Laplacian Kernel Splatting for Efficient Depth-of-field and Motion Blur Synthesis or Reconstruction

THOMAS LEIMKÜHLER, MPI Informatik, Saarland Informatics Campus, Germany
HANS-PETER SEIDEL, MPI Informatik, Saarland Informatics Campus, Germany
TOBIAS RITSCHEL, University College London, United Kingdom

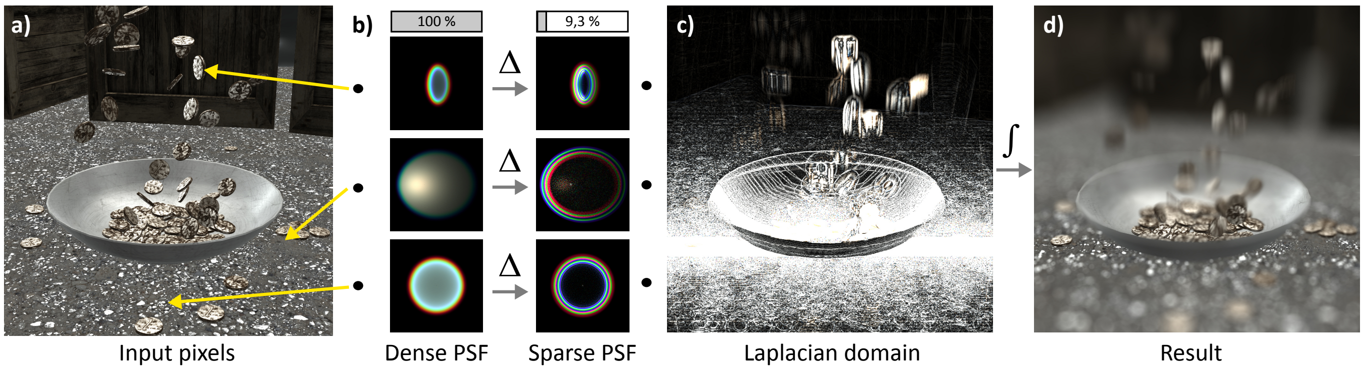

Fig. 1. Computing motion blur and depth-of-field by applying a point spread function (PSF) to every pixel (**a**) is computationally costly. We suggest splatting a pre-computed sparse approximation of the Laplacian of a PSF (**b**) to the Laplacian of an image (**c**) that under integration provides the same result (**d**). Note the circular bokeh combined with motion blur (1024×1024 pixels, 2 layers, 190 ms, Nvidia GTX 980Ti at .97 SSIM to a path-traced reference).

Simulating combinations of depth-of-field and motion blur is an important factor to cinematic quality in synthetic images but can take long to compute. Splatting the point-spread function (PSF) of every pixel is general and provides high quality, but requires prohibitive compute time. We accelerate this in two steps: In a pre-process we optimize for sparse representations of the Laplacian of all possible PSFs that we call *spreadlets*. At runtime, spreadlets can be splat efficiently to the Laplacian of an image. Integrating this image produces the final result. Our approach scales faithfully to strong motion and large out-of-focus areas and compares favorably in speed and quality with off-line and interactive approaches. It is applicable to both synthesizing from pinhole as well as reconstructing from stochastic images, with or without layering.

CCS Concepts: • **Computing methodologies → Rasterization**; **Image-based rendering**; *Massively parallel algorithms*;

Additional Key Words and Phrases: Distribution rendering; Gradient Domain; Depth-of-field; Motion blur.

ACM Reference Format:
Thomas Leimkühler, Hans-Peter Seidel, and Tobias Ritschel. 2018. Laplacian Kernel Splatting for Efficient Depth-of-field and Motion Blur Synthesis or Reconstruction. *ACM Trans. Graph.* 37, 4, Article 55 (August 2018), 11 pages. https://doi.org/10.1145/3197517.3201379

Authors' addresses: Thomas Leimkühler, MPI Informatik, Saarland Informatics Campus, Campus E1.4, Saarbrücken, 66123, Germany, tleimkueh@mpi-inf.mpg.de; Hans-Peter Seidel, MPI Informatik, Saarland Informatics Campus, Saarbrücken, Germany; Tobias Ritschel, University College London, London, United Kingdom.

1 INTRODUCTION

Depth-of-field (DoF) and motion blur (MB) are a key ingredient to the look and feel of most cinematic-quality feature films [Goy 2013]. Reproducing them in synthesized imagery is a typical and well-understood part of most photo-realistic rendering systems. When it comes to efficient, interactive or even real-time rendering, current solutions to DoF and MB typically make several key assumptions that result in computational efficiency but come at the cost of reduced quality. A typical example is to assume DoF and MB to be independent, to be able to approximate the space-time lens transport by a convolution [Potmesil and Chakravarty 1981] and often to approximate their reconstruction using Gaussian filtering [Belcour et al. 2013; Egan et al. 2009; Munkberg et al. 2014; Soler et al. 2009; Vaidyanathan et al. 2015]. In this work, we devise a method to synthesize or reconstruct cinematic quality motion blur and depth-of-field, while retaining most of the efficiency of typical approximations.

Input to our method are pixels (Fig. 1, a), which we see as light-field samples, labeled with additional geometric and dynamic information. We can work on pixels coming both from simple and layered images, as well as from a pinhole camera (synthesis), or stochastic path-tracing/rasterization (reconstruction). The point spread function (PSF) of each input pixel can affect a very large image area. Computing this contribution from each input point to a high number of output pixels is both the key to high quality, but regrettably also the reason for slow execution speed (Fig. 1, b).

Our key idea is to perform the required splatting operations in the Laplacian domain (Fig. 1, c). While the spatial extent affected by the typical PSF can be very large, it remains compressible, i. e., sparse, in the Laplacian domain (Fig. 1, Δ in b). Therefore, instead of splatting dense contribution onto an image, we splat sparse points we call *spreadlets* onto the Laplacian of the image, which is finally transformed into the primal domain (Fig. 1, d) using a fast method [Farbman et al. 2011]. We operate on different models of spaces of all PSFs, depending on depth, motion and image position. A pre-process jointly optimizes for a sparse representation and a small reconstruction error of all PSFs in a particular space.

2 PREVIOUS WORK

Computer graphics has a long history in modeling the imperfections of physical lens systems and film exposure to the end of providing the desired cinematic fidelity of real imagery.

A general solution to account for MB and DoF is Monte-Carlo (MC) ray-tracing [Cook et al. 1984; Pharr et al. 2016] in combination with a proper camera model [Kolb et al. 1995]. While this is accurate, it is not yet feasible for the interactive applications we target.

Alternatively, micropolygon-based solutions [Cook et al. 1987] allow for distribution effects but have not become mainstream for interactive graphics. Stochastic rasterization [Akenine-Möller et al. 2007; Fatahalian et al. 2009] brings distribution effects close to interactive performance, but still has difficulties with large motion and the remaining noise.

An improvement in speed, mainly due to re-using shading, is based on aggregating multiple views [Haeberli and Akeley 1990; Leimkühler et al. 2017; Yu et al. 2010]. These lack subtle effects found in MC, such as the impact of view-dependent shading on DoF and MB. The results are still not interactive for large MB and DoF or combinations thereof.

For increased efficiency, image-based methods [Potmesil and Chakravarty 1981] have been popular for DoF and MB and found commercial use in practical applications such as games [Göransson and Karlsson 2007]. Using image-space filters has been proposed in many different forms [McGuire et al. 2012; Rosado 2007]. A typical DoF solution is to use one [Kraus and Strengert 2007], or multiple MIP fetches [Lee et al. 2008] or learned kernels in a neural network [Nalbach et al. 2017]. While very fast, these methods use convolution and are only correct for translation-invariant PSFs.

Ray-marching (layered) depth images [Lee et al. 2009, 2010] is another method to produce high-quality DoF at high speed. They support complex DoF, producing a cinematic effect. However, they still result in MC noise and we are not aware of extensions to MB.

Very complex DoF in a lens system is typically related to lens flares that has been simulated using dedicated approaches [Hullin et al. 2011]. Here, the PSF is colorful and varies drastically across the sensor. Again, it is not clear how to combine such specialized solutions with MB as they are already far from interactive.

The reconstruction of noise-free images from stochastic images has received substantial attention [Kontkanen et al. 2006; Lehtinen et al. 2011; McCool 1999; Sen and Darabi 2012]. In particular for DoF and MB, Fourier theory [Belcour et al. 2013; Egan et al. 2009; Munkberg et al. 2014; Soler et al. 2009] is of great importance. Munkberg and colleagues [2014] explicitly model the combination of MB and DoF. Such approaches account for the effect of DoF and MB as a filter that reduces the bandwidth and therefore allows blurring the image, eventually also reducing MC noise. We refrain from filtering the image in a gathering fashion and instead scatter a PSF for every sample. Conceptually, we extrapolate the full space-time contribution from a single sample.

