Deep Point Correlation Design

THOMAS LEIMKÜHLER, MPI Informatik, Saarland Informatics Campus
GURPRIT SINGH, MPI Informatik, Saarland Informatics Campus
KAROL MYSZKOWSKI, MPI Informatik, Saarland Informatics Campus
HANS-PETER SEIDEL, MPI Informatik, Saarland Informatics Campus
TOBIAS RITSCHEL, University College London

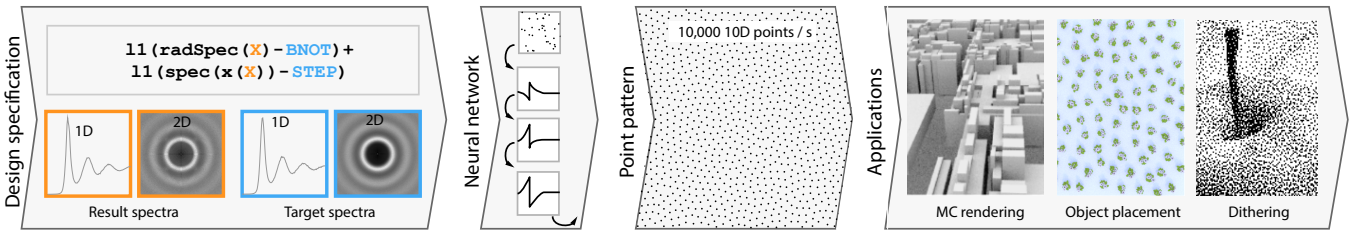

Fig. 1. We suggest a framework for designing correlated point patterns. A design specification **(left)** is used to train an architecture mapping random points to correlated ones **(middle)**. This architecture produces ten-thousands of points per second with use in rendering, object placement or dithering **(right)**.

Designing point patterns with desired properties can require substantial effort, both in hand-crafting coding and mathematical derivation. Retaining these properties in multiple dimensions or for a substantial number of points can be challenging and computationally expensive. Tackling those two issues, we suggest to automatically generate scalable point patterns from design goals using deep learning. We phrase pattern generation as a deep composition of weighted distance-based unstructured filters. Deep point pattern design means to optimize over the space of all such compositions according to a user-provided *point correlation loss*, a small program which measures a pattern's fidelity in respect to its spatial or spectral statistics, linear or non-linear (e. g., radial) projections, or any arbitrary combination thereof. Our analysis shows that we can emulate a large set of existing patterns (blue, green, step, projective, stair, etc.-noise), generalize them to countless new combinations in a systematic way and leverage existing error estimation formulations to generate novel point patterns for a user-provided class of integrand functions. Our point patterns scale favorably to multiple dimensions and numbers of points: we demonstrate nearly 10 k points in 10-D produced in one second on one GPU. All the resources (source code and the pre-trained networks) can be found at https://sampling.mpi-inf.mpg.de/deepsampling.html.

CCS Concepts: • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Stochastic Sampling; Blue noise; Optimization; Deep learning

Authors' addresses: Thomas Leimkühler, MPI Informatik, Saarland Informatics Campus, tleimkueh@mpi-inf.mpg.de; Gurprit Singh, MPI Informatik, Saarland Informatics Campus, gsingh@mpi-inf.mpg.de; Karol Myszkowski, MPI Informatik, Saarland Informatics Campus; Hans-Peter Seidel, MPI Informatik, Saarland Informatics Campus; Tobias Ritschel, University College London.

0730-0301/2019/11-ART226 $15.00
https://doi.org/10.1145/3355089.3356562

ACM Reference Format:
Thomas Leimkühler, Gurprit Singh, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel. 2019. Deep Point Correlation Design. *ACM Trans. Graph.* 38, 6, Article 226 (November 2019), 17 pages. https://doi.org/10.1145/3355089.3356562

1 INTRODUCTION

Point patterns have many important uses in computer graphics, linking apparently disparate topics such as natural placement of procedural plants, accurately casting shadows from an area light or arranging artistic stipples in a visually pleasing fashion. Many classic algorithms have been proposed to generate point patterns e. g., Lloyd's [1982] relaxation algorithm, dart throwing [McCool and Fiume 1992] or deterministic methods [Niederreiter 1992]. Their properties are analyzed in terms of low discrepancy [Shirley 1991], spectra [Yellott 1983], differentials [Öztireli and Gross 2012; Wei and Wang 2011] and their use in Monte Carlo (MC) and Quasi Monte Carlo (QMC) integration [Cook 1986; Keller et al. 2012; Öztireli 2016; Pilleboue et al. 2015; Subr and Kautz 2013].

Designing methods to provide the desired pattern quality for different applications is an active topic of research [De Goes et al. 2012; Fattal 2011; Heck et al. 2013; Kailkhura et al. 2016; Wei and Wang 2011; Zhou et al. 2012]. Devising such point patterns typically requires complex mathematical derivations which are only applicable in specific conditions, implementation effort, and finally compute time in order to run an optimization which produces a point set.

In this paper, we add a new level of abstraction and suggest to use modern deep learning to optimize over the space of point pattern generation methods itself. Instead of mathematical derivation, a user of our system provides a straightforward implementation of the desired properties in form of a *loss* (code in Fig. 1) demanding e. g., "a blue noise (BN) spectrum both in 2D and 1D (along the x axis)". We then optimize over the learnable parameters of a deep pipeline of recursive and unstructured filters which map uncorrelated to

correlated point patterns. Notably, we learn this weakly supervised, without observing any instances of the desired point patterns – some of the patterns we produce are not known how to produce – but directly from a description of the desired properties alone. The pipeline is deep, so that several steps of correction (typical ca. 40) can occur and communicate. We use filters based on distance measured in subspaces which allows both scaling to multiple (i. e., up to 10-D) dimensions but also supporting anisotropy that can achieve different characteristics in different subspaces. Our unstructured filters support sparse point sets, as a dense kernel representation would not scale to multiple dimensions. After learning has finished, producing points at deployment does only require running the resulting recursive massively parallel filters and no optimization.

In summary the contributions made in this paper are:

- A parallel method to generate point patterns using tunable, recursive and unstructured filtering in multiple dimensions.
- A method to learn these filters from prescribed design goals alone, without mathematical derivation or coding.
- Novel point patterns such as multi-dimensional isotropic BN, and mixed forms of BN in multiple dimensions and subspaces.
- MC estimation of the radially-averaged spectra that allows for analysis and optimization in higher dimensions.
- Learning well-suited sampling patterns for a class of functions matching diverse computer graphics tasks.

2 POINT PATTERNS IN COMPUTER GRAPHICS

Point patterns can be quite useful for many computer graphics tasks. Which point pattern, however, is most suited for solving a specific task is an active topic of research. We will here quickly review different properties of point patterns (refer to [Yan et al. 2015] for a recent in-depth survey) before relating to the machine learning and domain-specific language background relevant for this work.

2.1 Correlated Patterns

Blue noise. Yellot [1983] first noted that the receptors on the retina are neither regular nor random but follow very specific patterns where they keep a minimal distance. These patterns are routinely characterized by their expected *power spectrum* [Lagae and Dutre 2008; Ulichney 1988]. For two or more dimensions, the full spectrum is often further *radially averaged* to a one-dimensional subspace. The variance of this radial estimate is called the radial *anisotropy* which is low for radially symmetric (isotropic) patterns and large for others. We use these quantities as a loss for our trainable architecture.

Power spectra are frequently characterized by their "colors". A *blue noise* (BN) pattern has a power spectrum with little energy in the low-frequency region. BN was first used in graphics for dithering [Ulichney 1988] and stippling [Oliver et al. 2001; Secord 2002]. Classic ways to produce BN patterns are dart throwing [McCool and Fiume 1992] and Lloyd relaxation [Lloyd 1982]. The first can be slow, while the latter often suffers from regularity artifacts, that need extra effort to be overcome [Balzer et al. 2009; Claici et al. 2018; De Goes et al. 2012]. As in the context of dithering, models of human perception can be used to improve quality [Mulligan and Ahumada 1992]. BN patterns are also used for MC integration-based image synthesis, as they shift the error into the high-frequency bands, to which humans are less sensitive [Cook 1986] and signals contain less energy [Mandelbrot 1983]. This low frequency region also directly affects the variance and its convergence during MC estimation [Singh et al. 2019b]. Besides BN, other colors of noise are useful in tasks such as simulating physical and biological distributions [Condit et al. 2000].

Heck et al. [2013] were the first to address oscillations in the power spectrum. Kailkhura et al. [2016] suggest to add a stair to the spectrum to widen its zero-region. Our method can generate both step and stair spectra as targets.

Methods that generate point patterns with desired spectrum exist [Heck et al. 2013; Kailkhura et al. 2016; Wei and Wang 2011; Zhou et al. 2012] but operate on a different level of abstraction: Output of those methods are point patterns, while the output of ours is a point generation method. Consequently, their methods need to be run every time when a point pattern is produced, while we only optimize during training and execute learned filters to actually produce patterns in the deployed software. We shift the time-consuming optimization over point patterns at run-time into an abstract learning pre-process that optimizes over point pattern generation methods.

There also exists approaches that generate blue noise using *dart throwing* [Lagae and Dutre 2008; McCool and Fiume 1992] or max-min distance. In such a pattern, the minimal distance from one point to the others is maximized over all points i. e., all points keep a minimal distance. Many technical alternatives have been considered to produce blue noise patterns such as variational [Chen et al. 2012], optimal transport [Claici et al. 2018; De Goes et al. 2012; Qin et al. 2017], tiling [Ostromoukhov et al. 2004; Wachtel et al. 2014], Wang tiles [Kopf et al. 2006], kernel-density estimation [Fattal 2011], smooth particle hydro-dynamics [Jiang et al. 2015] or electrostatics [Schmaltz et al. 2010]. Mitchell et al. [2018] suggested methods to produce BN point sets in multiple dimensions using spoke-darts, including saturation. All these methods include involved mathematical derivations that are specific for isolated point set properties and do not generalize over higher dimensions, while often incurring significant computational costs. Our approach scales to multiple dimensions for a substantial number of points and for a variety of target spectra, including BN, by simply specifying a loss function.

Spatial statistics. As an alternative to Fourier analysis, we also allow losses using histograms of point distances (the *differential domain*) [Bowers et al. 2010; Wei and Wang 2011] or pair correlation [Öztireli and Gross 2012], which are flexible to analyze non-uniform point patterns. Our formalism is free to mix and match both differential representations and spectra as well as the anisotropic properties. As always, power comes with responsibility, asking the user to choose adequate weighting when combining multiple goals.

