convolution theorem, each of these convolutions can be written as a multiplication in the frequency domain. To this end, we shift and scale the image domain into the circle with radius $\frac{1-\varepsilon_B}{4}$ around (0, 0), and regularise K near 0 and $\pm\frac{1}{2}$ in each dimension using the small positive constants $\varepsilon_I, \varepsilon_B \ll 1$. This yields a smooth 1-periodic kernel

$$
\mathcal{K}_R(x) = \begin{cases} K_I(|x|), & |x| < \varepsilon_I\\ K(|x|), & \varepsilon_I \leqslant |x| < \frac{1-\varepsilon_B}{2}\\ K_B(|x|), & \frac{1-\varepsilon_B}{2} \leqslant |x| < \frac{1}{2}\\ K_B(\frac{1}{2}), & \frac{1}{2} \leqslant |x| \end{cases}. \tag{7}
$$

As suggested in Fenn and Steidl [11] and Teuber et al. [34], we use a two-point Taylor interpolation with polynomials of degree $\hat{p}$ to obtain K_I and K_B, where larger $\hat{p}$ lead to smaller approximation errors. Moreover, the kernel $\mathcal{K}_R$ is approximated by its truncated Fourier series

$$
\mathcal{K}_F(x) = \sum_{\mathbf{j}\in J_N} b_{\mathbf{j}} \mathrm{e}^{2\pi i\langle \mathbf{j},\mathbf{x}\rangle}, \tag{8}
$$

$$
b_{\mathbf{j}} = \frac{1}{N^2} \sum_{\mathbf{k}\in J_N} \mathcal{K}_R\left(\frac{k}{N}\right) \mathrm{e}^{-2\pi i \frac{\langle \mathbf{j},\mathbf{k}\rangle}{N}}, \tag{9}
$$

where $J_N = \{-\frac{N}{2}, \ldots, \frac{N}{2}-1\}^2$ with N even. If N is sufficiently large, it holds that $\mathcal{K}_F \approx \mathcal{K}_R$. Thus, by defining a near-field kernel

$$
\mathcal{K}_N = K - \mathcal{K}_R, \tag{10}
$$

we finally obtain

$$
K \approx \mathcal{K}_F + \mathcal{K}_N. \tag{11}
$$

For $\varepsilon_I < |x| \leqslant \frac{1-\varepsilon_B}{2}$, the near-field kernel $\mathcal{K}_N(x)$ vanishes, because $K = \mathcal{K}_R \approx \mathcal{K}_F$. Moreover, because $|\mathbf{p}_\beta - \mathbf{p}_\alpha| < \frac{1-\varepsilon_B}{2}$, $\mathcal{K}_N$ describes a purely local interaction of particles within a small neighbourhood. Consequently, the algorithm consists of two steps for each of the three sums in (3):

1. *Far-field interactions* First, we evaluate $\mathcal{K}_F$ in the frequency domain, i.e. compute three sums of type

$$
\begin{aligned}
&\sum_{\beta=1}^{M} \gamma(\beta)\mathcal{K}_F(\mathbf{p}_\beta - \mathbf{p}_\alpha)\\
&= \sum_{\mathbf{j}\in J_N} b_{\mathbf{j}} \left(\sum_{\beta=1}^{M} \gamma(\beta)\mathrm{e}^{2\pi i\langle \mathbf{j},\mathbf{p}_\beta\rangle}\right) \mathrm{e}^{-2\pi i\langle \mathbf{j},\mathbf{p}_\alpha\rangle}.
\end{aligned}
\tag{12}
$$

   The challenge in this part is that both $\mathbf{p}_\alpha$ and $\mathbf{p}_\beta$ are not residing on a regular grid. This prevents us from using a standard fast Fourier transform (FFT) to evaluate the two sums. As proposed in the literature, we solve this problem by means of a non-equispaced fast Fourier transform (NFFT) which approximates the Fourier transform of a randomly sampled signal [14, 34]. Such algorithms are often highly efficient on sequential hardware, but hard to parallelise for massively parallel architectures such as GPUs. As it turns out, approaches from the literature such as Sørensen et al. [31] or Gregerson [14] are not applicable for electrostatic halftoning. They require particles to be aligned in a particular order, which cannot be guaranteed for this class of algorithms. Other efficient GPU algorithms such as the ones by Gumerov and Duraiswami [15] can also not be used for electrostatic halftoning, because they require the vector of particle locations to be sorted. Since particles move, the vector must be re-sorted in every iteration, which infers an additional operation with complexity $\mathcal{O}(M \log M)$. In the following section, we present an approach that does not suffer from these problems.
2. *Near-field interactions* The second part is the evaluation of the near-field kernel $\mathcal{K}_N$, which comes down to a direct summation over a small number of neighbours within a circle of radius ε_I. Albeit the actual evaluation step is straightforward and similar in spirit to the algorithm of Schmaltz et al. [29], the

challenge lies in the retrieval of the set of neighbours. We will discuss a massively parallel algorithm for this purpose in Sect. 5.2.

Finally, let us consider the first sum from (1), which we have neglected so far. In Schmaltz et al. [29], the authors suggest precomputing samples for $\mathbf{p}_\alpha$ at grid locations at the beginning of the program run. Intermediate values can then be obtained by bilinear interpolation which represents a computationally inexpensive texturing operation on the GPU. We follow a similar strategy, but precompute the force texture by a fast summation approach. The arising terms are again of type (3), but the sampling points corresponding to $\mathbf{p}_\alpha$ and $\mathbf{p}_\beta$ are now residing on a regular grid. We can thus apply a standard FFT, and use the same implementation that we also need in the course of the NFFT. Please see the next section for details.

To summarise this section, let us briefly sketch our algorithm. Each minimisation step of our particle system looks as follows:

1. Compute far-field contributions from (12):
   (a) Perform adjunct NFFT on all $\gamma(\mathbf{x})$ from (6).
   (b) Scale coefficients by b_j.
   (c) Use NFFT to transform scaled coefficients back.
2. Evaluate and add near-field contributions.
3. Retrieve and add attractive image forces.
4. Move particles according to the force acting on them.
5. If particles are moved off the image domain, project them back to the nearest image point.

For more details, we refer the reader to the work of Teuber et al. [34].

4 Non-equispaced fast Fourier transform

The core of our fast summation algorithm is an efficient GPU implementation of the 2-D NFFT of Potts et al. [27]. In matrix-vector notation, the NFFT of $\hat{\mathbf{f}} \in \mathbb{C}^N$ at nodes $(\mathbf{x}_j)_{1\leqslant j\leqslant M} \subset \Pi^2 := \left[-\frac{1}{2},\frac{1}{2}\right]^2$ yields $\mathbf{f} \in \mathbb{R}^M$ as

$$\mathbf{f} = \mathsf{A}\hat{\mathbf{f}}, \quad \mathsf{A} := \left(\mathrm{e}^{-2\pi i k \mathbf{x}_j}\right)_{j=1,\ldots,M;\ k\in I_N}. \tag{13}$$

(see Keiner et al. [20]). Its adjoint is given by

$$\hat{\mathbf{h}} = \overline{\mathsf{A}}^\top \mathbf{f}. \tag{14}$$

In this context, $I_N := \{(k_1,k_2) \in \mathbb{Z}^2 | -\frac{N}{2} \leqslant k_* < \frac{N}{2}\}$ describes the ‘shifted’ index set in the frequency space, $M = |\mathcal{P}|$ is again the number of nodes, and $\overline{\mathsf{A}}^\top$ denotes the conjugate transpose of A. For the current application, we use the same ‘resolution’ in time and frequency domain, such that $N \sim \sqrt{\hat{p}M}$, where $\hat{p}$ again refers to the degree of the Taylor polynomial from the previous section. Moreover, we choose $\varepsilon_I = \frac{\hat{p}}{n} \sim \frac{\hat{p}}{N}$, $0 < \varepsilon_B \ll 1$ in accordance with Teuber et al. [34] and Potts et al. [28].