Another design dimension is to either work with layers [Kraus and Strengert 2007; Lee et al. 2009, 2010] or without [Belcour et al. 2013; Egan et al. 2009; Potmesil and Chakravarty 1981; Soler et al. 2009]. Our approach addresses all combinations of layered/non-layered and reconstruction/synthesis as the input is merely a set of labeled pixels.

Heckbert [1986] has shown how filtering can be done effectively using differentials and Simard et al. [1999] have extended this idea to efficient, spatially-varying convolution. Other approaches use uniform filtering of basis functions to approximate spatially-varying filtering [Fournier and Fiume 1988]. Such linear bases do not work well when reproducing the detailed and discontinuous functions of a typical PSF, including circles, lines, capsules, etc. At any rate, convolving the image with any gathering filter is only a crude approximation to the effect of applying a scattering PSF.

In summary, we see that many different attempts to model DoF and MB exist, that some approaches seek to approximate the effect of spatially-varying convolutions, but regrettably no approach has yet managed to provide cinematic fidelity at interactive rates. We suggest, for the first time, using the differential domain to overcome this limitation. The gradient domain has previously been used for image processing [Bhat et al. 2010; Pérez et al. 2003], and recently received new interest in contexts such as image compression [Galić et al. 2008], vector graphics [Orzan et al. 2013], image-based rendering [Kopf et al. 2013], texture representation [Sun et al. 2012], and realistic image synthesis [Lehtinen et al. 2013], but not yet for MB or DoF and not in an interactive setting.

3 OVERVIEW

Our approach comprises of two steps: a pre-calculation (Fig. 2, a) followed by a runtime step (Fig. 2 b–e).

Pre-computation. The pre-computation (Fig. 2, b and Sec. 5) samples the space of all PSFs according to a specific PSF model. Each PSF is converted into the Laplacian domain where it is approximated by a sparse set of optimized points. This “spreadlet” representation is stored on disk.

Input. At runtime, input to our method are pixels labeled with shading and lens-time-etc. coordinates that we interpret as temporal light field samples. Such a sample captures the radiance emitted from a certain world position at a certain point in time through a certain lens position. This general notion allows to work on any simple, layered and stochastic (Fig. 2, b) image, produced either using a pinhole (OpenGL) rasterization, a deep framebuffer [Nalbach et al. 2014], LDI-style [Shade et al. 1998], using ray-tracing or by stochastic rasterization [Akenine-Möller et al. 2007].

Runtime. Actual rendering is performed on (soft) global depth layers [Kraus and Strengert 2007] (Fig. 2, c and Sec. 6). A global depth layer holds the appearance and transparency of pixels at a specific depth interval, where intervals are typically smaller close

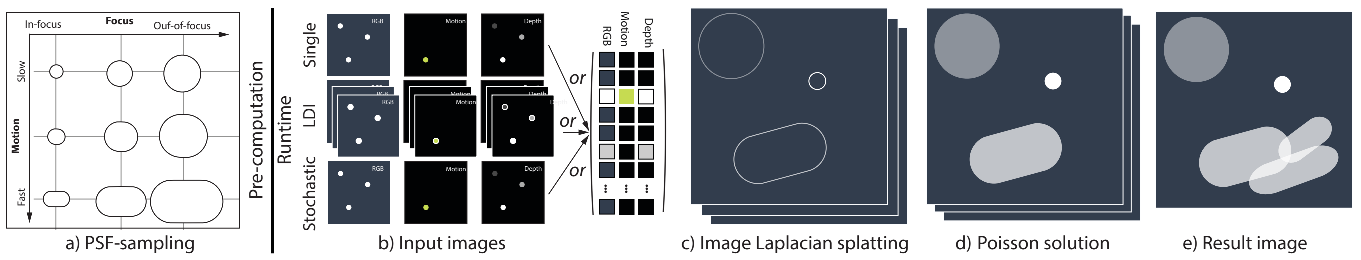

Fig. 2. Overview of our approach. The pre-computation **(a)** samples the space of all PSFs into a sparse representation. At runtime, one of the three types of images we support **(b)** are treated as lists of labeled pixels, here shown as three column vectors, the first holding appearance, the second motion and the third depth. To render, a PSF is splat for each pixel onto layered Laplacian images **(c)** that are integrated **(d)** and composed to produce the final result **(e)**.

to the camera. Layering is required to capture non-linear occlusion relations while our splatting performs linear addition within a layer. Composing all layers provides the final image. For every input pixel, the pre-computed sparse PSF representation is drawn additively onto one or multiple layers, each holding the Laplacian of the image to reconstruct. When all pixels were splat, all layers are efficiently integrated [Farbman et al. 2011], i. e., converted from the Laplacian into the primal domain (Fig. 2, d). Finally, all layers are composed into the result image (Fig. 2, e).

4 BACKGROUND

Here we recall the formalization of DoF and MB into point-spread functions as well as the role of Laplacians in image manipulation.

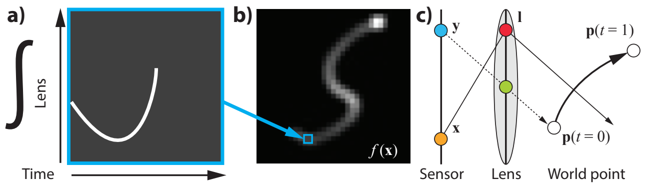

Fig. 3. Point-spread Functions: **a)** The lens-time integration domain and range for a relative sensor location **x**. At different time coordinates, different lens coordinates receive a contribution. **b)** There is one such function to integrate for every sensor location. **c)** The sensor-lens-world geometry.

Point-spread Functions. Classic DoF can be described using point-spread functions (PSFs) [Kolb et al. 1995]. In this work we formalize the combination of DoF and MB using generalized *space-time* PSFs (Fig. 3). Such a PSF describes the contribution of a constantly light-emitting 3D point moving during a shutter interval $\mathcal{T}$ along a path $\mathbf{p}(t) \in \mathcal{T} \rightarrow \mathbb{R}^3$ to every relative sensor location **x**. Therefore,

$$f(\mathbf{x}) = \int_{\mathcal{T}} \int_{\mathcal{L}} R(\mathbf{y} \rightarrow \mathbf{l}, \mathbf{p}(t))\, \mathrm{d}\mathbf{l}\, \mathrm{d}t, \qquad (1)$$

where, $\mathbf{x} \in \mathbb{R}^2$ is a 2D coordinate relative to $\mathbf{y} = P(\mathbf{p}(0)) + \mathbf{x}$, the absolute 2D sensor coordinate using projection P at start time, $\mathcal{L}$ is the lens area and $R(\mathbf{y} \rightarrow \mathbf{l}, \mathbf{p})$ is a Dirac that peaks when a ray $\mathbf{y} \rightarrow \mathbf{l}$ starting at 2D coordinate **y** in the sensor plane passing through 2D lens coordinate **l** (a two-plane light field parametrization) intersects the world point **p** (a “ray-point intersection”).

As we will be dealing with multiple PSFs for different motion, lenses and absolute sensor locations we define a space of PSFs $f(\mathbf{x})(\mathbf{s})$ subject to a PSF parameter vector $\mathbf{s} \in \mathbb{R}^{n_s}$, as in

$$f(\mathbf{x})(\mathbf{s}) = \int_{\mathcal{T}} \int_{\mathcal{L}} R(\mathbf{s})(\mathbf{y} \rightarrow \mathbf{l}, \mathbf{p}(\mathbf{s})(t))\, \mathrm{d}\mathbf{l}\, \mathrm{d}t, \qquad (2)$$

so the ray formation R and the motion **p** depend on the PSF parameter vector **s**.

Laplacians. The Laplacian $f(\mathbf{x}) \in \mathbb{R}^2 \rightarrow \mathbb{R}$ of a 2D image is defined as a scalar field

$$\Delta f(\mathbf{x}) = \operatorname{div}(\nabla f) = \frac{\partial^2 f(\mathbf{x})}{\partial x^2} + \frac{\partial^2 f(\mathbf{x})}{\partial y^2}, \qquad \mathbf{x} = (x, y)^{\mathsf{T}},$$

where div is the scalar divergence of a 2D vector field and ∇ the gradient, a 2D vector field, of a scalar function.

Typically, Green’s functions are used to reconstruct f from Δf. For this Poisson problem we consider an infinite 2D domain with no boundary conditions, which leads to the free-space Green’s convolution kernel $g(d) = \frac{1}{2\pi} \log(d + \epsilon)$. This kernel is radially symmetric and only depends on the distance d to the origin. The value ϵ (we use $\epsilon = .5\,px$) prevents a singularity at $d = 0$. Our image is compactly supported, that is, it is zero outside the PSF-extended unit square, which enforces enough constraints to make the integration solution unique. We will use G to denote the operator applying the convolution with Green’s function. This is routinely done in the Fourier domain [Bhat et al. 2008] or – even more efficiently – using a pyramidal scheme [Farbman et al. 2011]. A useful property is the rotation-invariance of G: integration of a rotated Laplacian yields the same results as the rotation of an integrated Laplacian.