Low discrepancy. Deterministic (QMC) approaches emphasize point placement to ensure low discrepancy [Niederreiter 1978]. These methods are specifically tailored for numerical integration. Our architecture allows sample generation for multi-purpose tasks including numerical integration and object placement. We do not directly optimize for low discrepancy but provide comparisons with Sobol [1967], Halton [1960], rank-1 [Niederreiter 1992] and Fibonacci lattices [Nuyens 2013; Sloan and Joe 1994] to show improvements in certain cases.

2.2 Subspaces

Several methods try to explicitly produce patterns with desired spectra also in their projections. Chiu et al. [1994] propose well-stratification in both 1D and 2D. Jarosz et al. [2019] extended these stratification using orthogonal array-based construction [Hedayat et al. 1999] that preserves stratification across multiple projections. Reinert et al. [2016] produce n-D BN patterns which have BN properties in low-dimensional projections as well. These approaches result in a typical cross-like spectrum. Singh and Jarosz [2017] show convergence improvements by aligning these cross-like subspace spectra with the integrands' spectra using shearing. Production renderers [Kulla et al. 2018] also notice improvements when mixed subspaces are properly handled. Joe and Kuo [2008] improve upon Sobol sequences that may have poor 2D projections. Ahmed et al. [2016] suggest producing 2D patterns that have both BN and low-discrepancy properties. Perrier et al. [2018] enhance Sobol 2D projections with BN properties by improving Owen's [1997] scrambling.

Overall, the relation between blue noise, low-discrepancy and variance reduction in MC integration is not fully clear [Christensen et al. 2018; Dobkin et al. 1996; Mitchell 1992; Shirley 1991]. It is evident, however, that there are methods that produce a low error, yet produce a more suspicious artifact pattern and that there are other approaches that produce visually pleasing patterns but a high error [Georgiev and Fajardo 2016].

While we do not look into low-discrepancy explicitly [Doerr and De Rainville 2013], our subspace construction allows to achieve a key property of low-discrepancy patterns: their projections are well-behaved, too. This can manifest as a low variance when using them as sample points in MC integration as shown in our analysis.

2.3 Learning

Computer graphics and in particular filtering recently sees a push towards a learning-based paradigm, where, instead of implementing algorithms from first principles and mathematical derivations, data is used to optimize a general architecture in a learning phase to perform a task in the deployment phase.

In particular for inverse problems, this idea has led to groundbreaking achievements [Krizhevsky et al. 2012]. Typically in these convolutional neural networks (CNNs), optimization is performed over the space of image convolution kernels. The term "deep" refers to the fact that convolutions are linear operations, followed by non-linearities and stacked in a complex pipeline. Stochastic gradient descent (SGD) is able to perform this non-linear optimization even for very deep pipelines made of simple operations. SGD works in batches, where only a stochastic subset of the training data is used. As our training data comprises of infinitely many sets of random points, working in batches is mandatory. We also use a CNN, but require filters more advanced than simple 3×3 masks used in image processing. Early, Fattal et al. [2011] optimized pyramids of convolutions to also solve tasks such as Poisson integration.

A popular methodology is *supervised* learning: Here, images labeled as "cat" or "dog" are provided and a cascade of filters is tuned to produce accurate class labels for each image. Such an approach is regrettably limited by the availability of (human) supervision. In our case, we do not even attempt to build a supervised system that is trained on a list of curated examples patterns but proceed in a *semi-supervised* fashion [Zhou 2017]: we do provide supervision, but in an abstract form that does not enforce an answer, but lists constraints such as spectral quality, that need to hold for a solution.

Rendering is a key application of point patterns, where deep learning has been used for relighting [Ren et al. 2015], screen-space shading [Nalbach et al. 2017], volume rendering [Kallweit et al. 2017], de-noising [Bako et al. 2017; Chaitanya et al. 2017; Kalantari et al. 2015], or improved importance sampling in light transport simulation [Dahm and Keller 2017; Müller et al. 2018; Zheng and Zwicker 2018]. Specifically, the focus of the latter is to build deep models of the indirect light field in a given scene and placing samples where this function is high in an importance-sampling spirit. Note, that this sample placement is a scene-dependent process that is trained per-scene to optimize for variance reduction in the specific integrand solution. While the goals of this work are quite different as training is performed once to find point patterns that generalize across all scenes / integrands, we also demonstrate that learning point patterns optimized for MC integration is feasible.

We make use of learned operations on point clouds, as pioneered by PointNet [Qi et al. 2017]. PointNet and following papers have made use of symmetric functions and rotational transformers in a tunable fully-connected architecture, but without translation-invariance, i. e., using fully connected settings. This might work when sampling a chair into a small number of points in 3D, but not to many points in higher dimensions as our comparison to a fully-connected network will show.

Other work has generalized image convolutions to unstructured 3D [Atzmon et al. 2018; Wang et al. 2018a], e. g., by phrasing the convolution kernel as a neural network [Hermosilla et al. 2018; Wang et al. 2018b], but without scalability to higher-dimensional convolution domains. Our filters are both convolutional and scale to higher dimensions. Our learnable architecture has many similarities with (convolutional) neural networks, such as trainable filters, but also differences, e. g., we work exclusively on point positions and their changes. Instead of inferring labels or per-pixel or per-point attributes such as normals, we optimize for filters that transform sets of random points into sets of points with the desired properties.

Generating samples proportional to a certain target distribution is at the heart of generative modeling [Dinh et al. 2017; Goodfellow et al. 2014; Kingma and Welling 2013; Rezende and Mohamed 2015]. However, none of these consider sample correlation. In our case, the probability is uniform and only the correlation of points is relevant. Yet, distribution and correlation could be treated jointly in future work based on our approach.

2.4 Domain-specific languages

Our system is inspired by the rise of domain specific languages, such as recently proposed for image synthesis [Anderson et al. 2017], high-performance image processing [Li et al. 2018], non-linear image optimization [Devito et al. 2017; Heide et al. 2016] or physics [Bernstein et al. 2016]. Instead of deriving our own parser, we provide a compact suite of functions in TensorFlow [Abadi et al. 2016] that are parsed and evaluated efficiently during training and testing using TensorFlow's symbolic analysis and GPU evaluation.

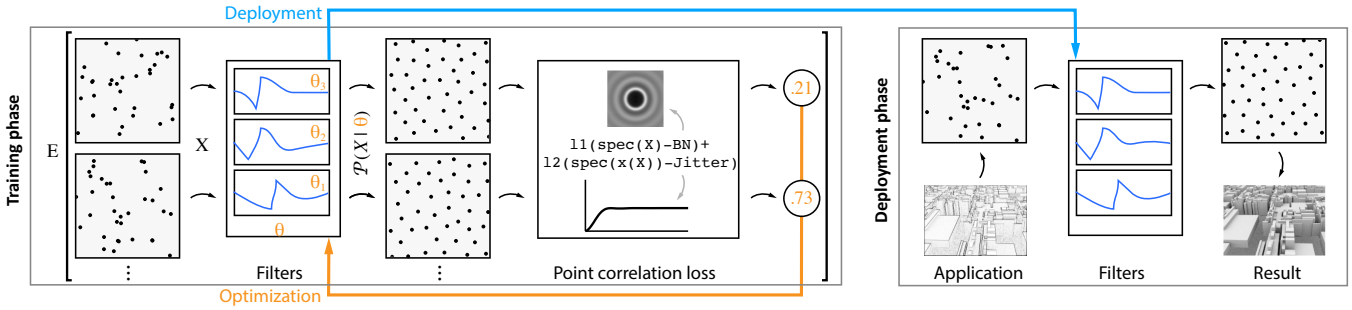

Fig. 2. Our system comprises of two main parts: learning and deployment. At the learning stage **(left)**, a user provides uniform random input point sets (denoted X, two realizations shown) which are then mapped by our pipeline to a user-defined loss in respect to a target (here a combination of a BN and a jittered spectrum). During optimization (orange arrow), filters (blue shapes) with tunable parameters (Θ) perform convolutions to produce a new point set $\mathcal{P}(X|\Theta)$. In the deployment stage **(right)**, these trained kernel weights are used (blue arrow) to produce point patterns with specified correlations in an application (here sampling ambient occlusion). Please see Sec. 3 for more details.

3 OVERVIEW

Our exposition has two main parts (Fig. 2): First, we introduce the notion of a point correlation design loss (Sec. 4) which defines the desired properties. Second, we describe a deep architecture (Sec. 5) that can be optimized in respect to this objective.

A point correlation loss is a functional programming snippet, that maps a point pattern to a scalar value. It is the only user-provided input to our system at training time. Devising operations to compose point correlation losses which allow for high flexibility in defining point patterns is the first key technical contribution of this paper.

We optimize for a point generation method by back-propagating it through a trainable architecture. This architecture is our second main contribution and consists of a deep cascade of weighted distance-based, unstructured filters (Sec. 5). The unstructured representation we choose allows to scale to point sets that are naturally sparse in higher dimensions (>3D) and could not be captured using regular representations commonly used for 2D images or 3D volumes. Our filters map higher-D signals to higher-D signals, but rely on (subspace) distances only. This scales better to higher dimensions, as the notion of distance between a pair of points remains unchanged across spaces with any dimensionality. Common convolution kernels would fail, as they face a combinatorial explosion with at least 2^{10} values per kernel in 10-D. Finally, our filters have compact support, allowing to execute fast, once trained, for substantial point numbers and high dimensions.

Our system is comprised of two stages (left and right in Fig. 2): a learning stage in which the architecture is optimized and a deployment part, in which the result of this optimization is executed to generate new point patterns. The training is supervised only by the loss, that measures point pattern quality and never requires supervision by any point pattern example. Hence, our approach produces patterns which were previously unknown and consequently could not have been input to traditional supervised learning.

4 POINT CORRELATION LOSSES

To design our loss functions, we first recall the formalism of point correlations and the different analysis tools developed over the past decades to study these correlations [Singh et al. 2019a]. Later, we define a range of operators that enable the design of these point correlations. We start by simple spectral and differential domain properties, include linear and non-linear projections and finally discuss the metrics between point correlations.