The algorithm presented in Potts et al. [27] to compute (13) approximates A as

$$\mathsf{A} \approx \mathsf{BFD}, \tag{15}$$

with the complex-valued discrete 2-D Fourier transform

$$\mathsf{F} := \left(\frac{1}{n^2}\mathrm{e}^{-2\pi i k\ell/n}\right)_{k,\ell\,\in\, I_n}, \tag{16}$$

a real-valued sparse matrix

$$\mathsf{B} := \left(\widetilde{\varphi}\left(\mathbf{x}_j - \frac{1}{n}\ell\right)\right)_{j=1,\ldots,M;\ \ell\,\in\, I_{n,m}(\mathbf{x}_j)}, \tag{17}$$

and a real-valued ‘diagonal’ scaling matrix

$$\mathsf{D} := \bigotimes_{t=1}^{2}\left(O \,\middle|\, \mathrm{diag}\left(\frac{1}{c_{k_t}(\varphi)}\right)_{k_t\in\mathbb{Z}\,:\, -\frac{N}{2}\leqslant k_t < \frac{N}{2}} \,\middle|\, O\right)^\top. \tag{18}$$

Here, O denotes zero matrices of size $N \times (n-N)/2$, $\bigotimes$ denotes the tensor product, and $n = \alpha N$ is the size of the Fourier plane oversampled by a factor $2 \leq \alpha < 4$ such that there exists a p: $\alpha N = 2^p$. Furthermore, $\widetilde{\varphi}(\mathbf{x}) := \varphi(\mathbf{x_1})\varphi(\mathbf{x_2})$, where φ denotes the 1-periodic continuation of the Kaiser–Bessel window function [6] on a torus, and $I_{n,m}(\mathbf{x}_j) := \{\ell \in I_n : n\mathbf{x}_{j,t} - m \leqslant \ell_t \leqslant n\mathbf{x}_{j,t} + m, t \in \{1,2\}\}$ denotes a 2-D index set. Note that indexing $\widetilde{\varphi}$ by elements from $I_{n,m}$ is equivalent to truncating the kernel at $\pm\frac{m}{n}$ in both dimensions prior to periodisation. The choice of m thus comes down to a trade-off between accuracy and speed. Finally, c_k denotes the 2-D Fourier coefficients which are given by $c_k : = c_{k_1}c_{k_2}$ due to their separability, where c_{k_*} are the 1-D Fourier coefficients given by

$$c_{k_*}(\varphi) = \int_{\Pi} \varphi(v)\mathrm{e}^{2\pi i k_* v} dv \ (k_* \in \mathbb{Z}). \tag{19}$$

In this terminology, the adjoint NFFT $\overline{A}^\top$ is given by

$$\overline{\mathsf{A}}^\top \approx \mathsf{D}^\top \overline{\mathsf{F}}^\top \mathsf{B}^\top. \tag{20}$$

Please note that in the non-equispaced case, i.e. if the nodes $\mathbf{x}_j$ are not aligned on a regular grid, $\hat{\mathbf{h}} = \overline{\mathsf{A}}^\top \mathsf{A}\hat{\mathbf{f}} \neq \hat{\mathbf{f}}$. Still, $\hat{\mathbf{h}}$ is a good approximation of $\hat{\mathbf{f}}$, such that we can use the NFFT in the context of the fast summation method.

To this end, each NFFT step in the algorithm from Sect. 3 works as follows:

1. Apply D: Scale input by $\frac{1}{c_k(\varphi)}$, and call the result $\mathbf{s}$. Create a zero vector $\mathbf{v}$ of size n^2 and copy $\mathbf{s}$ into $\mathbf{v}$:

$$\mathbf{v}_{(j+(n-N)/2)n+(n-N)/2+i} = \mathbf{s}_{jN+i} \tag{21}$$

for $j \in \{0,\ldots,N-1\}$, and $i \in \{1,\ldots,N\}$.

2. Apply F: Call 2-D FFT on intermediate result $\mathbf{v}$ from Step 1. The result is called $\mathbf{g}$.
3. Apply B: Compute the jth entry of the output $\mathbf{f}$ as the sum

$$\mathbf{f}_j := \sum_{\ell \in I_{n,m}(\mathbf{x}_j)} \mathbf{g}_\ell \, \widetilde{\varphi}(\mathbf{x}_j - \frac{1}{n}\ell). \tag{22}$$

Note that Step 3 only involves few multiplications and additions per entry. It comes down to summing up weighted entries in a small square-shaped neighbourhood around $\mathbf{x}_j$. We obtain the algorithm for the adjoint NFFT by 'reversing' the operations from above:

1. Apply $\mathsf{B}^\top$: compute result $\mathbf{g}_\ell$ as the sum

$$\mathbf{g}_\ell := \sum_{j \in I^\top_{n,m}(\ell)} \mathbf{f}_j \, \widetilde{\varphi}(\mathbf{x}_j - \frac{1}{n}\ell). \tag{23}$$

2. Apply $\overline{\mathsf{F}}^\top$: call inverse 2-D FFT on result $\mathbf{g}_\ell$ from 1.
3. Apply $\mathsf{D}^\top$: scale input by $\frac{1}{c_k(\widetilde{\varphi})}$ and call the result $\mathbf{v}$. Extract output $\mathbf{s}$ of size N^2 from $\mathbf{v}$ by reversing (21).

Here, we use $I^\top_{n,m}(\ell) := \{j = 0,\ldots,M-1 : \ell_t - m \leqslant n\mathbf{x}_{j,t} \leqslant \ell_t + m, t \in \{1,2\}\}$. Similar to before, we can compute Step 1 very efficiently if we exploit the special structure of $\mathsf{B}^\top$. Instead of computing each $\mathbf{g}_\ell$, we subsequently fix one j, evaluate all $\ell \in I_{n,m}(\mathbf{x}_j)$, and *accumulate* the contributions of $\mathbf{f}_j \, \widetilde{\varphi}(\mathbf{x}_j - \frac{1}{n}\ell)$ into the respective blocks of $\mathbf{g}$. This operation saves many multiplications with zero. Note that blocks written for different j can overlap, such that this case must be explicitly handled by the algorithm. For more details on the different steps, we refer the reader to the work of Keiner et al. [20].

5 GPU implementation

In this section, we design an efficient parallel fast summation algorithm for electrostatic halftoning on the GPU. Because our algorithm is designed for modern graphics cards produced by NVidia, we consequently use the CUDA framework. Nevertheless, our considerations mainly address general parallelisation aspects that can well be mapped to other frameworks such as OpenCL, BrookGPU, or Stream SDK. Our algorithm consists of a far-field computation and a near-field computation, both of which jointly replace the basic direct-summation algorithm from Schmaltz et al. [29]. Since the optional extensions proposed in this paper do not need to be adapted to our new algorithm, we consequently do not go into detail about them, but refer to the original work. In this paper, the authors also introduced a 'shaking' procedure to avoid local minima. This works by slightly moving particles into random directions. Due to a lack of efficient random number generators on GPUs, the authors performed this shaking step on the CPU. With the new `cuRand` library provided by NVidia, this operation can now be efficiently realised on the GPU without the need to transfer data from and to the device. This is very important for our fast summation algorithm, since it computes much larger sets of points than direct summation in the same amount of time. Data transfers via the PCI bus would unnecessarily slow down the overall algorithm.