In typical gradient image editing tasks, the manipulated [Bhat et al. 2010] or noisy [Lehtinen et al. 2013] gradient ∇f is given and a function f is to be found by employing the Laplacian as a means for finding a least-squares solution, often with additional regularization (screened). Our method never acts on gradient vector fields ∇f, but directly on the scalar Laplacian Δf, allowing both sparse processing and accurate, yet efficient integration [Farbman et al. 2011].

Laplacian Rasterization. Our approach heavily relies on the fact that splatting can be performed in the Laplacian domain, i. e.,

$$\operatorname{splat}(f, I) = G(\Delta(\operatorname{splat}(f, I))) = G(\operatorname{splat}(\Delta f, I)),$$

where splat(f, I) denotes additive splatting (scattering) of a spatially-varying function f into an image I. The second equality holds due to the commutativity and distributivity of all involved operations. Please note that the above would also hold if splatting was replaced by convolution [Heckbert 1986]. We opt for splatting, however, as it delivers higher-quality results for our application domain.

To understand the relation of MB and DoF to the Laplacian, lets consider the cost of splatting PSFs onto pixels using different methods in the following three paragraphs:

Drawing a *solid circle* area appears to require filling all the, say, n_a pixels inside an image with n_p pixels. Simple drawing equals to evaluating a function $f(\mathbf{x})$ that is 1 inside the circle and 0 outside. Alternatively, we could draw $\Delta f(\mathbf{x})$ and later solve for f, leading to the same result. Now, drawing this Laplacian would in general also cost n_a drawing operations and additionally n_p operations to solve a Poisson problem $f = \mathrm{G}\Delta f$ using a pyramidal approach [Farbman et al. 2011]. This does not yet provide any benefit.

Consider drawing the *sum of n_p circles*, one for each pixel. This requires $n_a \times n_p$ fill operations. Drawing using the Laplacian, still requires $2 \times n_p \times n_a$ operations, while classic drawing requires $n_p \times n_a$ so no immediate benefit here either.

The key insight is that the Laplacian of the typical PSFs found in DoF/MB is very *sparse*: Splatting only a sparse approximation comprising of n_a pixels of the Laplacian, can result in a very similar reconstructed result. This means that n_a is much smaller for Δf than n_a is for f. Our approach builds on this property.

5 PRE-CALCULATION: PSF SAMPLING

Sampling the space of all PSFs comprises of different stages (Fig. 4). First, we have to parametrize the space using a PSF model, such that we have a low-dimensional effective way to cover it as explained in Sec. 5.1. Second, we need to choose where to place samples, such that it is best represented where it is needed most (Sec. 5.2). Third, computing the PSF at specific sample positions can be challenging for complex lenses in combination with MB as explained in Sec. 5.3. Fourth, pre-filtering (Sec. 5.4) as with any sampling, also in the space of PSFs can prevent aliasing. Finally, the dense pre-filtered PSF sample is converted into a sparse set of points (spreadlets) approximating its Laplacian in an optimization step (Sec. 5.5).

5.1 PSF Model

Our approach allows for different PSF models (Tbl. 1) and resulting spaces to be discussed next in increasing complexity. Some PSFs are monochromatic, others support chromatic aberration. Coordinates in this space will be denoted as $\mathbf{s} \in \mathbb{S} = \mathbb{R}^{n_s}$.

Some PSF models exhibit a natural rotational symmetry. As the Laplacian and, consequently, the integration operator G are rotational invariant, we can omit the corresponding angular dimension during sampling. Note that this would not hold for other (differential) representations and reconstructions such as quad trees [Crow 1984]. While an omitted sampling dimension significantly reduces memory, symmetries are not necessary for our approach to work.

Circular Depth-of-field. The SimpleLens model is assuming a thin lens [Kolb et al. 1995]. The result is a one-dimensional space of monochromatic PSFs parametrized by the scalar circle of confusion (CoC) radius. The PSF is shift-invariant, i. e., points at the same depth are mapped to the same circle, regardless of the image position.

Table 1. The PSF model zoo.

Name	n_s	Color	Smp.	β	Size	Spar.
SimpleLens	1	✗	100	2.0	1.1 MB	31.3 %
Combined	2	✗	400	2.0, 1.0	4.2 MB	9.3 %
PhysicalLens	2	✓	4,000	2.5, 1.2	30.8 MB	2.2 %
Volume	1	✓	200	1.0	5.3 MB	18.4 %
Stylized	1	✓	200	1.5	9.0 MB	24.9 %

DoF and MB. The Combined model adds motion blur to the previous model. It uses linear motion of constant projected speed and no motion in depth. This supports arbitrary viewer and object motion and deformation. More complex motion is to be composed from linear segments. This results in a two-dimensional monochromatic model, with scalar motion length (speed) as an additional parameter.

Physical Lens. The PSF shape depends on absolute sensor position in our PhysicalLens model. Assuming this spatial layout to be rotation-invariant, we parametrize using the distance to the sensor center (eccentricity). This resulting space is two-dimensional: depth and eccentricity. Here, a chromatic PSF becomes important: In a physical lens model, light paths depend on wavelength and the PSF shows colored fringes. We use a simple bi-convex spherical lens in all our results.

Scattering. Light scattering in participating media can also be described as a PSF [Premože et al. 2004]. Notably, the shape of the PSF in this case resembles the Green's function itself, i. e., it is highly compressible. For our Volume model we implemented volumetric light tracing in a homogeneous medium using Woodcock tracking and a Henyey-Greenstein (HG) phase function. The parameter of this space is the distance to the camera and the medium parameters remain fixed. We use a density of .9 and a HG anisotropy of .8 and RGB albedo of (1, 1, .9).

Stylized. Finally, we show how our approach is not limited to any physical model, but can use any mapping between pixel labels and PSFs. The Stylized PSF in our experiments is a logo with chromatic aberrations, scaled depending on the distance to the focal plane.

5.2 Sample placement

Our approach achieves an appropriate cover of the PSF space using a *power remapping* and a *nested grid*. Both methods seek to place samples non-uniformly across the sample space, while retaining constant access time. For our PSF models it proved beneficial to allocate more samples to parameter ranges where artifacts due to quantization and pre-filtering (Sec. 5.4) would be most objectionable. This is typically the case for PSFs where one or multiple coordinates are small, e. g., the slow-motion or near-focus PSFs in Combined.

Power Remapping. The power remapping changes the physical PSF coordinates (Fig. 5, a) such that they cover the range from 0 to 1 non-uniformly by using a component-wise exponentiation $p(\mathbf{s}) = \mathbf{s}^{\beta}$ with a PSF model-specific vector β listed in Tbl. 1 as seen in Fig. 5, b. For $\beta > 1$ this results in a higher resolution for small coordinates and a lower resolution for large coordinates.

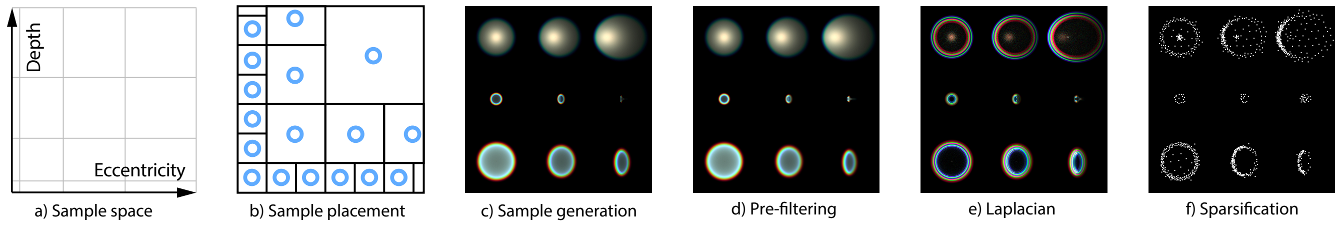

Fig. 4. The steps of our PSF sampling: **a)** Definition of the sample space. **b)** Non-uniform placement of samples (blue circles). **c)** Generation of the PSF at each sample. **d)** Pre-filtering in the sample space. **e)** Computing the Laplacian and **f)** sparsification into a set of points.

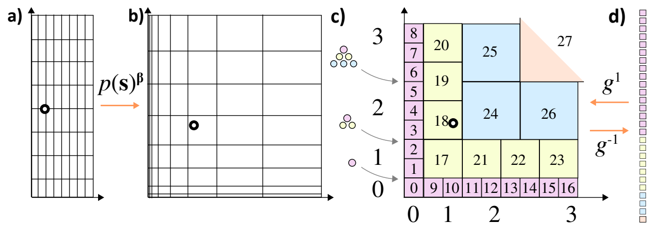

Fig. 5. Nested grid: **a)** Physical parameter grid. **b)** Its power remapping. **c)** Our nested grid topology. **d)** List of cells. (Please see text in Sec. 5.2.)