Note, that the key idea to enable scalability is, that all point correlation operators are only ever computed at training time, never at test time. Training is slow; execution is fast in comparison.

4.1 Notation

A point correlation loss $\mathcal{L}(X) \in \mathbb{R}^{n\times m} \rightarrow \mathbb{R}_{\geq 0}$ maps an unstructured set X of m points in n-dimensional space to a single non-negative scalar, which is smaller for point patterns that are closer to the design goal. The loss can make use of simple operations such as addition, multiplication, etc. to combine our custom operations to be listed next. Losses are simple: all patterns produced in this paper contain only up to three terms. The design of point patterns for a specific MC integration task is even easier, as users only need to provide access to a method to sample integrand functions (Sec. 6.3).

These operations enable losses on unstructured point correlations, which are modular in their design, where the scope of each loss component can be strictly limited to selected dimensions in the point distribution. Appendix B lists all symbols used in this work.

4.2 Point correlation

We define the point correlation (Fig. 3) of a point set X as

$$P_\kappa(X)(\mathbf{q}) = \mathbb{E}_{\mathbf{x}_1\sim X}[\mathbb{E}_{\mathbf{x}_2\sim X}[\kappa(\mathbf{x}_1 \ominus \mathbf{x}_2, \mathbf{q})]], \tag{1}$$

where $\mathbb{E}_{\mathbf{x}\sim X}$ is the expected value of the random variable X, κ is a *kernel* working on *offsets* between pairs of points we denote as $\mathbf{d} = \mathbf{x}_1 \ominus \mathbf{x}_2$, $\ominus$ is the toroidal vector difference (wrap-around in the unit hyper-cube) and $\mathbf{q}$ is what we will call the *correlation* coordinate.

The notion of P_κ is a two-fold abstraction. First, it generalizes along one axis over Fourier spectra [Zhou et al. 2012] and the differential domain [Wei and Wang 2011]. Along a second axis, generalization is folded into κ, which can work both on linear high-D offset and on non-linear 1D distance. $P_\kappa(X)(\mathbf{q})$ is a distribution across the correlation coordinate $\mathbf{q}$, which is again an abstraction of frequency or distance of points. The abstract notion is made concrete in an application by the choice of correlation kernel κ. This kernel puts $\mathbf{d}$

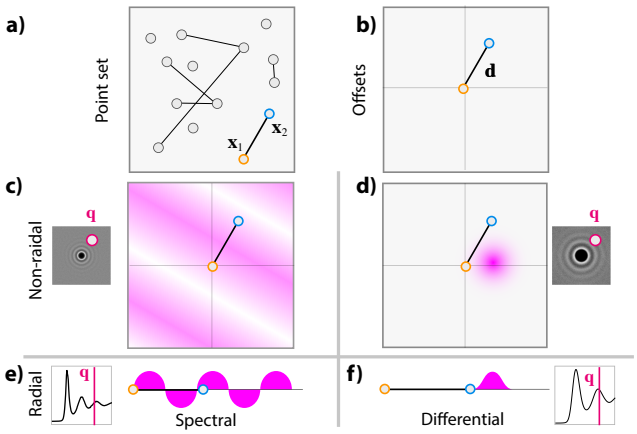

Fig. 3. Point correlation (Eq. 1), here shown for a 2D example, is defined as an expected value over all pairs of points from the point set X **(a)**. We here show some pairs as links. For each pair the offsets are produced by subtracting the first point **(b)**, here shown for a single pair $\mathbf{x}_1$ and $\mathbf{x}_2$ (thick line). This offset $\mathbf{d}$ is then given to a function κ that can work in different ways, out of which we illustrate four **(c-f)**. The second row **(c,d)** works in 2D and correlates with the 2D Fourier basis (first column **(c)**, "Spectral") or the Gaussian basis (second column **(d)**, "Differential") at all 2D correlation coordinates $\mathbf{q}$ (one sinusoidal/Gaussian instance shown in magenta). The third row **(e,f)** proceeds the same, just that the correlation coordinate $\mathbf{q}$ is a single scalar distance.

and $\mathbf{q}$ in different relations: it can work on distances or offsets as well as it can apply non-linearities such as complex exponentiation followed by a norm.

As $P_\kappa(X)(\mathbf{q})$ is an expected value, it could be evaluated using quadrature or estimated using MC. We will however use specifically optimized implementations for specific κ.

4.3 Spectrum

The spectrum of a point set X is computed using

$$\mathrm{spec}(X) = P_\kappa(X) \qquad \text{with} \qquad \kappa(\mathbf{d}, \mathbf{q}) = \cos(2\pi \langle \mathbf{d}, \mathbf{q} \rangle)$$

and maps the n-dimensional sparse point pattern to an n-dimensional spectrum. In this case $\mathbf{q}$ is an n-dimensional frequency. The resolution n_c of the spectrum can be configured by the user, but is typically chosen to be a multiple of $m^{1/n}$.

This operation is commonly used in conjuction with a norm such as $\mathcal{L}_1$ to measure the difference to a reference spectrum. The point correlation loss

$$\mathcal{L}(X) = \mathtt{l1}(\mathtt{spec}(X) - \mathtt{BNOT})$$

for example computes the spectrum of a pattern X and compares it to a reference blue noise spectrum generated through optimal transport (BNOT) [De Goes et al. 2012].

As the DC term (frequency 0) amounts to be the number of points, and we would like to train independent of that number, we decided to remove it from the spectrum i. e., set it to zero as in $P(X)(\mathbf{0}) := 0$.

The power spectrum (Fourier coefficient's magnitude square value):

$$\mathrm{spec}(X)(\mathbf{q}) = \left| \frac{1}{m} \sum_{k=1}^{m} \exp(2\pi \mathrm{i} \langle \mathbf{q}, \mathbf{x}_k \rangle) \right|^2, \tag{2}$$

is best computed in time linear in the number of samples.

4.4 Differential domain

The function

$$\mathrm{dDom}(X) = P_\kappa(X) \qquad \text{with} \qquad \kappa(\mathbf{d}, \mathbf{q}) = \mathcal{N}(\mathbf{q} - \mathbf{d})$$

maps the sparse point pattern to a dense distribution of offsets, where $\mathcal{N}$ is a zero-mean Gaussian with a standard deviation of $2/n_c$.; effectively, an n_c-bin n-D histogram. Now, $\mathbf{q}$ is an n-dimensional differential coordinate. It is used similar to the spectrum e. g.,

$$\mathcal{L}(X) = \mathtt{l2}(\mathtt{dDom}(X) - \mathtt{Jitter})$$

would ask for a power histogram that is $\mathcal{L}_2$-similar to the histogram `Jitter` of jittered sampling.

The number of bins n_c could be chosen optimal in respect to spectrum and point count [Scott 1979]. For our targets, however, more than 128 bins did not improve result quality. As the computational overhead of an increased number of bins is small, we retained this conservative configuration in all our experiments.

For computation, we iterate all pairs of points, compute their distance, and scatter them to the respective bins around $\mathbf{q}$ with Gaussian weights instead of iterating all pairs for all values of $\mathbf{q}$. Soft bins make this process differentiable.

Wei and Wang [2011] observed that limiting the interaction to pairs of only nearby points can produce high-quality results, with observed (though not guaranteed) linear time complexity. For this kind of range search to work efficiently for high numbers of points, a spatial acceleration structure [Agarwal et al. 1999] is needed, which we regard as future work.

4.5 Radial mean

Besides producing an n-D spectrum or histogram from n-D points we also support working on the radially-averaged spectrum

$$\mathtt{radSpec}(X)(\rho) = \mathbb{E}_{\mathbf{q} \sim \Omega} \mathtt{spec}(X)(\rho \cdot \mathbf{q})$$

and the radially-averaged histogram

$$\mathtt{radDDom}(X)(\rho) = \mathbb{E}_{\mathbf{q} \sim \Omega} \mathtt{dDom}(X)(\rho \cdot \mathbf{q}),$$

where ρ is a radius and Ω is the n-dimensional unit hyper-sphere.

Monte Carlo estimate. A trivial construction of `radSpec` would not operate on the point set X, but on $\mathtt{spec}(X)$. This indeed would be more modular, but regrettably does not scale well to multiple dimensions like $n = 10$, where a regularly sampled spectrum would require prohibitive amounts in the order of $O(n_c^n)$ of memory. Note, that the output of the radially averaged spectrum is always a compact 1D function of radius, so it requires only $O(1)$ memory.

Addressing this difficulty we suggest estimating the spectrum using Monte Carlo integration as follows: First, we randomly generate $\mathbf{q}$ locations–that represent frequencies—on the hypersphere Ω using the method of Hicks and Wheeling [1959] and scale them by ρ. Since only integer frequencies matter for unit sampling domains [Singh et al. 2019b], we then snap $\mathbf{q}$ to the integer grid. Finally, we evaluate

$P(\rho \cdot \mathbf{q})$ as described above and average. For low-dimensional point correlations (in practice $n \leq 2$), we evaluate all $\mathbf{q}$ exhaustively in a grid using quadrature. We will analyze the convergence of this estimate in Sec. 6.5.

4.6 Swizzle

As we are interested in the properties of subspaces, we make use of the typical swizzle operations to extract the dimensions of our interest, denoted as $\mathsf{x}(X)$ for the x component, $\mathsf{y}(X)$ for the y component, $\mathsf{xy}(X)$ for the xy component, etc.

4.7 Metrics

Typical metrics to compare n-D spectra and histograms (distributions) are $\mathcal{L}_1, \mathcal{L}_2$, which we denote as $\mathsf{l}1, \mathsf{l}2$. In our experiments, we mostly use $\mathcal{L}_1$ thanks to its robustness. It is commonly achievable in deep learning, but harder to achieve in mathematical derivations, that often revert to $\mathcal{L}_2$.

5 POINT PATTERNS VIA ITERATED FILTERING

Here we introduce a deep trainable architecture for generating point patterns. The idea is to use a deep pipeline (Sec. 5.1) of learnable filters (Sec. 5.2) that convert a high number of random high-D points into correlated points as defined in Sec. 4.