Following the design principles of graphics card programming, our algorithm consists of comparably small data-parallel GPU programs, so-called kernels. Each kernel computes one data-parallel operation, such as the execution of the operators B or D for the NFFT. Our kernels are executed on graphics memory, but are dispatched and controlled by the CPU which takes care of the program flow. In addition to this, the CPU also handles memory copy operations to and from the graphics card at the beginning and ending of the program run. Once a kernel is invoked, it first retrieves a bunch of data from this GPU RAM to its fast on-chip memory, uses these data to compute a result, and stores the results back to GPU memory. Since large bunches of memory can be read faster than scattered values, memory access patterns play an important role for the performance of our algorithm. We address this issue later in this section. A second important performance criterion is the degree of data parallelism of each operator. In order to achieve a good performance, blocks of 32 variables in a row must be processed with exactly the same operations at the same time. If partitions of these blocks are processed differently, all conditional parts are processed sequentially, while the unaffected partitions are idle. By rewriting affected operations, we avoid this so-called warp divergence wherever this is possible. For example, we pad all vectors such that their length is a multiple of the width of a CUDA block. This allows to process them with all available threads, but does not destroy the data integrity: during data read, CUDA textures are simply bound to the actual size of the vector, such that surplus values are efficiently occluded.

Our framework consists of 18 custom kernels, plus calls to 16 CUDA library functions for the Fourier transform, memory operations, and the CUDA randomiser. Among our individual kernels are 8 which are exclusively called for the initialisation, and 10 which are also executed during each iteration. This accounts for the different data layouts and dependencies that are involved in our process, such as the linear storage of point locations, the 2-D representation in the Fourier space, or the lookup map for the nearest-neighbour identification scheme. As a side effect of this design, both the NFFT and its adjoint are encapsulated as static pipelines of CUDA kernels, such that they can easily be applied for applications other than fast summation approaches.

5.1 Far-field interaction

Let us first consider the far-field interactions. Here, some operations do not need to be complex-valued in the case of fast summation over real values, as can be seen from (8) and (12). The locations of the particles as well as the computed forces are both real-valued. As the first specialisation of the process, we thus use complex-to-real valued NFFTs and their real-to-complex valued adjoint operations which require significantly less runtime than the canonical complex-to-complex valued versions. Moreover, we should recall that the matrices B and D are both sparse such that the number of effective operations is much smaller than their size suggests. Moreover, note that neither of the matrices B, F, and D are stored. Instead, we compute their effect on the fly, as it is detailed in the following paragraphs.

Let us now detail on the single operators of standard and adjoint NFFTs. For the central operation, the FFT of length n described by F, we apply version 4.0 of the `cuFFT` library provided by NVidia [24]. This library still has a bad reputation for being slower than the alternative implementations such as the approach of Govindaraju et al. [12]. However, we cannot confirm this statement with our modern version of `cuFFT` which might be due to the recent improvements in this library [24].

Let us now have a look at the matrices D and $\mathsf{D}^\top$. Their diagonal sub-matrices of size $N \times N$ can be computed in parallel with each thread handling one diagonal entry. Unlike suggested by Potts et al. [27], we do not precompute $c_k(\widetilde{\varphi})$, as the GPU kernel is already strongly memory-bound and further (random) lookups lead to severe memory bottlenecks. In our experiments, evaluating $c_k(\widetilde{\varphi})$ on-the-fly turned out to be much less expensive. In particular, since c_k is a product of two 1-D functions, each thread reuses its value from on-chip memory by applying D or $\mathsf{D}^\top$ to four entries of the vector in parallel. This design also helps to hide memory latencies behind computations, and it accelerates the process significantly.

Finally, let us focus on B and $\mathsf{B}^\top$. Both require the computation of $\widetilde{\varphi}$, which comes down to the evaluation of Chebychev polynomials. Like before, we compute the values for $\widetilde{\varphi}$ on-the-fly. This is particularly interesting since the few coefficients for the Chebychev polynomials are cached and can thus be read without additional latencies. Still, the application of B and $\mathsf{B}^\top$ is the most expensive part of the NFFT (see Fig. 8), which is due to the arising memory patterns and the high data throughput. While the multiplication with B involves the load of patches of size $(2m + 1)^2$ from random positions in the input, the application of its adjoint requires to store patches to random locations. In the first case, we can partly reduce the expensiveness of this operation by reading data from textures. Whenever one cache miss is encountered, following reads in the neighbourhood are likely to be cached. However, the inverse operation which is required for the application of $\mathsf{B}^\top$ cannot be accelerated by similar ideas. As an additional challenge, it might even happen that different threads write to the same memory locations at the same time, which causes race conditions. Consequently, we use a CUDA atomic operation for the addition of single floating point numbers. This causes the application of $\mathsf{B}^\top$ to be slightly slower than the multiplication with B, but is still reasonably efficient since writing operations on modern graphics cards are buffered by linear L2 caches.

5.2 Near-field identification and interaction

After having computed the interaction of all particles with all other nodes that are sufficiently far away, we still have to compute the interaction with particles in their direct neighbourhoods. Due to the potential, forces between two particles that are close together are much higher than those between distant particles. As a consequence, these forces must be evaluated in a very accurate way, i.e. by direct summation. Although the involved number of computations is small due to the limited number of particles in a neighbourhood, the implementation still requires a careful algorithmic design: only a very limited number of candidates should be considered, and a k-d-tree as in Teuber et al. [34] is no option on GPUs because of the arising bad memory patterns and sequentialisation due to warp divergence.

As an alternative, we propose a new data-parallel approach which exploits the texturing unit of graphics

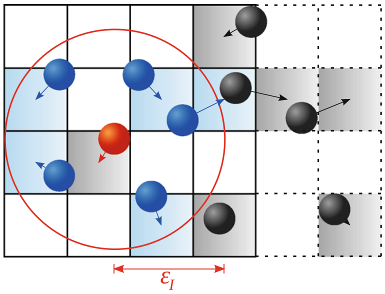

Fig. 1 Neighbourhood in a lookup map for the near-field of a particle (*red*): Neighbours (*blue*) and false positives (*black*). *Arrows* indicate mappings