Nested Grid. A straight-forward solution is to sample in a regular grid after the power-remapping. While a regular grid can be accessed in constant time it requires exponential pre-compute time and storage. Note that the power remapping does not change this property. We observe that our PSF models allow for a dramatically reduced resolution for large coordinates to such an extent that we chose to abandon the grid topology. Therefore, we suggest a *nested grid*, reducing the storage to polynomial time while retaining constant access time (Fig. 5, c).

We achieve this by increasing the length of the sample cell edges linearly with increasing coordinates for all required dimensions. This naturally results in a nested structure with different resolution levels (colors and large numbers in Fig. 5, c). We note that for the 2D 9×9 example shown in Fig. 5 a regular grid would have 81 entries, while our nested grid requires 28 entries. To work with such grids, we require two functions: a mapping $g(\mathbf{s}) \in \mathbb{R}^{n_s} \to \mathbb{N}^0$ from a continuous coordinate to an index and an inverse mapping $g^{-1}(i) \in \mathbb{N}^0 \to \mathbb{R}^{n_s}$ from an index to a coordinate. The forward map is required at runtime, the backward one at the pre-computation step.

The backward mapping $g^{-1}(i)$ is constructed incrementally by placing boxes until a level is filled, continuing until the space is filled. Note that some of those boxes are not cubic, as a cube would fall outside the space on such a lattice. It is consequently stored as a list of boxes (Fig. 5, d).

The forward mapping $g(\mathbf{s})$ is performed in two steps: First, we compute the minimum coordinate $s_{\min} = \min(s_1, \ldots, s_{n_s})$. By construction of the nested grid, this value determines the resolution level. Since every level starts at a triangular number (triangles in Fig. 5, c), the level index l equals the *triangular root* of $s_{\min}$, i. e., $l = \lfloor(\sqrt{8s_{\min} + 1} - 1)/2\rfloor$. Second, the final cell index i is the sum $i = i_1 + i_2$ of the *inter-level* index and the *intra-level* index. The inter-level index i_1 is the sum of all indices before the current level l and we store this for all levels in a small table (indices of the boxes on the diagonal in Fig. 5, c). The intra-level index i_2 is computed just as in a regular grid within the level.

5.3 Sample generation

Sampling $f(\mathbf{s}_i)$ for each $\mathbf{s}_i$ means to evaluate many (for each pixel) complex integrals as defined in Eq. 1. The value of the function is an RGB 2D image, for us of resolution 512 pixels square. While a closed-form solution might exist for special models, such as SimpleLens, the task becomes harder for motion, forming capsule-shaped intensity-varying profiles as well as the famous cat eye-shaped flares. To compute the PSF of a complex lens system, including chromatic aberration is a research question on its own [Hullin et al. 2011]. The very general solution is light tracing [Dutré et al. 1993], which we opt to use, as it scales to complex lens systems including time-sampling. We typically use 700 million rays per PSF in a specialized GPU implementation. The high number of rays is required as differentiation in later steps will amplify any remaining variance. Note that our run-time efficiency is independent of the compute time of the PSF, only the sparsity in the Laplacian domain is relevant.

5.4 Pre-filtering

One sample $\mathbf{s}_i$ is a representative for an entire hyper-volume $\mathcal{S}_i$ in the space of PSFs. As we will use a single discrete pre-computed PSF sample that is nearest to the PSF required at runtime, a PSF sample should represent all PSFs that are closer to it than to any other. Failure to do so would result in aliasing or require a prohibitively large number of PSF samples. As a solution we suggest to pre-filter the PSF as in

$$\bar{f}(\mathbf{x})(\mathbf{s}_i) = \int_{\mathcal{N}(\mathbf{s}_i)} \int_{\mathcal{T}} \int_{\mathcal{L}} r(||\mathbf{s}_i - \mathbf{s}||) R(\mathbf{s})(\mathbf{y} \to \mathbf{l}, \mathbf{p}(\mathbf{s})(t))\, \mathrm{d}\mathbf{l}\, \mathrm{d}t\, \mathrm{d}\mathbf{s}\,,$$

for all samples in its neighborhood $\mathcal{N}(\mathbf{s}_i)$, where $r(d)$ is a reconstruction kernel such as a Gaussian. This can be achieved by just another outer integration over the hyper-volume of the neighborhood in PSF space in the light tracing MC loop evaluating Eq. 1 above. Instead of tracing a particle through always the same PSF $f(\mathbf{s}_i)$, the PSF parameters are varied as well to fall into $\mathcal{N}(\mathbf{s}_i)$. For SimpleLens, instead of using a single discrete confusion, a range of confusions is used, etc. The neighborhood $\mathcal{N}(\mathbf{s}_i)$ is a simple-to-filter axis-aligned box with varying extent that can be computed from the inverse sample density used in Sec. 5.2.

Effectively, pre-filtering blurs the PSF spatially, trading aliasing against blur. As a typical PSF is spatially band-limited as well – no CoC in a real camera system is fully sharp – this appears plausible.

5.5 Sparsification

Instead of storing each PSF $\bar{f}(\mathbf{s}_i)$, which is dense, we store its sparse Laplacian $\Delta\bar{f}(\mathbf{s}_i)$. This helps representing entire areas of the dense PSF by sparse isolated peaks we call "spreadlets". We therefore would like to find a set of $n_{\mathrm{p},i}$ points with 2D position $\mathbf{x}_{i,j}$ and values $\Delta\bar{f}_{i,j}$ that minimizes the reconstruction cost

$$c(n_{\mathrm{p},i}, \mathbf{x}_i, \Delta\bar{f}_i | f) = \int_{(0,1)^2} | \underbrace{f(\mathbf{x})}_{\text{Signal}} - \underbrace{\mathrm{G}\sum_{j=0}^{n_{\mathrm{p},i}} \Delta\bar{f}_{i,j} \mathbb{1}(\mathbf{x}_i, \mathbf{x})}_{\text{Reconstruction}} | \mathrm{d}\mathbf{x},$$

in respect to a PSF f, where $\mathbb{1}(\mathbf{x}_0, \mathbf{x}_1)$ is an indicator function that is one if $\mathbf{x}_0 = \mathbf{x}_1$ and zero otherwise.

Minimizing c poses several challenges: *i)* The cost landscape is highly non-convex, since every spreadlet adds one local extremum to it. *ii)* The reconstruction operator G has global support, resulting in a difficult condition. *iii)* The dimensionality of the problem is variable, since the point count $n_{\mathrm{p},i}$ is unknown. *iv)* We can splat only to discrete pixel coordinates, making the optimization a mixed problem where the $\mathbf{x}_i$ are integer and $\Delta\bar{f}_i$ are continuous.

Our attempts to partially (i. e., with a fixed $n_{\mathrm{p},i}$ and continuous $\mathbf{x}_i$) optimize via gradient descent or nonlinear conjugate gradient failed. However, we found the following practical procedure to minimize the cost in several steps (Fig. 6).

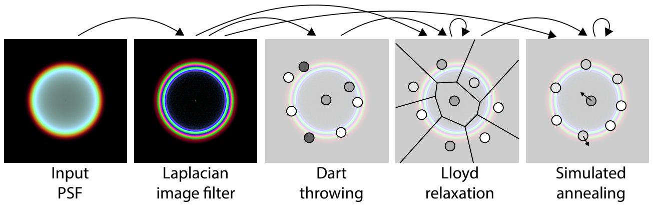

Fig. 6. Our four steps of PSF sparsification (conceptual illustration).

First, we apply a 3×3 Laplacian filter to $\bar{f}$, producing $\Delta\bar{f}$. This transformation into our target domain maps constant and linear regions to zero.

Second, we create a 2D Poisson disk pattern $\{\mathbf{x}_{i,0}, \ldots, \mathbf{x}_{i,n_{\mathrm{p},i}}\}$ with $|\Delta\bar{f}|$ as an importance function using a dart-throwing algorithm. We stop the placement after 10,000 failed random attempts. This step initializes the sparse spreadlet representation we seek to obtain, by placing samples according to the local complexity of the PSF and at the same time determining the first free variable $n_{\mathrm{p},i}$ of our cost function.

To improve the spatial arrangement, we run 50 iterations of Lloyd [1982] relaxation, again using $|\Delta\bar{f}|$ for weighting.

Next, we sum the Laplacian PSF values in the Voronoi cell of every $\mathbf{x}_{i,j}$ and store it as a value $\Delta\bar{f}(\mathbf{x}_{i,j})$. This way, each Voronoi cell of the Laplacian is collapsed into a single pixel. This significantly increases sparsity, especially for areas with large cells. As the cell area is inversely proportional to the Laplacian PSF value, the values $\Delta\bar{f}(\mathbf{x}_{i,j})$ are very similar for different j, i. e., have a similar contribution to the final image.