5.1 Architecture

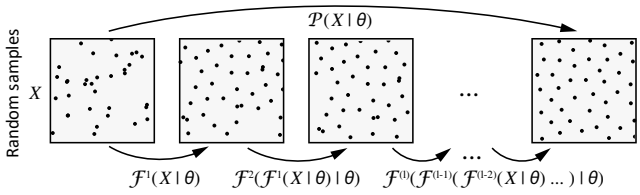

Fig. 4. Illustration of the forward pipeline ($\mathcal{P}$) of our network architecture for $m = 32$ denoted as X in $n = 2$. From left to right: The uniform random input points (X) are processed using a cascade of filters $\mathcal{F}$ to produce the intermediate results $\mathcal{F}(X)$ and $\mathcal{F}(\mathcal{F}(X))$. After several iterations (dots) the final outcome, on the right, has the desired BN spectrum.

We formalize this as learnable mapping $\mathcal{P}_{\mathcal{L}}(X|\theta)$ from a set of m uniformly random n-D input points X without any specific properties, into a set of m output points $X_{\text{Out}} = \mathcal{P}(X|\theta)$ with specific properties according to the loss $\mathcal{L}$. The mapping $\mathcal{P}$ has tunable parameters θ, for which we learn,

$$\theta = \arg\min_{\theta'} \mathrm{E}_{X \sim \mathcal{U}^{m \times n}}[\mathcal{L}(\mathcal{P}(X|\theta'))],$$

where $\mathcal{U}$ is a uniform random distribution. We choose to implement $\mathcal{P}$ as a cascade

$$\mathcal{P} = \mathcal{F}^{(l-1)}(\mathcal{F}^{(l-2)}(\ldots \mathcal{F}^{(1)}(X|\theta)|\theta)|\theta)$$

of l unstructured filters $\mathcal{F}_i$, with learnable parameters θ (Fig. 4). These filters simply map a point set of a certain size and dimension to another point set of the same size and dimensions and will be described next.

5.2 Filters

We will first introduce distance-based filters (Sec. 5.2.1), limited to isotropic losses, before we generalize them to subspace filters (Sec. 5.2.2) that can be used for anisotropic designs.

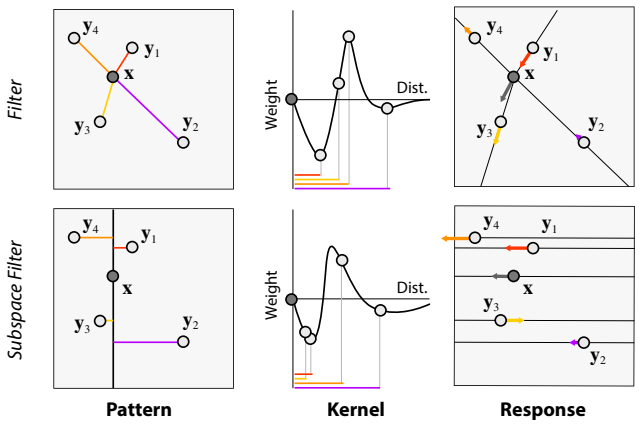

Fig. 5. Our tunable filters, based on full-dimensional distance **(top)** and based on subspace distances **(bottom)**. The first column shows a point $\mathbf{x}$ in 2D and all other pattern points. Colored lines indicate difference between other points and the center point. The second column shows the filter kernel (x axis is distance, y axis is weight). The four color-coded distances are used to look up the kernel weight. The third column shows how the offset vectors between $\mathbf{x}$ and all other points are scaled by the kernel response, and added, resulting in a single dark-grey correction vector that moves $\mathbf{x}$. While the first row uses common distances, the second row uses x-subspace distances.

5.2.1 Distance-based Filters. A first solution is to work on one-dimensional distance, i. e., with radially symmetric kernels, as shown in the top row of Fig. 5. These are simple enough to be represented using a 1D table θ of weights with b entries θ_i.

Let a kernel g, parametrized by a b-D weight vector θ be

$$g(d|\theta) = (1 - \mathrm{frac}(\hat{d})) \cdot \theta_{\lfloor \hat{d} \rfloor} + \mathrm{frac}(\hat{d}) \cdot \theta_{\lceil \hat{d} \rceil} \quad \text{where} \quad \hat{d} = d \cdot b / r.$$

Now filtering $\mathcal{F}'$ with the kernel g is defined as

$$\mathcal{F}'(\mathbf{x}|\theta)_i = \mathbf{x}_i + \sum_{j \neq i}^{m} g(||\mathbf{x}_j \ominus \mathbf{x}_i||_2|\theta) \frac{\mathbf{x}_j \ominus \mathbf{x}_i}{||\mathbf{x}_j \ominus \mathbf{x}_i||_2}, \tag{3}$$

where $\ominus$ denotes the toroidal vector difference. In a slight abuse of notation, we will refer to the (overloaded) filtering of all points X as $\mathcal{F}(X)$ as well. See appendix A for a derivation of the gradients required for more efficient learning.

To avoid computing interactions between all points, which would imply quadratic time complexity, we limit the filters to a constantly-sized neighborhood of a *receptive field* r that is typically chosen to be only a fraction of the domain.

Reading the kernel table continuously at distance $d \in (0, r)$ using linear sampling, makes it differentiable. We always learn a residual to adjust the point position, which improves gradient flow.

Discussion. Please note, that the weights θ can – and also need to be – negative. Positive weights attract $\mathbf{x}_i$ into the direction of $\mathbf{x}_j$, negative weights repel. This is shown by the positive and negative weights in the column "Kernel" in Fig. 5.

Please also note that the number of filter kernel bins only adds to the number of trainable parameters, but does not affect the speed at deployment time: It is a $O(1)$ table look-up. This formulation is similar to the one of Zhou et al. [2012] and Heck et al. [2013], who also update point positions with new positions in an optimization. The "Response" column in Fig. 5 (top row) visualizes this: the magenta and the red arrow point towards $\mathbf{x}$ (repulsion), the orange and the yellow one point away (attraction). The sum of all these is applied to $\mathbf{x}$ (black arrow). We do not perform an optimization for a point set X at run-time, but optimize over all methods that are quick to execute on a GPU at deployment instead.

5.2.2 Subspace filters. We found the distance-based unstructured filters to work well, if the loss does not ask for anisotropic effects: When requiring an elliptical spectrum, a filter based only on distances has no way of disambiguating if a distance of .1 was along the x or y direction. In other words the point correlation loss can ask for more than what the architecture can achieve.

As a solution, it is tempting to use full-dimensional filters, but a kernel would now need to learn up to all 10-D interactions, resulting in b^{10} learnable parameters, so at least $3^{10} = 49,304$ parameters for each filter, i. e., way too much to learn or even execute efficiently.

As a middle ground, we opt to learn combinations of separable filters. The weighted distance between two points $\mathbf{x}_1$ and $\mathbf{x}_2$ is $||\mathsf{M}(\mathbf{x}_1 \ominus \mathbf{x}_2)||_2 \in \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$ where M is a diagonal binary weight matrix. The common distance is a special case of an identity weighting $\mathsf{M} = \mathbf{1}$. The subspace filter is now

$$\mathcal{F}(\mathbf{x}|\theta)_i = \mathbf{x}_i + \sum_{j \neq i}^{m} g(||\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)||_2|\theta) \frac{\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)}{||\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)||_2}. \quad (4)$$

By choosing different weight matrices M, different subspaces can be addressed individually, respectively others can be ignored, by setting a column to zero. In practice, we encode this by simply listing the non-zero elements, as done in the swizzle operation. Please note that the weight matrix M is input to our method. Since the number of subspaces s grows exponentially with the dimensionality, the user has to choose relevant subspaces based on domain-knowledge.

5.2.3 Kernel interpolation. A deep pipeline of many filters might introduce a high number of trainable variables in the order of $b \cdot l$. While several iterations (typically, $l > 20$) are required to get the pattern right and several bins are required to resolve accurate response to distances (typically, $b = 128$), what needs to be done might not need to be very different in each iteration.

We capitalize on this observation as follows: We do not learn distinct kernels $g^{(1)}, \ldots, g^{(l)}$ for each iteration, but a continuous two-dimensional kernel space h to be used in the i-th kernel as $g^{(i)} = h(d, i/l_\mathrm{u}|\theta)$, where $l_\mathrm{u} \ll l$ is the number of *unique* kernels. An example is shown in Fig. 6. Here, one axis is the distance and the second axis is iteration. We choose h to be piecewise bilinear which also is differentiable. This allows drastically reducing the number of tunable parameters, e. g., using $l = 32$ iterations with only $l_\mathrm{u} = 8$ unique kernels.

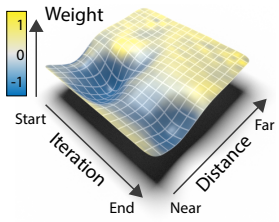

Fig. 6. Kernel values as a height field, depending on iteration **(left)** and distance **(right)**.

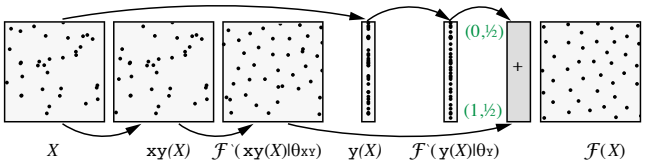

Fig. 7. To optimize samples across subspace y and full space xy, filters in y need to work jointly with the filters in xy. The outcomes of the filters are combined using a normalized sum (green weights; here, x is taken from the xy-filter, and y is taken to be the average y outcome of the y- and the xy-filter) to obtain filters with prescribed properties in each subspace.

Combination. The filtering described in Sec. 5.2.2 can learn how to apply filters in subspaces, but to be useful, multiple subspaces (e. g., x and y) need to work together, as well as jointly with the space they are a subspace of (xy). To this end, the outcome of multiple subspaces needs to be combined. In particular, this combination has to be done after each layer. The necessity to do so is seen when considering a loss optimizing y and xy jointly (Fig. 7): after each adjustment in each dimension, both outcomes affect the other; they need to share information and have to produce one joint xy result, not two disparate y and xy results.

We achieve this by a normalized sum of subspaces selected using M. The normalization divides each element in that vector by the number of occurrences in all subspaces. A point correlation loss that e. g., asks for xy and y would add the two, but divide y by 2 (green numbers in Fig. 7).

5.2.4 Gridding. Our method can operate in a gridded and ungridded mode. The ungridded is the default and used for general point patterns. The gridded variant holds some dimensions fixed.

In a *non-gridded* point pattern, all dimensions are filtered as processed and $\mathcal{P}$ maps from n to n dimensions.