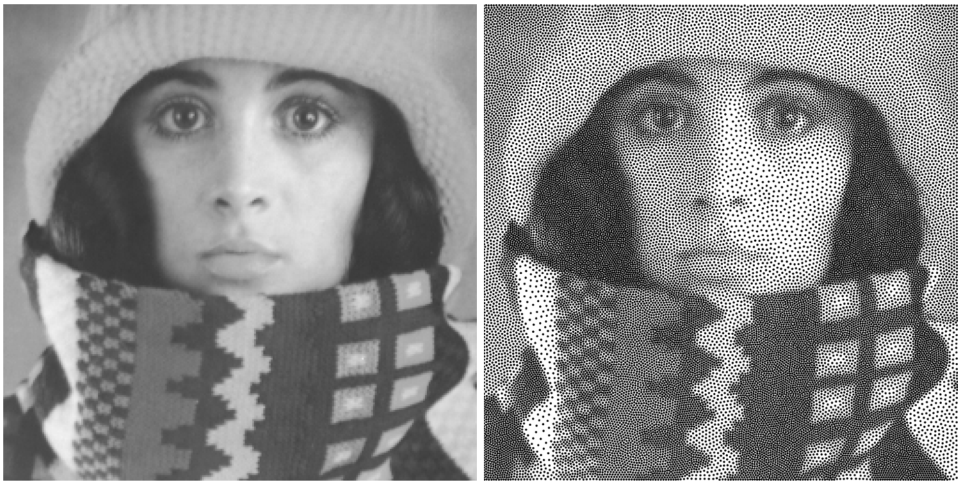

Fig. 2 Test image *Trui*, 256 × 256 pixels. *Left* Original. *Right* Stippling with 30,150 dots. This image serves as a basis for our quantitative evaluation. Runtime on GeForce GTX 480: 33.6 ms per iteration, or 6.72 s for 200 iterations

cards. It is motivated by the observation that electrostatic halftoning locally preserves the average grey value. The peak density of particles within a region of the result is thus the density corresponding to plain black. This allows to allocate a 2-D map that is large enough to absorb a completely black (i.e. saturated) image, and to assign each particle to the closest unoccupied cell in its neighbourhood. This cell will then contain the particle's exact position as a two-element vector. All unoccupied cells contain the empty pair. This is visualised in Fig. 1. In CUDA, we denote the empty pair by the bitstring $(1^{64})_2$, a vector of two (non-signalling) NaNs, which allows a fast initialisation by the use of `cudaMemset` and does not conflict with an actual particle residing at position $(0, 0)^\top$.

During the near-field evaluation, we can then resolve the rectangle occupied by the near field of a particle (the solid area in Fig. 1), perform a texture fetch at this point and, given a particle is inside, obtain its exact coordinates. We then check whether the exact distance to this particle is smaller than ε_I. If this is the case, we extend the sums from (3) by the respective near-field contributions. Otherwise, the given particle is a false positive and is thus ignored. Figure 1 visualises such mismatches by black dots. Please note that texture fetches in a neighbourhood benefit from the 2-D aware texture cache of graphics cards and can thus be performed very quickly. Moreover, we touch only a constant neighbourhood per particle and obtain very few false positives, such that this operation has linear runtime complexity in the number of particles and scales well over the cores provided by the graphics card.

Because the map changes in every iteration, the construction phase must not be too expensive as well. Moreover, we must pay particular attention to the case that two particles occupy the same cell (as in Fig. 1). For the parallel insertion process, we thus follow an idea inspired by cuckoo hashing [25]: a candidate is always placed in its designated cell, and pushes aside any potential particle that is already there. If the cell to the right is not empty, this process will be repeated until all particles found their place. In CUDA, we realise this strategy by atomic, i.e. thread-safe, exchange operations which we repeat with an incremented pointer until the empty pair is returned. This allows to insert many particles in parallel. Because there is no such operation for a `float2` vector type available, we use the `atomicExch` operation for `unsigned long long` types instead which works on equally long data chunks.

Since particles can be mapped outside the area covered by the near field, we extend the search window by a few optional positions to the right (the dashed cells in Fig. 1). Experimentally, our heuristic nearest-neighbour detection scheme works well even on very saturated images. Using the initialisation of the point locations as proposed in Schmaltz et al. [29], a bound of two pixels is sufficient to offer enough extra space for potentially offset particles. However, our algorithm is also robust against sub-optimal initialisations. In such cases, particles in oversaturated image regions might not be found during the first iterations of the algorithm. Nevertheless, the rough estimate obtained in this case suffices to let the algorithm gradually converge. Once the oversaturated regions disappear, our neighbour-identification scheme rehabilitates and yields the exact results.

6 Experiments

6.1 Examples

Let us first show some examples for the performance of our algorithm. Figure 2 shows a stippling of the test image *Trui*, 256 × 256 pixels, where the size of the dots has been chosen such that each dot covers the area of one pixel. This leads to a total of 30,150 particles. The resulting halftone has the same quality as the results from Schmaltz et al. [29]

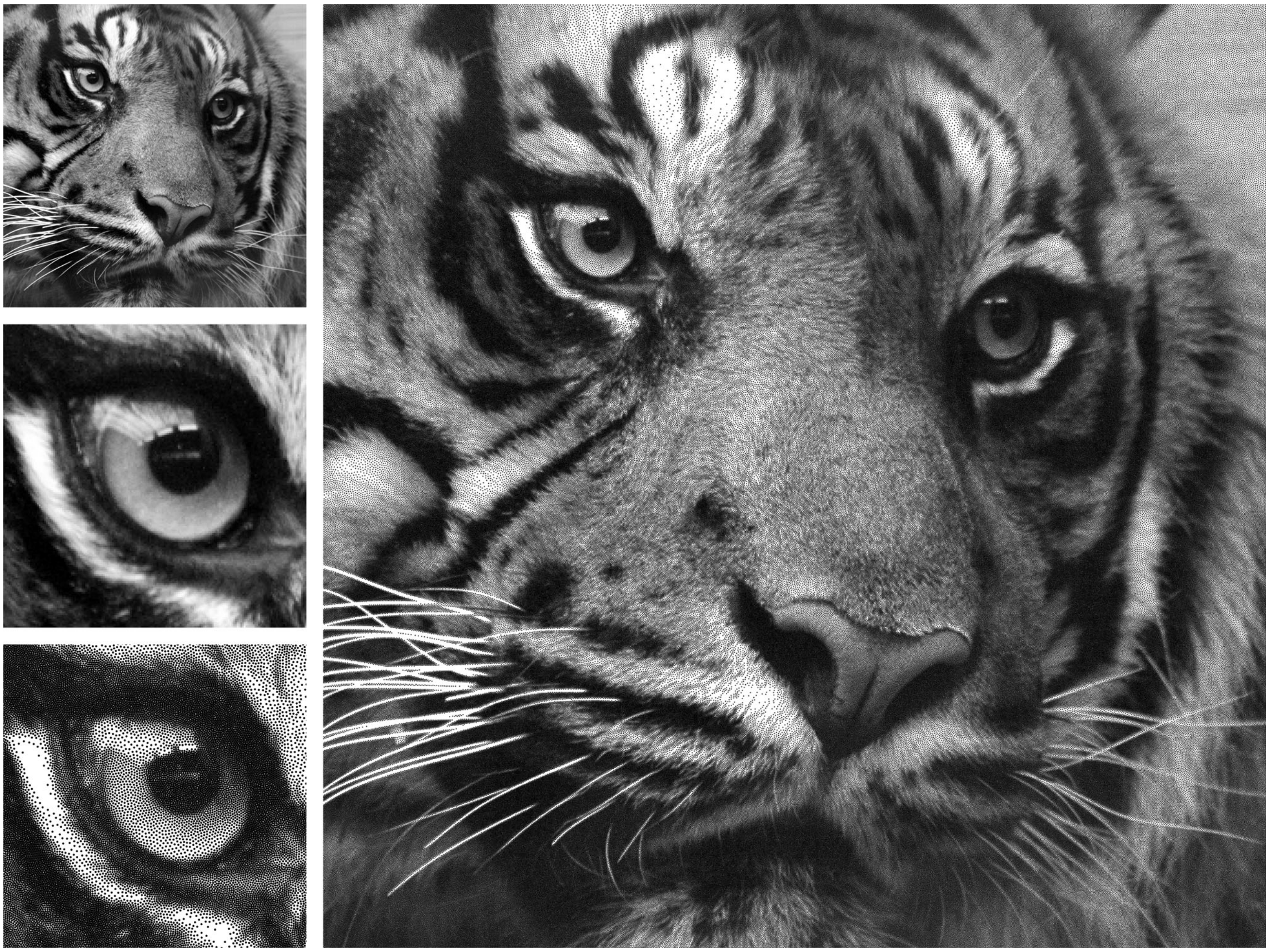

Fig. 3 *Right* Stippling with 647,770 dots on test image *Tiger*, 1,024 × 1,024 pixels (License: TeryKats, flickr.com, CC-BY). *Left, top to bottom* Original, zoom into original, zoom into result. Runtime on GeForce GTX 480: 820 ms per iteration, or 164 s for 200 iterations