Finally, we apply 400 steps of simulated annealing, where in each iteration, we first pick a fraction (1 %) of the integer positions and change them by at most one pixel, and second, re-assign the values $\Delta\bar{f}(\mathbf{x}_{i,j})$ according to the new Voronoi cells as described above. When two points happen to fall on the same integer grid coordinate, they can be merged, further increasing sparsity.

6 RUNTIME: PSF SPLATTING

The representation of all possible PSFs acquired in the previous section can now be used to efficiently compute new images with distribution effects. The procedure is similar to a trivial code that iterates all pixels and densely draws their PSF in linear time [Lee et al. 2008]. Instead, we store (Sec. 6.1) and splat the sparse Laplacian of the PSF (Sec. 6.2) of all pixels, followed by a final transform of the entire image from the Laplacian to the primal domain (Sec. 6.3).

6.1 Sample storage

Each sample $\mathbf{s}_i$ has a varying number of points $n_{\mathrm{p},i}$. We concatenate all points $\mathbf{x}_{i,j}$ of all samples into a large sequence that is stored as a VBO P. The same is done for all function values $\Delta\bar{f}_{i,j}$ stored in a VBO V. The typical size of such a representation is several mega-bytes (Tbl. 1). The number of points changes for every sample: An in-focus sample requires fewer points than a moving lens flare. To efficiently handle a sequence of unstructured lists, we first pre-compute the vector of cumulative sums $n_{\mathrm{c},i}$ of all points in all samples with indices smaller than i and store it into a vector C.

6.2 Sample splatting

Splatting happens for all input pixels in all layers independently and in parallel. We will therefore describe it for a single pixel at absolute sensor position $\mathbf{y}$ here. Let $\mathbf{s}$ be the PSF coordinate of that pixel. We now pick the sample $\mathbf{s}_i$ that is closest to $\mathbf{s}$ and draw all points to positions $\mathbf{y} + \mathbf{x}_i$ with value $\Delta\bar{f}(\mathbf{x}_i)$.

GPU Implementation. Splatting is implemented in a compute shader that executes one thread for every pixel. Each thread fetches $\mathbf{s}$ for each pixel and computes i, the index of the nearest PSF sample. After compensating for the non-linearities, the nested grid structure of our space (Sec. 5.2) makes this a simple and efficient $O(1)$ operation. If the PSF model employs symmetries, they have to be applied at this step: e. g., for motion blur, the spreadlets are pre-computed for motion in a certain reference direction and now have to be rotated to align with a specific direction of motion. Next the spreadlet is multiplied by the pixel color and drawn in a `for` loop over all points using `atomicFloatAdd` into four `R32F` textures.

Boundary. The image has to be padded by a boundary large enough to accommodate for the largest PSF used. It is not sufficient to simply cut the kernel: consider a simple 1D example of a hat function that spans the image boundary. Depending on the PSF, that can include a translational part, to sample a certain output sensor size, the size of a virtual sensor that generates the input pixels might need to be substantially bigger or can be much smaller than the output sensor. The same applies for sampling considerations if the PSF is magnifying. Also note that the boundary only consumes memory, no splatting time and extra amount of integration time linear in its size.

Layering Details. Since our approach operates on labeled pixel lists (Fig. 2, b), we naturally support (soft) global depth layers, LDIs [Shade et al. 1998], or deep framebuffers [Nalbach et al. 2014] as input formats. However, we need to splat into global depth layers for being able to properly pre-integrate per-layer radiance and opacity [Vaidyanathan et al. 2015]. Splatting is done independently for each output layer. If soft layering is desired [Kraus and Strengert 2007], splats have to be drawn into more than one layer and weighted. In any case, we apply the re-weighting as suggested by Lee and co-workers [2008] when compositing the layers back-to-front.

In all our experiments we use n_l global input and output layers, where $n_\mathrm{l} = 1$ can be useful in some conditions. We bin them in units of constant parallax [Lee et al. 2009; Munkberg et al. 2014].

Note that layering only amplifies memory and merely shifts around the work: In particular the dominant splatting cost is not multiplied by n_l as a pixel is typically only contained in one layer (or two layers if soft) and empty pixels will be culled very early on.

Stochastic Frame-buffers. Special considerations are to be taken if the framebuffer is stochastic (Fig. 7, a). An example is DoF: a surface projecting to the sensor position $\mathbf{y}'$ at $\mathbf{l} = t = 0$ will move to a new sensor position $\mathbf{y}$ (yellow point in Fig. 7, b), as they are distributed across the entire circle of confusion.

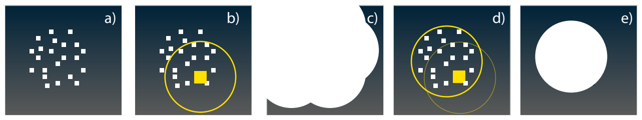

Fig. 7. Illustration of PSF splatting in a stochastic image with DoF: **a)** Stochastic image of a single bright point under defocus. **b)** A single PSF splat (yellow) centered around a single pixel at y. **c)** Overlay of all PSFs. **d)** The same single splat, but now centered around y′. **e)** Overlay of all PSFs.

Just running the above procedure on this data would mean to place another circle of confusion on an already distributed pixel i. e., to apply the PSF twice (Fig. 7, c). As every pixel has a unique distribution coordinate $\mathbf{s}_i$, we can re-compute its original absolute sensor position $\mathbf{y}'$, and splat the PSF around $\mathbf{y}'$ instead of $\mathbf{y}$. Conceptually, for the DoF example, this realigns the PSF such that it is drawn around the center of the CoC it belongs to, not around the pixel itself that is part of the CoC (Fig. 7, d–e). Consequently, non-stochastic input is a special case of stochastic input with $\mathbf{y} = \mathbf{y}'$.

6.3 Integration

After all points for all pixels were drawn, the image is transformed from the Laplacian into the primary domain. This is efficiently done using convolutional pyramids by Farbman [2011] which takes 6 ms for a 1024×1024 image.

6.4 Fast Track

In practice, some PSFs can have less sparsity than others. The main speed-up we achieve is for large PSFs that are sparse, which also implies that splatting small PSFs using our sparsification scheme is not effective. Fortunately, our approach can combine both strategies seamlessly. To this end, we maintain two images per layer: A Laplacian image to which sparse points are splat and a direct one. The decision to draw sparse or dense is made simply on the number of points. Both images are added after the Laplacian image was integrated. This strategy is used in all results shown in this paper and typically amounts to about 9% of the PSFs.

7 RESULTS

In this section we show qualitative (Sec. 7.1), quantitative (Sec. 7.2) results of our approach and an analysis of its properties (Sec. 7.3).

Methods. We compare Our approach to different alternatives:

A Splatting approach draws the dense ground-truth PSF. This is an upper bound on what we can achieve, as our Laplacians are just an approximation. We would hope to achieve similar quality, just at a much higher speed.

The Filtering method uses the same space of PSFs we use, but instead of splatting the PSF, we filter using the PSF. Note that this is an upper bound on what any filtering-based method can achieve. We expect to achieve both higher quality and speed. We do not apply filtering to PhysicalLens, as neither the pre-computed dense PSF images fit into memory, nor is it feasible to compute them for every pixel on the fly.

Many methods to remove noise, in particular the noise specific to path-traced images, exist [Belcour et al. 2013; Egan et al. 2009; Kontkanen et al. 2006; McCool 1999; Munkberg et al. 2014; Sen and Darabi 2012; Soler et al. 2009]. As all these methods have different trade-offs and assumptions, we here opt for BM3D, a general state-of-the-art image denoiser [Dabov et al. 2006] that has been used to denoise path-traced images before [Kalantari and Sen 2013].

The Reference method uses path tracing based on a reasonably implemented GPU ray-tracer with an SAH-build BVH where nodes are extended to bound space-time primitives.

7.1 Qualitative

Qualitative results of synthesis and reconstruction are shown in Fig. 8 and Fig. 9.

Synthesis. In Fig. 8 we see in “Whirl” how Our method produces detailed PSFs that add cinematic quality to the shot, with circular bokeh and long motion trails. MB and DoF arising from the complex motion patterns are faithfully synthesized, while the colorful specular highlights in “Coins” are transformed into overlapping, yet distinct circles of confusion. The rotational motion in “Gears”, here in combination with bright specular highlights under defocus, gives rise to appealing high-contrast image regions entirely defined by the PSFs produced. The “Rain” scene rendered with the physical model shows the expected complex cat-eye shapes in image corners, where the CoC is deformed. Chromatic aberration is reproduced as well. The same figure shows the comparison to alternative methods as insets. We see that the Filtering method looks quite different, while the method based on Splatting is very similar to Ours and the Reference.