A point pattern is *gridded* when it maps from n_In input to n_Out output dimensions. Here, the first $n_\mathrm{In} - n_\mathrm{out}$ dimensions are initialized using a regular grid. These values are left unchanged by the filter pipeline. However, all n_In values are used to compute distances.

An example of a gridded point pattern with $n_\mathrm{In} = 3, n_\mathrm{Out} = 1$ is a pattern where the first two fixed dimensions are the pixel centers, and the third dimension is the wavelength for spectral rendering. In other words, a gridded pattern is a height field: x and y are implicit and fixed, and the height z changes. Our method can now learn a $\mathcal{P}$ that arranges the wavelength such that they are BN in respect to other nearby wavelengths at fixed spatial positions.

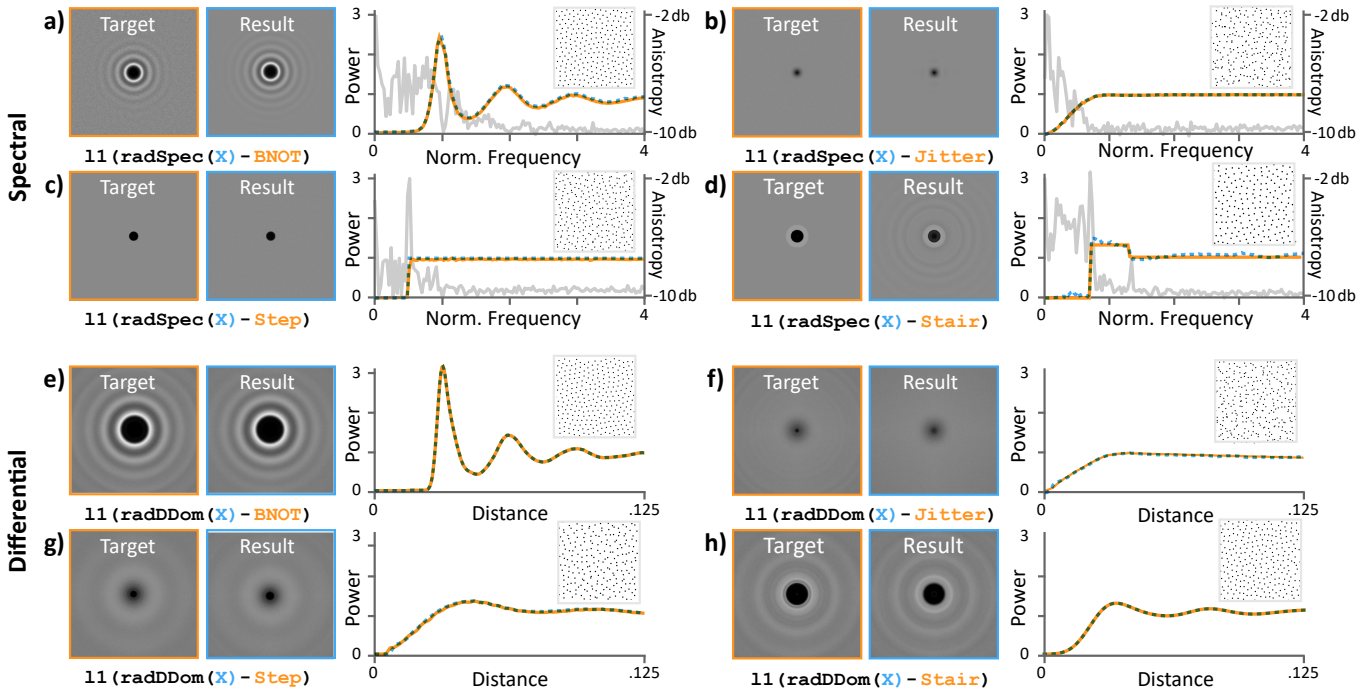

Fig. 8. Results of our approach for different targets and different losses. Each sub-figure shows the program snippet of the loss used, the point pattern produced by our approach as well as the 2D and radially-averaged power spectra reference (orange) respectively, our result (blue, dashed in 1D). The grey curves refer to the radial variance (aka. anisotropy) of the Fourier power spectrum. Experiments **a**-**d** use the spectral, **e**-**h** the differential domain. We see, that our method faithfully reproduces state-of-the-art 2D jittered and blue noise samplers like Step [Heck et al. 2013] and Stair-case [Kailkhura et al. 2016].

5.3 Training

Optimization. The filter weights θ are the only learnable parameters of our system. Training is using TensorFlow's SGD employing the the ADAM optimizer [Kingma and Ba 2014], with an exponentially decreasing learning rate initialized by 10^{-7}. Learning typically requires around 8 hours on an Nvidia Tesla V100-PCIE with 32 GB memory and 1.38 GHz memory clock rate.

Variance reduction. Individual point set realizations give rise to noisy estimates of point correlations and, consequently, noisy gradients. For this reason we average the correlations of multiple realizations to reduce their variance. In particular for our Fourier design, we found the use of this unbiased estimate essential for convergence: In each training iteration, multiple output spectra of our system are averaged before they are evaluated within the loss, e. g., by comparing them to a target. This stabilizes training, especially for small point counts m. Crucially, the designer does not have the freedom to combine spectra of different realizations arbitrarily: Spectra are always averaged. Please notice that this procedure is subtly different from mini-batching, as detailed in the supplemental Sec. 2. In a slight abuse of terminology, we nevertheless refer to the point set realizations used for variance reduction as a mini-batch. We observe that a mini-batch size of 4 point sets – followed by radial averaging – leads to spectra that are converged enough to be used for gradient computations. We evaluate this choice in Fig. 18 and Sec. 6.5.

6 RESULTS

Here we perform a quantitative analysis for our approach in terms of spectral and differential properties (Sec. 6.1), look into discrepancy and error convergence (Sec. 6.2) as well as learning samplers directly from integrand spectra (Sec. 6.3). We instrument different aspects of scalability, parameter choices (Sec. 6.4), stability (Sec. 6.5), and metrics (Sec. 6.6), compare to a fully-connected design (Sec. 6.7), before we finally show applications to rendering, artistic stippling and object placement (Sec. 6.8).

Default parameters. Unless said otherwise, this section uses $m = 1024$ points, $n_c = 128$ pixels / bins for full-dimensional and, $n_a = 4m^{\frac{1}{n}}$ for radially averaged spectra or histograms, an architecture with a depth of $l = 60$ filters and 20 unique kernels where each filter has a kernel with $b = 96$ elements which span the full receptive field i. e., $r = .5$.

Presentation. Spectra or histograms shown are averaged across 100 realizations. Our radially averaged spectral plots are normalized horizontally by $m^{-1/n}$ to support comparison across different numbers of points m and dimensions n and cropped to the range $[0, 4]$. We call this *normalized* frequency.

6.1 Spectral and differential analysis

6.1.1 Isotropic 2D. We start by showing results from learning several state-of-the-art isotropic 2D patterns in Fig. 8. All losses here minimize an l1 difference to radially averaged reference spectra.

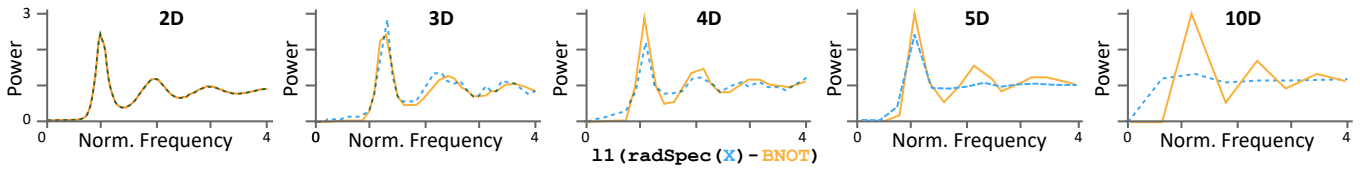

Fig. 9. Radially averaged power spectra for our method (dotted blue) and a reference (orange) for increasing dimensions from 2D to 10D (left to right). Results of our method match the target closely up to five dimensions, especially in the low frequency region (0, 1). With increase in dimensionality, the spectrum shrinks exponentially towards low frequencies which makes the spectrum looks increasingly piecewise-linear in the normalized range (0, 4). While at 5D the main features are present, at 10D, our results have deteriorated, compared to lower dimension like 2D.

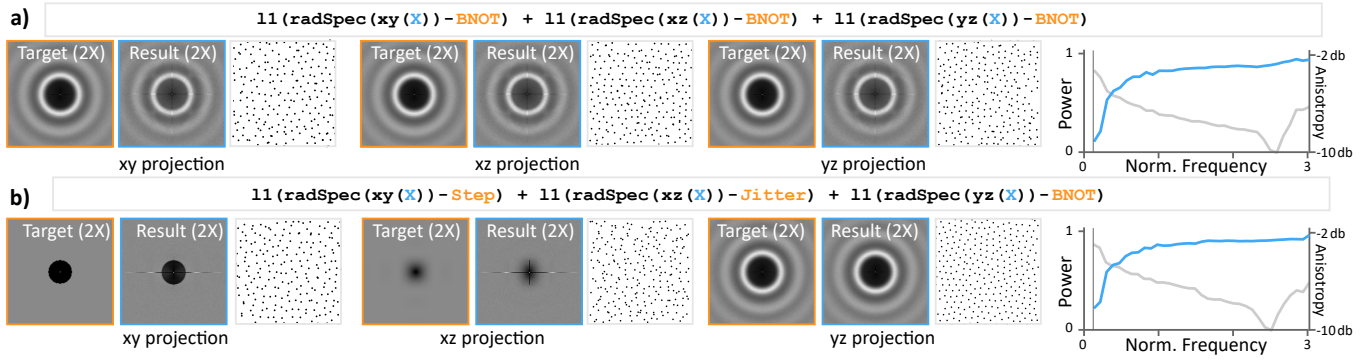

Fig. 10. Our method can handle anisotropic patterns with different properties in different subspaces. Here, we show different 2D subspaces (of a 3D point pattern) that are optimized for different target spectra. In the top row, each 2D subspace is given a same target spectrum (BNOT [De Goes et al. 2012]). In the bottom row, we provide Step [Heck et al. 2013], jittered, and BNOT target spectra for each 2D subspace respectively. The resulting spectra (in blue) closely match the target spectra (in orange). The rightmost plots show the radially averaged spectrum and the corresponding radial variance (anisotropy).