and the ‘logarithmic’ version from Teuber et al. [34], and shows no perceptible artefacts compared to the original. Because this image is small enough that it can still be processed by the algorithm from Balzer et al. [2] in a reasonable time, we use it as a basis of evaluation in the next section.

As a second experiment, we computed a halftoning of the test image *Tiger* (1,024 × 1,024 pixels). This experiment on real-world data with high contrast, fine structures, and flat regions demonstrates the very good performance of our algorithm with respect to both approximation quality and runtime. Figure 3 shows the result with 647,770 dots. The halftoning result is not only very accurate and almost indistinguishable from the original but also computed in a very fast way. An NFFT-based fast summation algorithm on an NVidia GeForce GTX 480 computes the required $420 \cdot 10^9$ interactions per iteration in less than 820 ms. Given that 200 iterations already suffice to obtain a high quality, our algorithm takes less than 3 min to yield a result such as the one shown in Fig. 3.

A small shortcoming of fast summation approaches is their high memory consumption. The Fourier plane used for (16) must have a power-of-two side length (see Sect. 4 and Potts et al. [27] for details). Given a graphics card such as our GTX 480 with 1.5 GB of RAM, or a 32-bit CPU, this limits both the image size and the number of particles to $2^{20} \approx 1$ million, each. Although this problem vanishes if we consider 64-bit systems with sufficient physical memory, it still seems to represent a drawback compared to direct summation approaches. They require only one vector that contains the two `float` coordinates of each pixel. Thus, direct summation algorithms can in principle deal with sets containing more than 65 million particles. However, since each iteration requires about 5 days on the GPU, and more than a year on the CPU, this is infeasible in practise.

6.2 Quality

After these first impressions, we now evaluate the approximation quality of the proposed algorithm, and compare it against the direct summation approach from Schmaltz et al. [29], and against the capacity-constrained approach of Balzer et al. [2]. A comparison against the CPU algorithm from Teuber et al. [34] is not shown in this experiment, as these results are equivalent to the ones of our GPU algorithm.

While the near-field is always computed accurately, the approximation quality of the far-field can be controlled by two parameters, namely the degree of the polynomials $\hat{p}$ (see page 9) and the cutoff parameter m (see page 13). Without going into detail about the theoretical background of these two parameters, we regard both as one abstract trade-off parameter $\hat{m} = m = \hat{p}$. In the following, we evaluate the effect of this parameter on the similarity of the halftone to the original image, as well as on the spectral properties of the halftone. While the first measure tells us how well the grey value of a given image is approximated at any point in the image domain, the second criterion measures the freedom of artefacts.

To this end, we first blur both the original image and the halftoning result by Gaussian convolution with the same standard deviation σ. This mimics the human visual system under a certain viewing distance, and can be used to compute the visual similarity of both results to a human observer. As a measure for similarity between both images, we apply the *peak signal to noise ratio* (PSNR). Because σ depends crucially on the viewing distance and the resolution of the halftone, we preserve the generality of this experiment by computing the similarity for many standard deviations simultaneously. The higher the PSNR is, the better is the performance for a specific standard deviation. Figure 4 shows the results of this experiment. As expected, higher values of $\hat{m}$ yield better approximations of the original. Using $\hat{m} = 3$, our method already yields solutions that clearly surpass the ones of Balzer et al. [2]. With $\hat{m} \geqslant 4$, the results are almost indistinguishable from those obtained with the direct summation approach from Schmaltz et al. [29].

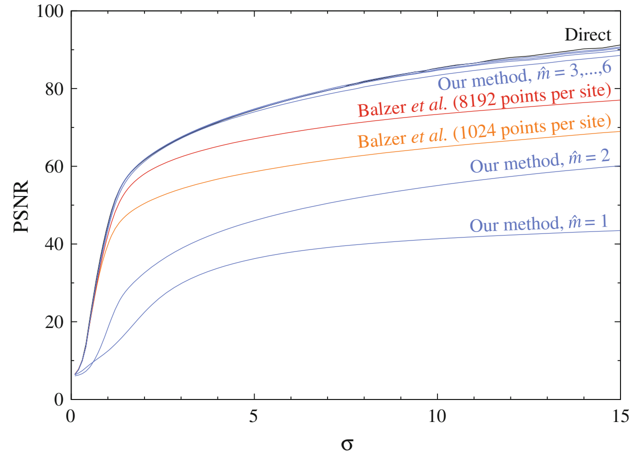

Fig. 4 Peak signal to noise ratio between blurred original and blurred halftone using different standard deviations σ

Secondly, let us measure the spectral properties of the results. Because this evaluation requires uniform images, we generate a halftone with the average grey value of 0.85, and analyse the regularity of the found point set in the frequency domain with respect to two criteria. Both are computed from the power spectrum of the result, as described by Ulichney [36]. While the *radially averaged power spectrum* (RAPS) is obtained from an averaging over concentric circles, the anisotropy results from the variance on these circles. Good halftones are supposed to possess *blue noise* properties, i.e. the RAPS should contain a single peak at the *principle frequency* f_g, it should vanish below f_g, the transition region between both regions should be steep, and the interval above f_g shall be flat. The expected principle frequency depends on the average grey value and on whether the underlying grid is rectangular or hexagonal. Since our method produces hexagonal structures on a rectangular grid, we expect f_g to lie between these extremes (see Ulichney [36]). Consequently, we depict both frequencies by vertical lines in the plot. Moreover, blue noise characteristics cause a flat and low anisotropy. Due to background noise, the theoretical limit for this measure lies at -10 dB. The closer a method approaches this limit, the better it is.