Results for the Stylized and Volume PSF models are shown in Fig. 12. We see how stylization provides a non-physical effect where the PSFs take the shape of a logo, while for the participating media PSF the colors shift according to the model. In addition, a distinct

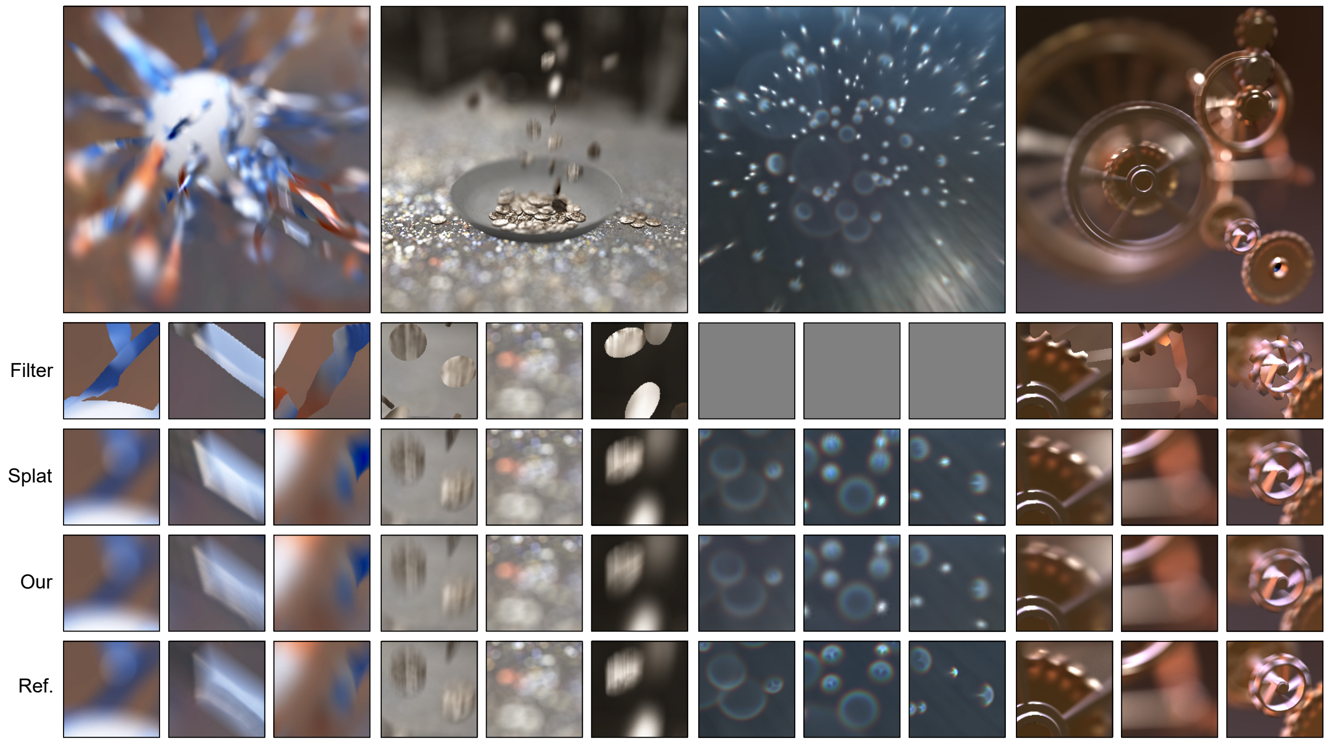

Fig. 8. Results of our (large) as well as other (insets) synthesis approaches on different scenes. (1024×1024 pixels).

blur can be observed, in particular for locations in the background, as expected from scattered light.

Reconstruction. In Fig. 9 we use our method for reconstructing from stochastic input. While the input already contains MB and DoF, our method preserves it, yet removes the noise. In several cases, our method reconstructs features that are almost invisible in the input to the naked eye, such as in "Rain". Please note how our carefully aligned high-frequency PSFs are able to reconstruct subtle semi-transparencies.

7.2 Quantitative

We provide quantitative results in terms of comparison to a reference and alternative approaches. Quality is measured in SSIM (larger is better) and speed in milliseconds. All comparisons are done in resolution 1024×1024 on an Intel Xeon E5-1607 CPU in combination with a Nvidia Geforce GTX 980Ti GPU. The pre-computation requires roughly 20 seconds for one PSF. Numerical results are stated in Tbl. 2 for synthesis and in Tbl. 3 for reconstruction. We see our approach consistently has the highest speed and provides images of high similarity to the reference. Splatting has a similar error, but is typically slower by almost one order of magnitude, as our representation is typically one order of magnitude more sparse.

7.3 Analysis

Here we analyze how our approach and variants thereof perform under different conditions. As we already established we can generate images similar to a reference, provided we have a suitable (sparse and low-error) PSF representation, we perform analysis purely on the space of PSFs images (the test set) instead of complex images. Three qualities are important: i) the *sparsity* in percentage (which translates into computational efficiency, up to a small additive constant overhead for integration), ii) the *similarity* measured between PSFs in terms of SSIM, and finally, iii) a good *efficiency* ratio between the two, measured in similarity-per-sparsity.

Sparsity. Our representation achieves an average sparsity of 9.3 % (Fig. 10, a) while retaining an average similarity of .97, which results in an efficiency if 10.75. We further look into the distribution of

Table 2. Numerical results for Fig. 8.

		Our		Filter		Splat		Ref.
	Model	Time	Err.	Time	Err.	Time	Err.	Time
Whirl	Comb.	**88.0** ms	**.94**	299.6 ms	.81	722.8 ms	.94	592 s
Coins	Comb.	**163.0** ms	**.96**	368.1 ms	.92	1930.9 ms	.96	232 s
Rain	Phy.	**126.3** ms	**.90**	—	—	7170.4 ms	.88	>1000 s
Gear	Comb.	**65.9** ms	**.94**	186.2 ms	.87	783.8 ms	.96	>1000 s

Table 3. Numerical results for Fig. 9.

		Our		BM3D	Splat		Ref.
	Model	Time	Err.	Err.	Time	Err.	Time
Whirl	Comb.	**75.8** ms	**.95**	.93	605.3 ms	.95	346 s
Coins	Comb.	**496.4** ms	**.94**	.91	2652.3 ms	.94	366 s
Rain	Comb.	**499.5** ms	**.96**	.89	2276.2 ms	.96	>1000 s
Gear	Comb.	**1699.1** ms	**.91**	.91	3495.8 ms	.92	>1000 s