We see, that our approach can produce four relevant methods (BNOT [De Goes et al. 2012], jitter [Cook 1986], Step BN [Heck et al. 2013] and Stair BN [Kailkhura et al. 2016] in Fig. 8, a-d) as the radially averaged spectral profiles (orange and dotted blue) match the reference closely, in particular, in the low-frequency regions. We also see that the 2D spectrum matches the reference while our learning was only supervised to produce a radial average. This shows that no additional anisotropy was introduced, further supported by the radial variance plots (grey). Fig. 8, e–h, repeats the above experiment in the differential domain, where the target is the 1D point correlation function. Again both 1D and 2D PCFs match.

6.1.2 Multi-dimensional isotropic. We explore the previous analysis for $n > 2$ in Fig. 9, for one specific important pattern, BNOT [De Goes et al. 2012]. Target spectra for higher dimensions were produced by scaling [Heck et al. 2013; Pilleboue et al. 2015] the two dimensional BNOT power spectrum computed over 1M samples. Thanks to our MC-based formulation (Sec. 4.5) that avoids reconstructing the multi-dimensional spectrum, our system is able to perform such multi-dimensional spectra optimization.

We note that the results match the target, but this gets increasingly difficult in higher dimensions. While at 5D, the BN peak is still distinct with a zero region and a peak at 1.0, at 10D the spectrum doesn't improve. This is one of the limitations of our current architecture. Our method, however, allows BN in subspaces as well, as explored next (see supplemental Fig. 3 for subspace spectra in 10D). The only other recent work in multi-dimensional blue noise is Spoke Darts [Mitchell et al. 2018], but without any control over subspaces, at much higher algorithmic complexity (no code is changed to work in high dimensions for us) and compute cost (we compute this pattern in 150 ms).

6.1.3 Subspace results. Taking it a step further, we now analyze losses that ask for different spectra in different subspaces [Ahmed et al. 2016; Chiu et al. 1994; Reinert et al. 2016]. Our exemplary analysis is in $n = 3$ dimensions where three canonical 2-D subspaces, xy, xz and yz, exist. The loss now sums three $\mathcal{L}_1$ losses in three subspaces in respect to three references. Results are shown in Fig. 10.

We see that our approach manages to produce a pattern that has the desired spectra in all projections Fig. 10, a. Note, that this would not be the case for a 3D BNOT pattern that is unaware of subspaces.

Furthermore, we can ask for different targets in different subspaces (Fig. 10, b). Overall, each subspace achieves the desired target spectra, but with slight mutual concessions to be made: We see, that the xy projection that seeks to produce a Step BN spectrum shares the y projection with BNOT which is reflected in the long horizontal anisotropic line in our xy / Step BN spectrum. Along the vertical axis of xy, the anisotropy is due to the shared x projection with Jitter. Jitter (x, z) shares the z projection with BNOT. This manifests as the long horizontal anisotropic line in our Jitter (x, z) spectrum. Along the vertical axis of Jitter, the anisotropy is due to the shared x projection with Step BN.

We also show the radial average and the radial variance for both patterns (Fig. 10, a & b). The radial average looks like a modified

jittered pattern for both: a small ramp and a constant range. As expected, for an anisotropic pattern, the variance is high.

Previous work has so far, by-construction, been only able to address special cases such as jitter in multiple spaces [Chiu et al. 1994; Jarosz et al. 2019], BN along canonical projections [Reinert et al. 2016] or combinations with low discrepancy patterns in 2D [Ahmed et al. 2016]. Our work combines arbitrary spectra and does so in multiple dimensions.

6.2 Error Analysis

In this section, we look at an important application of point patterns: their use as (Quasi) Monte Carlo sampling. We first look into the discrepancy and second analyze the variance for different m.

Discrepancy. Our approach does not incentivise discrepancy as defined using classic metrics [Doerr and De Rainville 2013]. Nonetheless, our patterns can have competitive discrepancy, as shown in Fig. 11, where we compute the box discrepancy [Niederreiter 1992] (100 k samples) of several common patterns in 2D and 3D, as well as for some of ours. Common methods are random, jittered, Sobol [1967]. Ours are BNOT and Step BN as used in Fig. 8.

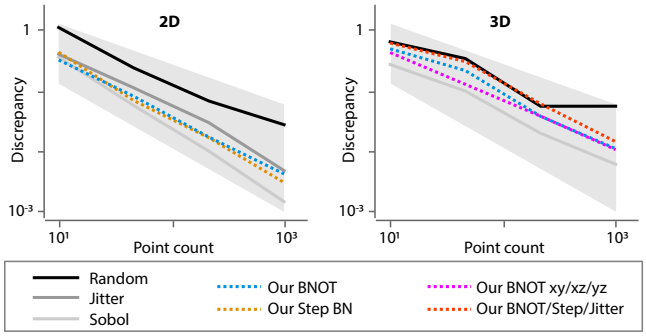

Fig. 11. Discrepancy analysis for a 2D (left) and a 3D (right) integration problem. In each plot, the vertical axis shows discrepancy (less is better) and the horizontal axis shows the point (sample) count. Different methods are shown in different colors. The grey-shaded wedges bound the theoretic limits. We see, that our methods (dotted blue and orange) achieve lower discrepancy than random and jittered samplers (black and medium grey) without having been provided any explicit supervision on discrepancy. We also see, how optimizing in subspaces can improve the result further (red and pink). Finally, low-discrepancy sequences like Sobol (light grey), provide a superior discrepancy by construction.

For 2D, we see, the expected relations between common methods are present. We also see, that the discrepancy of BNOT and Step BN (dotted lines in Fig. 11, 2D) is performing better than jitter, with a discrepancy closer to Sobol, which performs best. Discrepancy is lower as a concession to spectral properties, which dominate MC convergence, as demonstrated next. However, Sobol is a special solution for a special problem (integration) while we provide a general system for many tasks (object placement, stippling, etc.).

Variance. We conclude our analysis by demonstrating our samples in a rendering setting (Fig. 12). To this end, different scenes are considered, e. g., integrands in 2D (a simple disk and ambient occlusion) and 3D (pixel space and motion blur). All scenes are rendered with direct illumination using PBRT-v3 [Pharr et al. 2016]. We compare our BNOT and Step BN variants to rank-1 [Dammertz and Keller 2008; Niederreiter 1992], Fibonacci [Nuyens 2013; Sloan and Joe 1994], Halton [1960] and Sobol [Sobol 1967] samplers. For Sobol and rank-1 lattice, we use generating vectors provided by Frances Kuo [2007]. In our 2D experiments, the Fibonacci lattice generating vector use $\{1, F_{j-1}\}$ for $m = F_j$ samples for a given j-th Fibonacci sequence number. Note, that we compare point sets against sequences which may have better convergence rates for integration than sequences. The data plots in Fig. 12 were generated by averaging variance across 10 random pixels over the image. Each pixel was estimated over 200 realizations. To allow comparison across instances of the pixels, we computed relative variance: variance of the estimates divided by their squared mean. A low value is better, i. e., would render less noisy images.

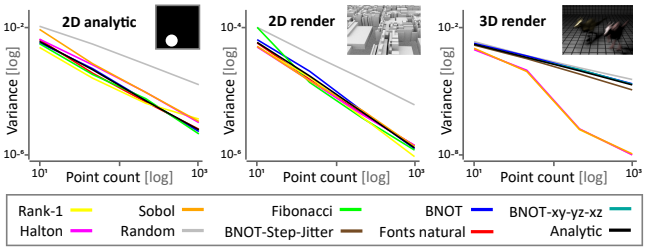

Fig. 12. Variance convergence for different sampling patterns (colors) at different integration problems ranging (left to right) from 2D analytic over 2D rendering to 3D rendering. In each sub-figure, the horizontal axis is sample count and the vertical axis is variance for a single image pixel i. e., image noise (less is better). Our method performs quite well for 2D integration, but requires more explicit loss functions in 3D and beyond to capture correlations across multiple subspaces.

In 2D, for a simple disk integrand, our learned methods show less variance compared to the well-known low discrepancy samplers (rank-1, Fibonacci, Halton and Sobol), whereas, with complex 2D visibility function our method gets competitive across different point count. We also compare it against our novel sampling patterns optimized directly for a given class of functions: Fonts spectra in red, functions having $m^{-1/n}$ radial spectrum fall-off [Brandolini et al. 2001] in black. More details are in Sec. 6.3.

In 3D, rank-1, Halton and Sobol are superior to our sampler.

We further observe that performing a full-dimensional optimization (blue curve in Fig. 12) for MC integration tasks is less rewarding than directly optimizing the lower (2D) subspaces (BNOT-xy-yz-xz and BNOT-Step-Jitter curves). Especially, a combination including Step, BNOT and jittered performs much better. See supplemental Fig. 4 for improvements.

This indicates that there is no universal solution for all tasks. Future work needs to consider other aspects (e. g., discrepancy) with Fourier statistics to better optimize the loss function. In rendering, since the variance convergence is mostly driven by high-dimensional integrands, improving scalability is a promising future avenue.

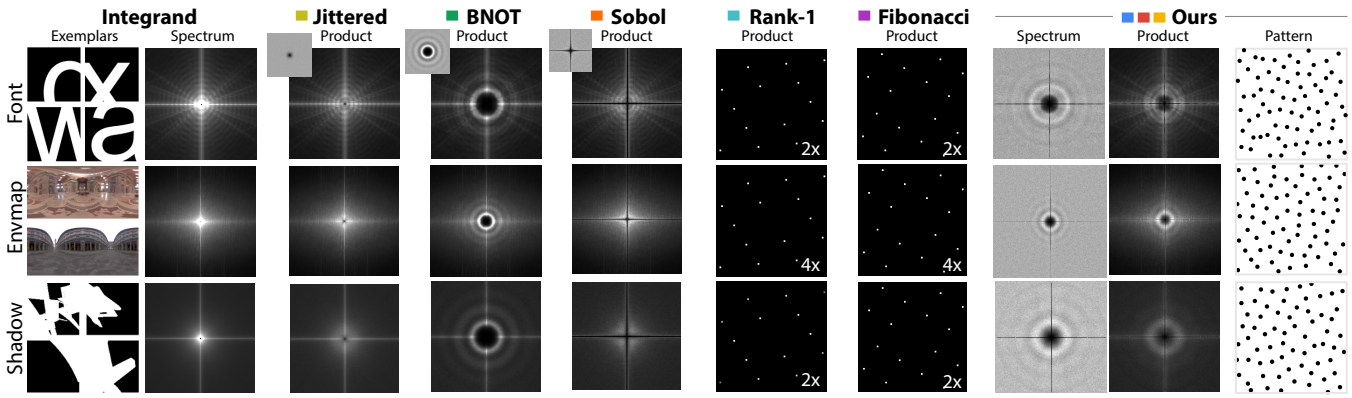

Fig. 13. Learning to sample from integrand statistics. Each row is a different integrand, i. e., graphics application: font super-sampling, natural illumination, and soft shadows. Each row shows: first, some examples from the distribution of integrands; second, the spectra product of the integrand with typical patterns including ours following Eq. 5. Finally, a single realization of our pattern is shown. The spectra used in the middle-row are 2x higher resolution than top and bottom rows. The spectra of Rank-1 and Fibonacci are magnified to aid visibility. Please see the supplemental Fig. 1 for full-sized spectra.