Figure 5 shows the results of this experiment for trade-off parameters $\hat{m} \in \{3, 4, 5\}$. In each graph, the result for

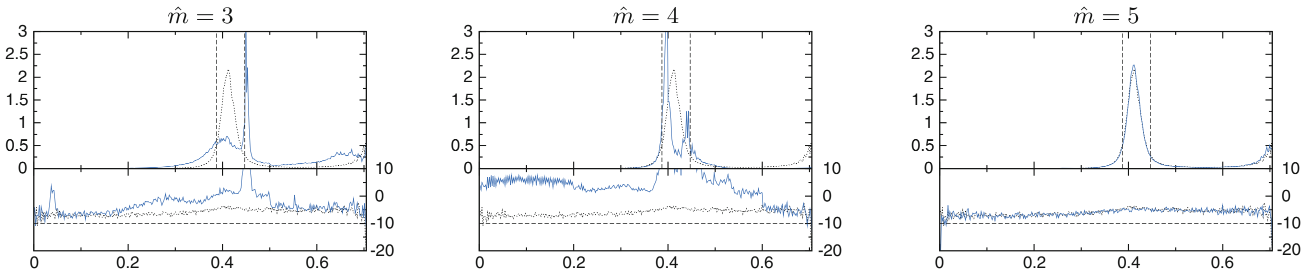

Fig. 5 Radially averaged power spectra (RAPS) (*top*) and anisotropy measures in decibels (*bottom*). *Blue* Proposed method using $\hat{m} = 3$, $\hat{m} = 4$, and $\hat{m} = 5$, respectively. *Black dotted* Reference measurement using the direct summation approach. The *two dashed lines in each graph* correspond to the two principal frequencies

Table 1 Runtime comparison for 1 iteration with different numbers of particles, and speedup factors

Particles	16,384			65,536			262,144			1,048,576		
	CPU	GPU	Speedup	CPU	GPU	Speedup	CPU	GPU	Speedup	CPU	GPU	Speedup
Direct	2.12	0.03	**70.67**	33.90	0.43	**78.84**	542.64	6.62	**81.97**	8,853.46	103.61	**85.45**
Fast	0.84	0.02	**41.12**	3.65	0.07	**56.00**	14.74	0.28	**53.58**	57.95	1.20	**48.20**
Speedup	**2.54**	**1.62**		**9.28**	**6.69**		**36.81**	**24.06**		**152.78**	**86.17**	
Total speedup			**104.38**			**519.85**			**1,972.48**			**7,363.50**

All times are given in seconds. Speedup factors describe the parallelisation benefit (horizontal), the difference between runtime classes (vertical), and the overall improvement of the fast summation GPU method over the original CPU direct summation technique (total)

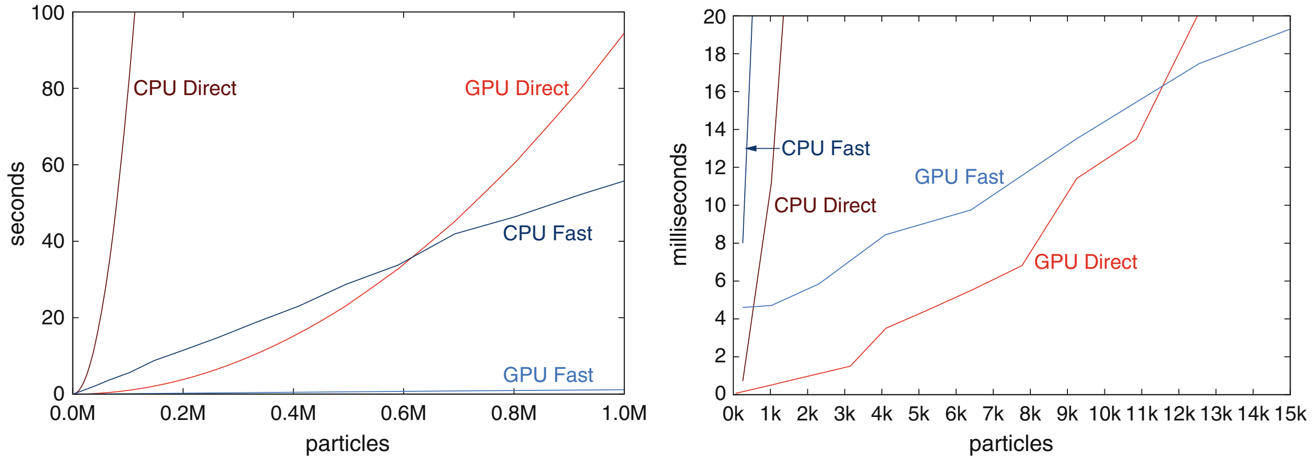

Fig. 6 *Left* Runtime per iteration using the direct summation approach and the NFFT-based fast summation approach on CPU and GPU using up to one million particles. *Right* Close-up into the lower left corner

the direct summation method from Schmaltz et al. [29] is superposed as a black dotted curve. Both for $\hat{m} = 3$ and $\hat{m} = 4$, we observe striking artefacts. In the case $\hat{m} = 3$, the method slightly adapts to a rectangular grid, which explains the RAPS peak at the dashed line. Surprisingly, even the case $\hat{m} = 4$ contains striking artefacts. Here, we obtain an almost regular hexagonal grid which explains the peaks in the RAPS and the very unsatisfying anisotropy measure. This behaviour changes drastically if we choose $\hat{m} \geqslant 5$. For these choices, we obtain a result that is equivalent to the one obtained by the direct summation approach. The insights obtained in this evaluation complement the previous experiment, and tell us to set $\hat{m} = 5$ if we require artefact-free results. Consequently, all further experiments in this article are conducted with this configuration.

6.3 Runtime

In this section, we evaluate the runtime of the proposed method on an NVidia GeForce GTX 480 graphics card, and compare it to the *direct summation* method on the GPU and the CPU (see Schmaltz et al. [29]), and to the *fast summation* method on the CPU (see Teuber et al. [34]). The CPU-based experiments were conducted on the same Intel Core 2 Duo E8200 CPU with 2.66 GHz, and could entirely be computed within physical memory.

Table 1 shows the runtime of a single iteration with 2^{14}, 2^{16}, 2^{18}, and 2^{20} particles, and the corresponding speedups obtained. Note that ‘speedup’ columns refer to the parallelisation gain, while the corresponding row refers to the numerical improvement by the NFFT algorithm on the respective architectures. The overall benefit obtained by both the algorithmic and numerical improvements is indicated in the bottom row.

Compared to the naive implementation on the CPU, our method obtains speedups of more than 7,000 for one million particles. This is both due to the efficient numerics, and due to our parallelisation efforts. While the NFFT algorithm already yields a factor of up to 150 over direct summation, the parallelisation yields another speedup of 50. The latter is impressive, considering that some operations such as the near-field evaluation or the execution of the operators B and $\mathsf{B}^{\top}$ are very time-consuming and hard to parallelise. However, the speedup of the proposed approach over the direct summation method on the GPU is only about half as high as it is for the corresponding

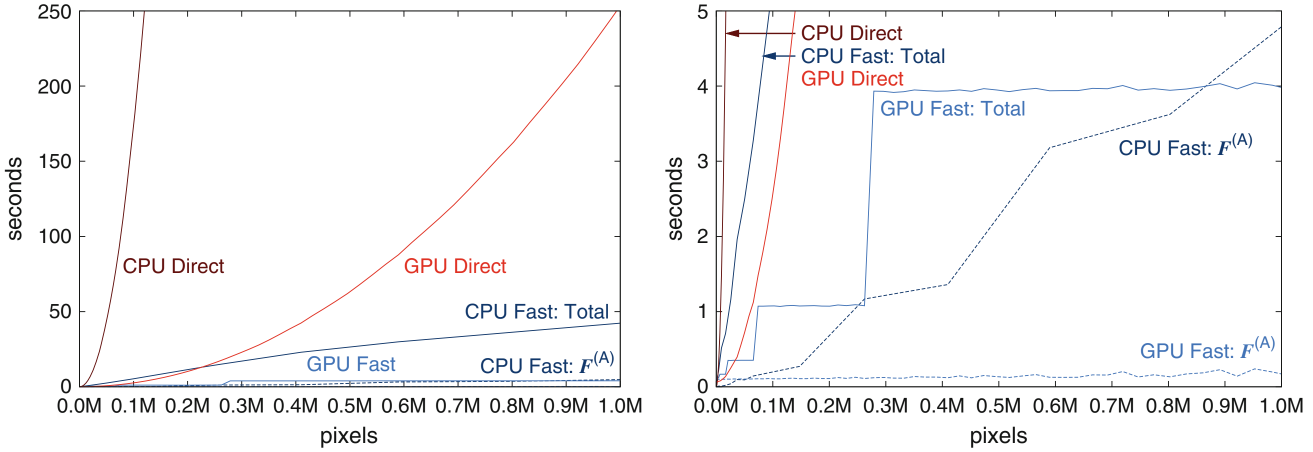

Fig. 7 *Left* Initialisation time using the direct summation approach and the NFFT-based fast summation approach on a CPU and on a GPU. *Right* Zoom in into the lower part of the left plot