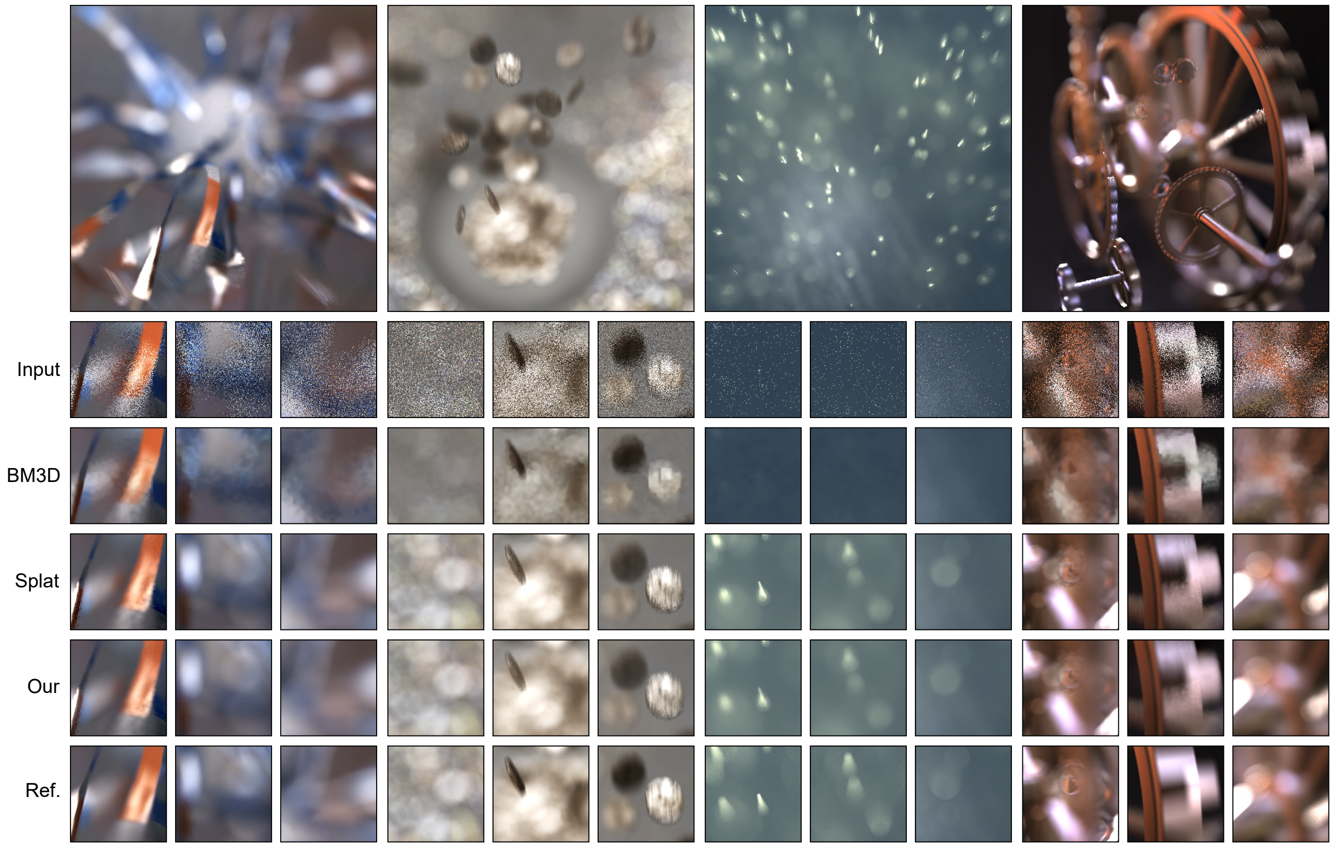

Fig. 9. Results of our (large) as well as other (insets) reconstruction approaches on different scenes. (1024×1024 pixels, 1spp path traced input).

sparsity (Fig. 10, b) and similarity (Fig. 10, c). The sparsity distribution is peaked around very sparse PSFs and most solutions have a high similarity. Only very few PSFs have a low similarity.

Laplacian Domain. We have chosen to use the Laplacian domain while other representations could also achieve the sparsification required. A typical sparse coding that is efficient to produce and generate is wavelets, i. e., a quad tree [Crow 1984] (QT). To this end we convert the test set into the QT domain. In Fig. 10, a, adjusting QT for a similar sparsity, we find a lower similarity of .89, providing an inferior efficiency of 9.67. A distribution of sparsity and similarity is seen in Fig. 10, b and c. We see that the sparsity is not as peaked for very sparse solutions while at the same time, many more solutions are of low similarity. Note that a QT is not rotation-invariant, leading to an order of magnitude more memory requirement. This indicates the Laplacian is a good representation for PSF sparsification in terms of memory and speed.

Spreadlet Count. We have chosen two other operational points of our approach in Fig. 10, d where the average sparsity is roughly half and twice as large as the one we suggest to use. We observe that quality saturates at a very high similarity to a reference, indicating that sparsity can be used to control quality.

Sample Count. We instrument the relation of PSF sample count and average similarity between the closest sampled PSF and the ground-truth PSF at 1,000 random coordinates in Fig. 10, e. We see that increasing the number of samples increases similarity. This is as pre-filtering introduces blur. In practice, the PSF is not applied to individual pixels, but to groups of pixels which also is a blur, resulting in the good end-image similarity we observe.

Approximation Quality. Our approach builds on the observation that a sparse approximation of the Laplacian of certain PSFs is feasible. To better understand the approximation behavior of our sparsification scheme we analyze the reconstruction quality for different PSF types of different sizes (S/M/L) with varying sparsity in Fig. 10, f. We compare isotropic Gaussian functions ($\sigma = 25/40/55\,px$) with our DoF CoCs ($r = 25/50/100\,px$) and capsule-shaped motion-blurred CoCs ($r = 50/50/100\,px, l = 50/128/128\,px$). We observe that the reconstruction quality for the typical PSFs we target increases with spatial extent, while the opposite can be found for corresponding Gaussians. This can be attributed to the relatively small total variation of our large-scale target PSFs, allowing a representation by a small set of Laplacian peaks. This is in contrast to the uniform smoothness of Gaussians. For the binary DoF kernels, which exhibit natural sparsity in the Laplacian domain, the reconstruction quality saturates at 100% once enough spreadlets are allocated.

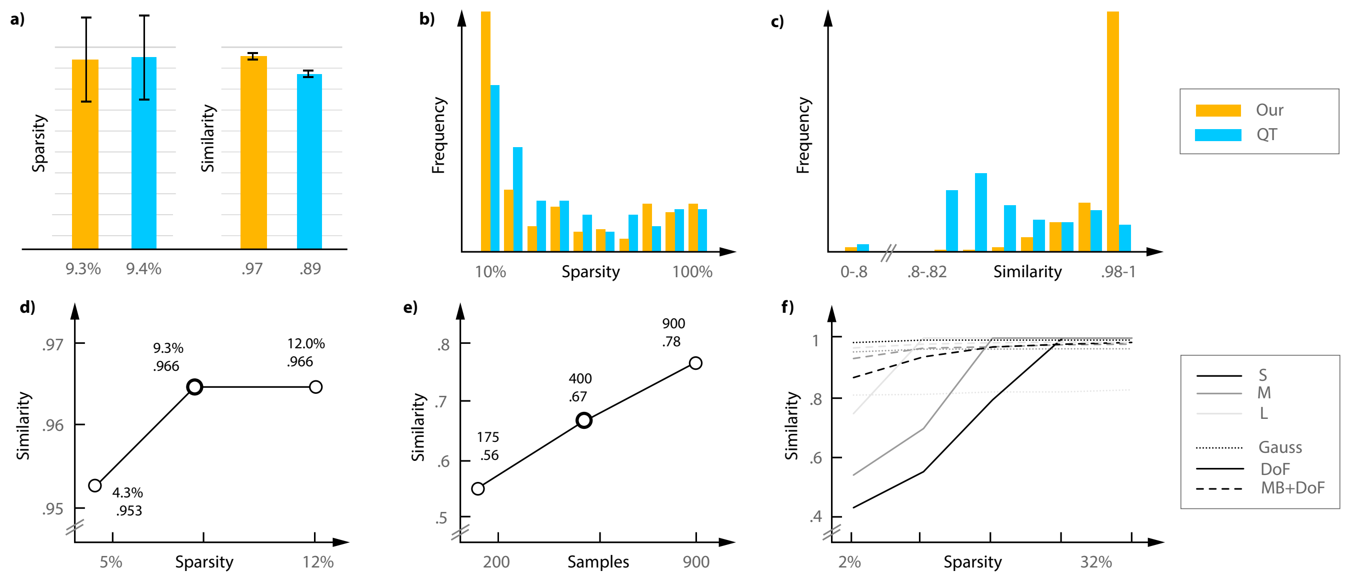

Fig. 10. Analysis for Combined: **a)** Mean sparsity and similarity for our approach and a quad tree (QT). **b, c)** Sparsity resp. similarity histograms for Laplacian and QT. Frequency counts how often a PSF with this property occurs. **d)** Relation of sparsity and similarity for a 0.5× and a 2×-sparsity operational point. **e)** Relation of sample count and similarity for the same operational points. **f)** Relation of sparsity (log. scale) and similarity for differently-sized (S, M, L) PSFs.

Optimization. Here we report results of ablational studies regarding our optimization procedure. The mean L1 error across our PSF corpus relative to dart throwing only (100%) is 96.7% for dart throwing and Lloyd relaxation, 87.1% for dart throwing and simulated annealing, and 86.5% for the full procedure. While this is a modest mean improvement, it removes outliers that are visually disturbing.

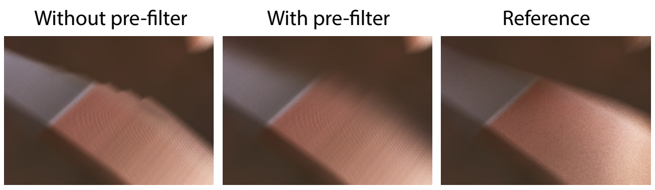

Fig. 11. Comparing the effect of pre-filtering for the Combined model.

7.4 Limitations

While our approach is impaired by typical limitations inherent to image-based synthesis and reconstruction, such as layer quantization and the assumption of diffuse reflectance, we discuss three limitations and artifacts unique to our approach in the following paragraphs.

Pre-filtering. The effect of pre-filtering for the Combined model is studied in Fig. 11. Without pre-filtering, we see discontinuity errors. With pre-filtering the aliasing is converted into blur, a less suspicious artifact when comparing to the reference.

Spreadlet Undersampling. Occasionally, for PSFs with small spatial extent and high frequencies, the dart throwing step of our optimization procedure fails to allocate enough spreadlets. This results in blotchy artifacts after reconstruction, as can be observed in the third main column of Fig. 8.

Curse of Dimensionality. Our approach requires full sample coverage of the PSF spaces it operates on. Even though the sample count is reduced by utilizing our nested grid structure and by exploiting rotational symmetries, higher-dimensional PSF models, like a physical lens in combination with a higher-order motion model, would require a prohibitive amount of pre-computation.