In summary, different samplers, including ours, show different strength for different integration problems. Seeing this as an opportunity, we next show how our framework can be used to optimize directly for a specific function class, resulting in novel patterns.

6.3 Learning Sampling MC from Integrand Statistics

Integrands can be expected to have a certain spectrum, such as the quadratic decay in natural images [Simoncelli and Olshausen 2001], the $m^{-1/n}$ fall-off for hyper-spherical caps [Brandolini et al. 2001] or the one of spherical natural illumination [Dror et al. 2001]. We show how any such spectrum can be used directly as a target of our design allowing to learn novel point patterns optimized for MC integration of a user-provided function class.

All the user needs to provide is the representative power spectrum of the chosen class of functions. Following this, estimating the integral of a class of functions with a power spectrum $\mathbf{F}$ using a *stationary* sampling pattern with an expected power spectrum $\mathbb{E}[\mathbf{S}]$ would give the variance [Singh et al. 2019a,b]

$$\sigma^2 = \sum_{q \in \mathbb{Z}} \mathbf{F}(\mathbf{q}) \cdot \mathbb{E}[\mathbf{S}(\mathbf{q})], \tag{5}$$

i. e., the sum over the product of the integrand and samples' power spectra at all *non-zero integer* frequencies $\mathbf{q}$. Similar derivations based on Campbell's theorem were made for the PCF by Öztireli [2016].

Intuitively, this means sampler spectra should have low response where the integrand spectrum has high values and vice versa, as this minimizes the product and ultimately the area-under-the-curve. In Fig. 14, we see a grey integrand spectrum $\mathbf{F}$ as well as a blue and an orange sampler spectrum $\mathbf{S}_{s,1}$ and $\mathbf{S}_{s,2}$. The estimator variance is equal to the area under the curve (shown solid, or as circles with area a_1 and a_2).

As many natural signals decay in amplitude with increasing frequency [Mandelbrot 1983], most sampling patterns try to be free of low frequencies i. e., to be "blue". While this is a good average strategy, specific integrands might differ substantially in the particular shape of this decay. Our method allows designing purpose-fitted sampling patterns for specific decay.

To this end, we first randomly select exemplars from the space of integrands. The first column in Fig. 13 shows representative examples of this. Second, a MC estimate of the integrand spectrum $\mathbf{F}$ is computed by averaging the power spectra of all these examples. Finally, we use this $\mathbf{F}$ in a loss $1(\text{spec}(X) \circ \mathbf{F})$, where $\circ$ is the Hadamard product. Please note, how we use full-dimensional (here 2D) spectra instead of radially averaged ones here, as the integrand spectra show marked anisotropy. Notably, this procedure occurs before training as an additional stage of preprocessing and does not affect runtime execution efficiency for sampling of new integrands.

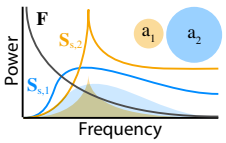

Fig. 14. Given an integrand spectrum (black plot) we here compare two sample spectra (orange and blue plot). A sample spectrum that negates the integrand spectrum (orange) has a lower area under the curve (visualized as an orange circle) than the blue one and hence, less error.

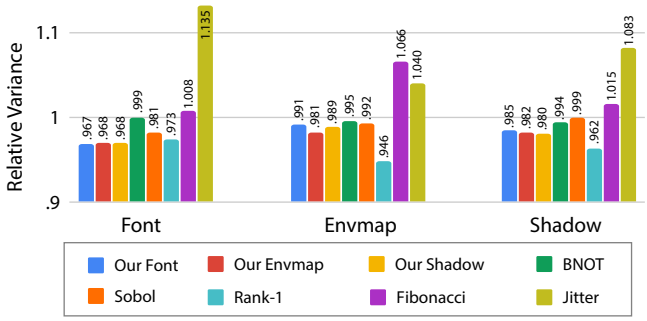

Fig. 15. Relative variance of the samplers in Fig. 13. Vertical axis is variance (less is better), different methods are different colors.

Font super-sampling. For demonstration, we first consider the practical example of font reproduction (Fig. 13, first row), which we phrase as solving the integral of the image function over a pixel by super-sampling it with a purpose-made pattern.

Envmap integration. A similar procedure can be applied to the integration of entire illumination-reflectance products in photorealistic rendering. To this end, we uniformly sample from a corpus of HDR environment maps, BRDFs, normal orientations and view positions and compute the integrand (without importance sampling) as a common 2D image where every pixel is weighted by reflectance from the BRDF slice for this view and normal. As before, we optimize in respect to the power spectrum of the average of these.

Soft shadows. In the case of area light illumination sampling, binary visibility is computed between scene and light points. Our sampling patterns are used to place light samples for a fixed scene point, integrating a binary image defined in a coordinate space parametrizing the area light surface. We here use quadratic area lights for simplicity but other parametrized shapes presumably will work similarly. Visibility is computed for random coordinates on randomly placed area light and random points on the surface of 3D scenes comprising of objects from ShapeNet [Chang et al. 2015] placed on a ground plane.

Comparisons. We find that for *font super-sampling*, our pattern performs better in terms of estimator variance than BNOT [De Goes et al. 2012], but is in close competition with Sobol and rank-1 lattice samplers. Our method outperforms a Fibonacci lattice generated using the $\{1, 144\}$ generator vector for $m = 233$ samples. For fair comparison, the Fibonacci spectrum is appropriately scaled to match the variance from $m = 256$ samples. See Fig. 15 for relative error values, i.e., $\mathcal{L}_1$ error divided by average error value of all samplers. In the case of *envmaps* and *soft shadows*, our samplers perform better than BNOT, Sobol and Fibonacci but are outperformed by a rank-1 lattice. This highlights the inexpressiveness of Eq. 5 for certain function classes in obtaining the optimal solution. To obtain maximal benefits, future work should consider the generalized variance form [Singh et al. 2019b] with potentially other characteristics (e. g., Koksma-Hlawka inequality [Niederreiter 1992]) in the loss function. Analyzing these novel patterns is also a promising future direction. The resulting improvements are documented in Fig. 15.

6.4 Scalability

Instrumentation. Fig. 16 analyzes the scalability of our approach. Fig. 16,a shows compute time (y-axis, less is better) as a function of points (x-axis) and number of dimensions (different plots). We tested for a radially averaged BNOT target with $l = 40$ layers on a Tesla K40m GPU. We see that our approach can produce substantial number of points (upto 8192). A typical point set with $1,024$ points requires ca. 100 ms to produce. In one second, we can produce sets in 2D to 10D of up to 8192 points, indicating performance scales favorably with dimensionality.

A key parameter of our architecture is its depth, i. e., the number of filter iterations. Consequently, Fig. 16,b repeats the experiment, but for architectures of varying depth. We see that the method scales linearly in all dimensions. We study the error across different depths in Fig. 16,c. We see that quality saturates around the 40 levels we suggest.

An important parameter of our method is the size of the receptive field, which we vary in Fig. 16,d to see that the receptive field of $r = .2$ of the domain size is a good compromise: Error for larger field saturates and smaller receptive fields produce low-frequency error (arrow in Fig. 16,d). This can benefit certain applications (e. g., object placement) where exact zero-energy low frequency region is not critical.

Finally, we look into the effect of filter kernels size (b) on the spectrum in Fig. 16,e, to find that quality saturate around the $b = 64$ we use. Smaller kernels smooth the profile (A) and create aliasing (B). Please note, that speed is not affected by kernel size (not shown), as it is a $O(1)$ look-up.

We conclude that our method scales across different domains and is adjustable to trade quality and speed.

6.5 Stability

Point count. Our architecture is trained with a certain point count m but can be applied to point sets of slightly different sizes. Fig. 17 shows that the spectrum remains stable when running on twice or half as many points. For point counts similar to 256, the quality is very similar to a reference and degrades noticeably at $\times 4$ or $\times .0625$. This indicates our architecture has learned to generalize over point counts to some degree. Note how we do not make point count input to the architecture and explicitly train for this generalization or find analytic ways to scale filters; which we both leave to future work.

Variance reduction. We use the average correlation of multiple realizations to reduce variance and aid training convergence. We verified this is effective by varying the number of sample patterns averaged per training step in Fig. 18.

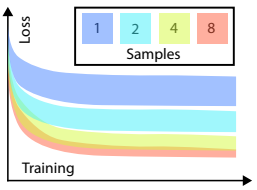

Fig. 18. Reduction of error (vertical, less is better) over training iterations for different numbers of training steps (horizontal) when using different numbers of pattern instances (colors). The funnel width corresponds to the .75 precentile of the loss over training.

Here, the horizontal axis is training progress, the vertical axis depicts loss and different colors show different number of realizations to average. The width of the bar is a smoothed .75 percentile of the loss. First, we see that adding more realizations reduces the loss. Second, a likely cause for this is also visible: the variance of the loss is reduced as well by using more realizations, seen in the tighter funnels. At the same time, we see that at around 8 realizations, the return likely diminishes.