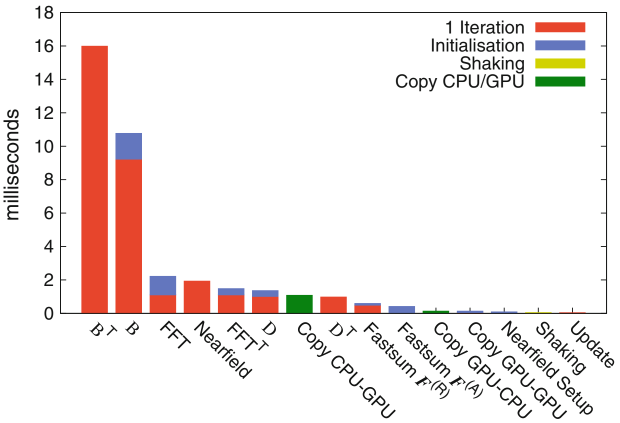

Fig. 8 Runtime required for different parts of our algorithm. Kernels are grouped according to the operation they perform. All graphs are normalised to the time required for one iteration (*red*), one initialisation (*blue*), one shaking step (*yellow*), and the corresponding memory copy operations (*green*)

problems on the CPU. This indicates that the process is memory-bounded.

Figure 6 shows the scaling behaviour of the analysed methods over a varying number of particles. We can easily see the different complexity classes which lead to a better asymptotic behaviour of the fast summation approach. On both the CPU and the GPU, this method clearly outperforms the direct summation method for large particle sets. However, the fast-summation method also possesses an overhead that results in a worse performance for very few particles. For the CPU-based method from Teuber et al. [34], this means that the direct summation approach on the GPU is still faster until the number of particles exceeds about 620,000. With our new parallelisation, this break-even point is drastically lowered to around 11,500 particles.

A second important property for real-world applications is the initialisation time of the process. For direct summation algorithms, it corresponds to the time required to precompute the attractive image forces $\mathbf{F}^{(\mathbf{A})}$. This number depends only on the number of pixels of the input image. In fast summation approaches, it is additionally necessary to compute the Fourier transform of the radial kernel in the frequency space. Moreover, the CPU-based fast summation method also precomputes samples for the function $\widetilde{\varphi}$. Our GPU algorithm computes these values on the fly (see Sect. 1 for details). In Figure 7, we thus depict total initialisation times with solid lines, and the time required to set up the attractive force field with dashed lines. The latter operation can be exchanged with a direct summation initialisation, which is again beneficial for small images.

The jumps visible in the total initialisation time for the fast summation algorithm occur whenever the image width or height reaches a power of two. This is because the image plane in the frequency space grows with the image, 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 the observed runtime behaviour. Since we used quadratic images to create the graphs shown in Fig. 7, jumps appear whenever the number of pixels exceeds a power of four.

6.4 Profiling

Finally, we detail on the time required for each individual operator of our algorithm on the graphics card. This gives insights about bottlenecks and shortcomings of our parallelisation approach. Using the CUDA profiler on a halftoning process for *Trui*, we measure the runtime over 100 iterations and 10 shaking operations, and normalise them to one iteration and one shaking step, each. Figure 8 shows the result of this experiment. Red denotes contributions that scale with the number of iterations, yellow depicts those which scale with the application of the shaking procedure, blue denotes one-off expenses such as initialisation, and green are one-off memory copy operations between CPU and GPU which are not required if the problem already resides on the GPU. The CUDA profiler reports 34 different kernels, out of which 18 are custom-built and 16 represent callbacks to CUDA libraries. Because many of these calls do not significantly affect the overall runtime, we grouped them into 15 meaningful units.

As can be seen in Fig. 8, executing the operators B and $\mathsf{B}^\top$ dominates the runtime of each iteration, and thus of the whole process. This is caused by the expensive convolution at random positions in the image domain, which leads to undesired memory patterns and a strong memory-boundedness of the algorithm. However, other operations with complex memory patterns, such as the computation of near-field interactions or the FFT, do not represent a bottleneck of the algorithm. Compared to the original approach from Schmaltz et al. [29], the time required for the shaking step almost vanishes in the overall runtime of the process. This is a consequence of the GPU-based perturbation of particles, and the resulting absence of additional memory copy operations.

7 Summary and conclusion

This article presents a highly efficient GPU implementation of the fast summation algorithm (see Teuber et al. [34]) for electrostatic halftoning. It introduces the first parallel algorithm of the non-equispaced Fourier transform (NFFT) on the GPU that does not assume special structural arrangements of nodes, and extends it by novel concepts such as a fast parallel nearest-neighbour retrieval for a continuous placement of points. Our sophisticated algorithm improves the runtime of the naive CPU algorithm for electrostatic halftoning by a factor of more than 7,000 without constraining its quality.

These results set new standards for the computation of state-of-the-art halftones with very large numbers of dots in a small runtime. While the overall runtime of several hours to days prevented electrostatic halftoning from being used in real-world applications, our new approach opens the doors for the application in interactive systems. Moreover, our algorithm enjoys a broad applicability beyond halftoning or sampling, as the NFFT nowadays represents a standard tool for many applications. We are confident that our contribution helps researchers in all of these areas to obtain highly qualitative results in a fraction of the usual runtime.

Acknowledgments The authors thank the Cluster of Excellence ‘Multimodal Computing and Interaction’ for partly funding this work, and Thomas Schlömer from the University of Constance for providing images for comparison.