8 CONCLUSION

We have described the first method to achieve interactive performance when synthesizing and reconstructing cinematic-quality combinations of DoF and MB. This was achieved by splatting pre-computed sparse PSFs in the Laplacian domain.

In future work we would like to overcome the limitation to a low number of parameter dimensions due to pre-calculation. For example, we approximate the camera motion using linear segments, mostly because of the dimensionality-restriction imposed by the pre-calculation. While DoF and MB are a typical and important combination, the ability to change the DoF in a dynamical optical system is desirable. We see our approach as a toolkit from which more complex interactions between depth of field, motion blur, participating media or artistic control can be made to fit computational efficiency requirements in the future.

ACKNOWLEDGEMENTS

The authors would like to thank Gabriel Brostow, Bernhard Reinert, and Rhaleb Zayer. This work was partly supported by the Fraunhofer and the Max Planck cooperation program within the framework of the German pact for research and innovation (PFI).

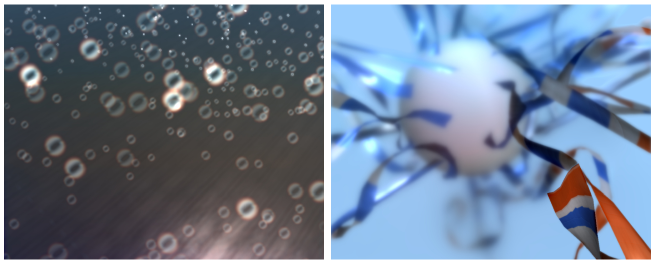

Fig. 12. Results for the Stylized (left) and Volume (right) PSF models.

REFERENCES

Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. 2007. Stochastic rasterization using time-continuous triangles. In *Proc. Graphics Hardware.* 9.

Laurent Belcour, Cyril Soler, Kartic Subr, Nicolas Holzschuch, and Fredo Durand. 2013. 5D covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph.* 32, 3 (2013).

Pravin Bhat, Brian Curless, Michael Cohen, and C Zitnick. 2008. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. *ECCV* (2008), 114–28.

Pravin Bhat, C Lawrence Zitnick, Michael Cohen, and Brian Curless. 2010. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Trans. Graph.* 29, 2 (2010).

Robert L Cook, Loren Carpenter, and Edwin Catmull. 1987. The REYES image rendering architecture. In *ACM SIGGRAPH Computer Graphics*, Vol. 21. 95–102.

Robert L Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, Vol. 18. ACM, 137–145.

Franklin C Crow. 1984. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics* 18, 3 (1984), 207–12.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. 2006. Image denoising with block-matching and 3D filtering. In *Proc. SPIE*, Vol. 6064.

Philip Dutré, Eric P Lafortune, and Yves Willems. 1993. Monte Carlo light tracing with direct computation of pixel intensities. In *Proc. Computational Graphics and Visualisation Techniques.* 128–37.

Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. (Proc. SIGGRAPH)* 28, 3 (2009).

Zeev Farbman, Raanan Fattal, and Dani Lischinski. 2011. Convolution pyramids. *ACM Trans. Graph.* 30, 6 (2011), 175–1.

Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R Mark, and Pat Hanrahan. 2009. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proc. HPG.* 59–68.

Alain Fournier and Eugene Fiume. 1988. Constant-time filtering with space-variant kernels. In *ACM SIGGRAPH Computer Graphics*, Vol. 22. ACM, 229–238.

Irena Galić, Joachim Weickert, Martin Welk, Andrés Bruhn, Alexander Belyaev, and Hans-Peter Seidel. 2008. Image compression with anisotropic diffusion. *J Math. Imaging and Vision* 31, 2 (2008), 255–69.

Jhonny Göransson and Andreas Karlsson. 2007. Practical post-process depth of field. In *GPU Gems 3.* 583–606.

Michael Goy. 2013. *American cinematographer manual.* Vol. 10. Am. Cinemat.

Paul Haeberli and Kurt Akeley. 1990. The accumulation buffer: hardware support for high-quality rendering. *ACM SIGGRAPH Computer Graphics* 24, 4 (1990), 309–318.

Paul S Heckbert. 1986. Filtering by repeated integration. In *ACM SIGGRAPH Computer Graphics*, Vol. 20. 315–21.

Matthias Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee. 2011. Physically-based real-time lens flare rendering. *ACM Trans Graph. (Proc. SIGGRAPH Asia)* 30, 4 (2011), 108.

Nima Khademi Kalantari and Pradeep Sen. 2013. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Comp. Graph. Forum* 32, 2 (2013), 93–102.

Craig Kolb, Don Mitchell, and Pat Hanrahan. 1995. A realistic camera model for computer graphics. In *SIGGRAPH.* 317–24.

Janne Kontkanen, Jussi Räsänen, and Alexander Keller. 2006. Irradiance filtering for Monte Carlo ray tracing. , 259–272 pages.

Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. 2013. Image-based rendering in the gradient domain. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 6 (2013).

Martin Kraus and Magnus Strengert. 2007. Depth-of-Field Rendering by Pyramidal Image Processing. In *Comp. Graph Forum*, Vol. 26. 645–54.

Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. 2009. Depth-of-field rendering with multiview synthesis. In *ACM Trans. Graph. (Proc. SIGRAPH Asia)*, Vol. 28.

Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. 2010. Real-time lens blur effects and focus control. *ACM Trans. Graph (Proc. SIGGRAPH)* 29, 4 (2010).

Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. 2008. Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers. *Comp. Graph. Forum* 27, 7 (2008), 1955–62.

Jaakko Lehtinen, Timo Aila, Jiawen Chen, Samuli Laine, and Frédo Durand. 2011. Temporal Light Field Reconstruction for Rendering Distribution Effects. *ACM Trans. Graph. (Proc. SIGGRAPH)* 30, 4 (2011).

Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. 2013. Gradient-domain metropolis light transport. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 4 (2013), 95.

Thomas Leimkühler, Hans-Peter Seidel, and Tobias Ritschel. 2017. Minimal Warping: Planning Incremental Novel-view Synthesis. *Comp. Graph. Forum* 36, 4 (2017), 1–14.

Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–37.

Michael D McCool. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph.* 18, 2 (1999), 171–94.

Morgan McGuire, Padraic Hennessy, Michael Bukowski, and Brian Osman. 2012. A reconstruction filter for plausible motion blur. In *i3D.* 135–42.

Jacob Munkberg, Karthik Vaidyanathan, Jon Hasselgren, Petrik Clarberg, and Tomas Akenine-Möller. 2014. Layered reconstruction for defocus and motion blur. *Comp. Graph. Forum* 33, 4 (2014), 81–92.

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen Space Shading. *Comp. Graph. Forum* 36, 4 (2017), 65–78.

Oliver Nalbach, Tobias Ritschel, and Hans-Peter Seidel. 2014. Deep Screen Space. In *I3D.* ACM.

Alexandrina Orzan, Adrien Bousseau, Pascal Barla, Holger Winnemöller, Joëlle Thollot, and David Salesin. 2013. Diffusion curves: a vector representation for smooth-shaded images. *Comm. ACM* 56, 7 (2013), 101–8.

Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. *ACM Trans. Graph (Proc. SIGGRAPH)* 22, 3 (2003), 313–18.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation.* Morgan Kaufmann.

Michael Potmesil and Indranil Chakravarty. 1981. A lens and aperture camera model for synthetic image generation. *ACM SIGGRAPH Computer Graphics* 15, 3 (1981), 297–305.

Simon Premože, Michael Ashikhmin, Jerry Tessendorf, Ravi Ramamoorthi, and Shree Nayar. 2004. Practical rendering of multiple scattering effects in participating media. In *Proc. EGWR.* 363–74.

Gilberto Rosado. 2007. *Motion blur as a post-processing effect.* 575–81.

Pradeep Sen and Soheil Darabi. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 3 (2012), 18–1.

Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. 1998. Layered depth images. In *Proc. SIGRAPH.* 231–42.

Patrice Simard, Léon Bottou, Patrick Haffner, and Yann LeCun. 1999. Boxlets: a fast convolution algorithm for signal processing and neural networks. In *NIPS.* 571–7.

Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. 2009. Fourier depth of field. *ACM Trans. Graph.* 28, 2 (2009), 18.

Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. 2012. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 4 (2012), 74–1.

Karthik Vaidyanathan, Jacob Munkberg, Petrik Clarberg, and Marco Salvi. 2015. Layered light field reconstruction for defocus blur. *ACM Trans. Graph.* 34, 2 (2015).

Xuan Yu, Rui Wang, and Jingyi Yu. 2010. Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering. *Comp. Graph. Forum* 29, 7 (2010), 2099–2107.

Received January 2018; revised April 2018; final version May 2018; accepted May 2018