References

1. Appel, A.W.: An efficient program for many-body simulation. SIAM J. Sci. Stat. Comput. **6**(1), 85–103 (1985)
2. Balzer, M., Schlömer, T., Deussen, O.: Capacity-constrained point distributions: A variant of Lloyd’s method. ACM Trans. Graph. **28**(3), 86–18 (2009)
3. Barnes, J., Hut, P.: A hierarchical $\mathcal{O}$ (n log n) force-calculation algorithm. Nature **324**, 446–449 (1986)
4. Beylkin, G., Cramer, R.: A multiresolution approach to regularization of singular operators and fast summation. SIAM J. Sci. Comput. **24**(1), 81–117 (2003)
5. Beylkin, G., Coifman, R., Rokhlin, V.: Fast wavelet transforms and numerical algorithms I. Commun. Pure Appl. Math. **44**(2), 141–183 (1991)
6. Chang, C.W., Lin, W.C., Ho, T.C., Huang, T.S., Chuang, J.H.: Real-time translucent rendering using GPU-based texture space importance sampling. EUROGRAPHICS **27**(2), 517–526 (2008)
7. Cook, R.L.: Stochastic sampling in computer graphics. ACM Trans. Graph. **5**(1), 51–72 (1986)
8. Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., Prusinkiewicz, P.: Realistic modeling and rendering of plant ecosystems. In: Proc. SIGGRAPH ’98, ACM, New York, pp. 275–286 (1998)
9. Dutt, A., Rokhlin, V.: Fast Fourier transforms for nonequispaced data. SIAM J. Sci. Comput. **14**(6), 1368–1393 (1993)
10. Fenn, M., Steidl, G.: FMM and $\mathcal{H}$-matrices: A short introduction to the basic idea. Tech. Rep. TR-2002-008, Department for Mathematics and Computer Science, University of Mannheim, Germany (2002)
11. Fenn, M., Steidl, G.: Fast NFFT based summation of radial functions. Sampl. Theory Signal Image Process. **3**(1), 1–28 (2004)
12. Govindaraju, NK., Lloyd, B., Dotsenko, Y., Smith, B., Manferdelli, J.: High performance discrete Fourier transforms on graphics processors. In: Proc. 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, pp. 2:1–2:12 (2008)
13. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
14. Gregerson, A.: Implementing fast MRI gridding on GPUs via CUDA. NVidia Whitepaper. Online: http://cn.nvidia.com/docs/IO/47905/ECE757_Project_Report_Gregerson.pdf. Retrieved 11 April 2011 (2008)
15. Gumerov, N.A., Duraiswami, R.: Fast multipole methods on graphics processors. J. Comput. Phys. **227**(18), 8290–8313 (2008)
16. Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. Computing **62**, 89–108 (1999)
17. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer. Math. **2**(1), 84–90 (1960)
18. Hockney, R., Eastwood, J.: Computer simulation using particles. McGraw-Hill, New York (1981)
19. Kaiser, J., Schafer, R.: On the use of the Io-sinh window for spectrum analysis. IEEE Trans. Acoust. Speech Signal Process. **28**(1), 105–107 (1980)
20. Keiner, J., Kunis, S., Potts, D.: Using NFFT 3—a software library for various nonequispaced fast Fourier transforms. ACM Trans. Math. Softw. **36**(4), 19–30 (2009)
21. Kollig, T., Keller, A.: Efficient illumination by high dynamic range images. In: Christensen P, Cohen-Or D (eds.) EGRW ’03:

Proceedings of the 14th Eurographics Workshop on Rendering, Eurographics Association, Aire-la-Ville, Switzerland, pp. 45–50 (2003)

22. Krishnan, S., Kalé, LV.: A parallel adaptive fast multipole algorithm for n-body problems. In: Gallivan K (ed.) Proc. 1995 International Conference on Parallel Processing, CRC Press, Inc., Urbana-Champain, IL, vol 3, pp. 46–51 (1995)
23. Nguyen, H.: (ed.) GPU Gems 3. Addison–Wesley Professional (2007)
24. NVidia.: NVIDIA CUDA CUFFT Library. NVIDIA Corporation, 3rd edn. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf. Retrieved 24 October 2010 (2010)
25. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**(2), 122–144 (2004)
26. Pharr, M., Humphreys, G.: Sampling and reconstruction, chap. 7. In: Physically Based Rendering—from Theory to Implementation. Morgan Kaufmann (2004)
27. Potts, D., Steidl, G., Tasche, M.: Fast Fourier transforms for nonequispaced data: A tutorial. In: Benedetto JJ, Ferreira PJSG (eds.) Modern Sampling Theory: Mathematics and Applications, Applied and Numerical Harmonic Analysis, chap 12, pp 251–274. Birkhäuser, Boston (2000)
28. Potts, D., Steidl, G., Nieslony, A.: Fast convolution with radial kernels at nonequispaced knots. Numer. Math. **98**(2), 329–351 (2004)
29. Schmaltz, C., Gwosdek, P., Bruhn, A., Weickert, J.: Electrostatic halftoning. Comput. Graph. Forum **29**(8), 2313–2327 (2010)
30. Secord A Weighted Voronoi stippling. In: Proceedings of the Second International Symposium on Non-photorealistic Animation and Rendering, ACM, New York, pp. 37–43 (2002)
31. Sørensen, T., Schaeffter, T., Noe, K., Hansen, M.: Accelerating the nonequispaced fast Fourier transform on commodity graphics hardware. IEEE Trans. Med. Imaging **27**(4), 538–547 (2008)
32. Sun, X., Pitsianis, N.P.: A matrix version of the fast multipole method. SIAM Rev. **43**(2), 289–300 (2001)
33. Surazhsky V, Alliez P, Gotsman C Isotropic remeshing of surfaces: A local parameterization approach. In: Proceedings of the 12th International Meshing Roundtable, pp. 215–224 (2003)
34. Teuber, T., Steidl, G., Gwosdek, P., Schmaltz, C., Weickert, J.: Dithering by differences of convex functions. SIAM J. Imaging Sci. **4**(1), 79–108 (2011)
35. Tyrtyshnikov, E.: Mosaic-skeleton approximation. Calcolo **33**, 47–57 (1996)
36. Ulichney, R.A.: Dithering with blue noise. Proc. IEEE **76**(1), 56–79 (1988)
37. Wei, L.: Multi-class blue noise sampling. ACM Trans. Graph. **29**(4), 79–18 (2010)

Author Biographies

Pascal Gwosdek received a bachelor and a master degree in computer science from Saarland University in 2006 and 2008, respectively. He joined the Mathematical Image Analysis Group as a Ph.D. student in May 2008. His scientific interests include high performance algorithms and hardware-focussed programming in visual computing. Pascal Gwosdek received the Günter Hotz Medal for the best master's thesis in computer science at Saarland University. In his spare time he heads the Jugend-forscht-Club Saar, an organisation that promotes scientific activities of high school students.

Christian Schmaltz received a Diplom (Master) degree in computer science from Saarland University in 2004, and a master degree in applied mathematics in 2009. Since April 2006, he has been with the Mathematical Image Analysis Group, Department of Mathematics and Computer Science, Saarland University, where he is currently pursuing his Ph.D. degree in computer science. His current research interests include 3-D pose tracking, image and video compression, and halftoning.

Joachim Weickert is professor of mathematics and computer science at Saarland University where he heads the Mathematical Image Analysis Group. He received a diploma and a Ph.D. degree in mathematics from the University of Kaiserslautern (1991, 1996), and a habilitation degree in computer science from the University of Mannheim (2001). He worked as research assistant at the University of Kaiserslautern, as post-doctoral researcher at the universities of Utrecht and Copenhagen, and as assistant professor at the University of Mannheim. Joachim Weickert has developed many models and efficient algorithms for image processing and computer vision using partial differential equations and optimisation principles. In particular, he has contributed to diffusion filtering, optical flow computation, processing of tensor fields, and image compression. His scientific work covers about 240 refereed publications. He has served in the editorial boards of nine international journals and given more than 130 invited talks. In 2010 he has received the Gottfried Wilhelm Leibniz Prize which is the highest German science award.

Tanja Teuber studied mathematics and computer science at the University of Mannheim, Germany, and the University of Swansea, UK. She received a diploma in mathematics from the University of Mannheim in 2008 and is currently pursuing a Ph.D. at the University of Kaiserslautern. Her research interests include variational and PDE-based image processing methods for applications like denoising, deblurring and digital halftoning.