Spectrum convergence. Fig. 19 shows the convergence of our MC spectral estimate for a 2D BNOT spectrum with varying numbers of samples. The given numbers signify the average sample count per radial frequency. We see that noise disappears gradually and a spectrum at around 16 samples is already very similar to the real

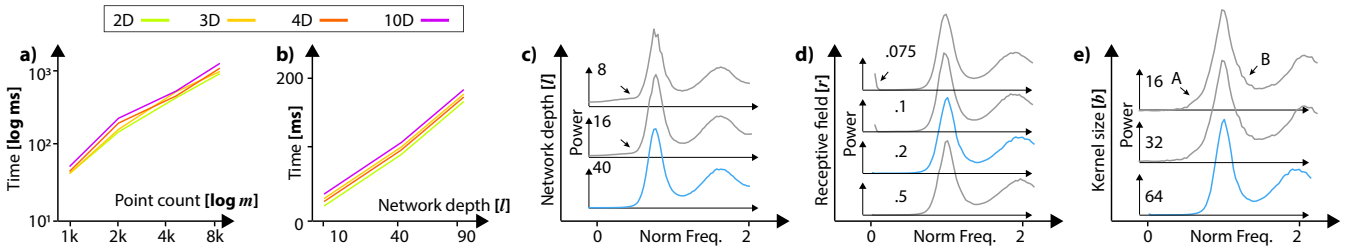

Fig. 16. Scalability analysis: Compute time per number of points **(a)** and layers **(b)**. In **(c)**, we show the change of spectrum (here and following: radially averaged 2D BNOT) for three different depths. The effect of four choices of receptive field size on the spectrum is seen in **(d)**. **(e)** documents the change of spectra when the number of filter kernel bins b varies. Please refer to Sec. 6.4 for a full discussion.

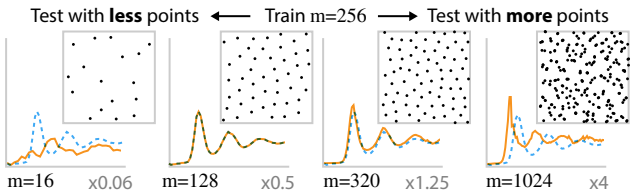

Fig. 17. Testing for point counts m' that differ from training point count $m = 256$. Test point count m' is stated as a multiple of the training point count m. Blue shows the radially-averaged 2D BNOT target; orange our result. A part of the resulting pattern is shown in each right top inset.

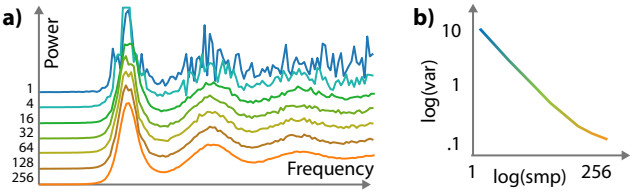

Fig. 19. To handle Fourier power spectra beyond two dimensions, we estimate them using the randomly picked frequencies $\mathbf{q}$ (in Monte Carlo fashion) at a given radius ρ. At the right (a), radial profiles for a 2D blue noise target [De Goes et al. 2012] are shown when estimated using different frequency-sample counts (colors, from 1 to 256). The plots are vertically shifted for visibility. In our experiments, we choose $\mathbf{q} = 128$ to estimate the radial spectra up to 10-dimensions. The right part (b) shows the convergence of the error (vertical axis) of the estimate of the spectrum for different numbers of samples (horizontal axis).

spectrum. At every training step the samples are different, so no bias is introduced.

Input pattern. Our method transforms a uniformly random pattern into a pattern with prescribed correlations. It is, however, important to ask how the properties of the input pattern affect the result. We hence analyze the effect of input properties on output properties in Fig. 20. The target is always radially averaged 2D BNOT but the input pattern is changed from jittered, to regular, to BNOT. We see that our approach struggles to turn a regular pattern into a BNOT pattern, while it successfully improves a jittered pattern. Finally, when ran on a BNOT input (that can also, recursively, be considered to be similar to its own output) the result is a BNOT pattern as well.

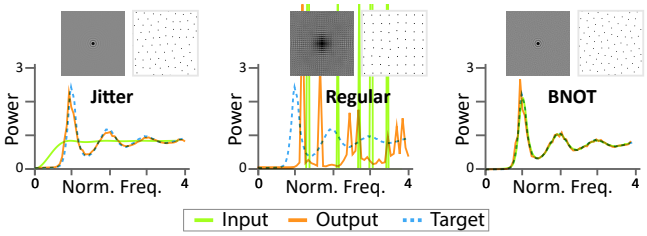

Fig. 20. Input stability analysis. Starting from different input sample patterns (shown in green: Jitter, Regular and BNOT itself – columns) our approach produces distributions (orange) that closely match the target (blue, dotted), except for the regular grid input.

6.6 Choice of metrics

We investigate how the choice of metrics (Sec. 4.7) influences result quality. We observe that $\mathcal{L}_1$ consistently achieves a more faithful fit to target spectra and differential histograms in comparison to $\mathcal{L}_2$. The difference between the metrics is particularly striking for blue-noise profiles, where we could achieve the required vanishing energies in the low-frequency regions only using $\mathcal{L}_1$ (Fig. 21).

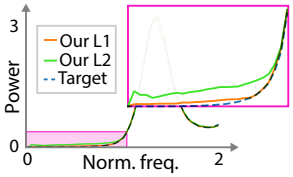

Fig. 21. Comparison of different metrics (colors) for matching a BNOT target. The pink insets zooms into the low-frequency region where differences appear.

6.7 MLP comparison

We here look into an ablation experiment that replaces the unstructured iterated convolution with a fully-connected network. We use a Multi-layer Perceptron (MLP) that maps uniform random points to point patterns: Input and output are vectors that stack all coordinates of all points. The mapping directly is regressing the coordinate values. An MLP has two main architectural parameters (Fig. 22, a): the number of hidden states and the number of layers. We found an MLP with eight hidden states per input point and five hidden layers to work best to reproduce n = 2-D BNOT with $m = 1024$ points, after experimenting with 1–8 hidden states and 1–5 layers. Regrettably, applying an MLP to our problem is difficult for two reasons: scalability and quality.

We first note, how different from our unstructured convolutional design, the number of learnable MLP parameters depends on the number of input points, and it does so quadratically due to the fully-connected weight matrix. At 1024 input points, we train at least $5 \times (1024 \times 8)^2$, that is ca. 335 M parameters (ResNet [He et al. 2016] uses 1.7 M parameters for state-of-the-art classification; we use less than 10 k). Higher point and layer counts cannot be trained on a machine equipped with 32 GB of memory.

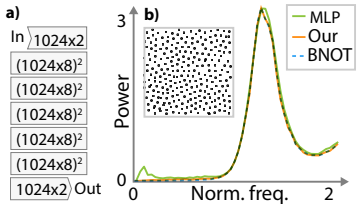

Fig. 22. MLP structure used for comparison **(a)** and spectra of a BNOT reference, our result and an MLP, where differences manifest in the low frequencies **(b)**.

Fig. 22,b shows a typical result produced by an MLP, that has two problems: quality and bias. We note that an MLP indeed produces the desired radially-averaged spectrum (Fig. 22,b, green), but with a marked bump around the low frequencies. This tendency can also be seen in the pattern itself that has uneven overall density. Note, how our approach (Fig. 22,b,orange) matches the spectrum everywhere. An MLP further produces a substantially biased pattern, which remains similar, disregarding the input. Numerical estimation of the bias [Öztireli 2016; Singh et al. 2019b; Subr and Kautz 2013] confirms this finding. This is a perfectly reasonable response to the loss used, which only constrains correlations between points. As the MLP works on absolute coordinates it can easily produce a translation-variant pattern with bias.

We conclude that an MLP is neither scalable to high point counts nor does it provide the quality required for most applications.

6.8 Applications

Our trained filters can be efficiently applied to problems such as dithering or object placement.

Dithering. The ability to compute gridding masks (dithering patterns for rendering) is demonstrated in Fig. 23. Similar to Georgiev and Fajardo [2016] we optimize for a grayscale dither mask (of size $32 \times 32 \times 1$ in this example), where x and y correspond to fixed pixel grid indices and z are optimized gray values. To achieve this, our framework does not require additional coding besides adding an enclosing `grid` operator that extracts the z dimension from a 3D pattern, keeping x and y fixed regular. Once the dither mask is created, it is tiled and the noise values are thresholded against the luminance values of the input image to determine the binary output value of each pixel. Explicit constructions of such masks can take considerable implementation effort (simulated annealing). This also demonstrates our framework's ability to handle different target spectra along different projections (in this case, blue noise along 1D and regular for the rest).

Object placement. We further demonstrate the capability of our framework to handle different target spectra. In Fig. 24, we trained for different colors of noise to proceduraly place a flower object.

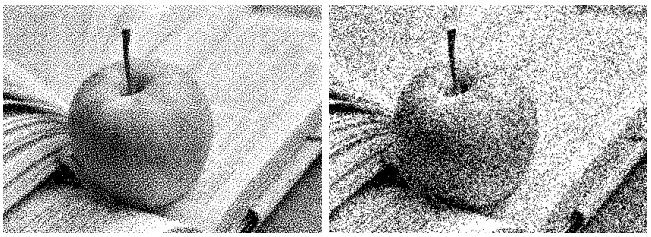

Fig. 23. Dithering for our **(left)** and a random mask **(right)**.

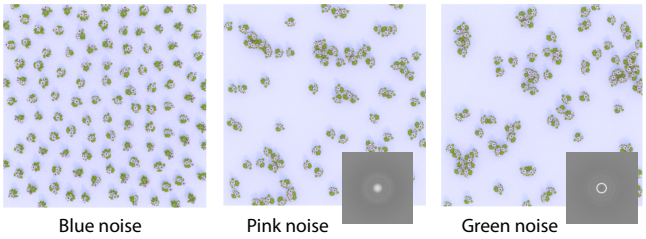

Fig. 24. Object placement for three learned colors-of-noise.

7 DISCUSSION AND LIMITATIONS

Fixed dimensionality. The trained pipeline $\mathcal{P}$ is specific to the dimension n. While the filters work on distances and are agnostic to dimensions, what they learned is dimension-dependent. Effectively, the architecture needs to be re-trained for every dimension.

Fixed point count. Our architecture is trained for a fixed number of points m. While the architecture can also be applied to similar numbers of points, drastically different point numbers require re-training. We analyze the effect of this choice in Fig. 17 to find that twice or half the point count is still similar to the original count, but more or less than that starts to degrade noticeably. Finally, scalability to a very high number of points is limited by computing the nearest-neighbor information.

User parameters. A user also has to choose the depth of the architecture l and the trained filters will be specific to that depth. As we find the solution to be progressive, i. e., the quality to improve with layers, it is possible to stop computation earlier, at $l' < l$, but the maximum quality is achieved at l. One solution is to train with a high l and then cut compute time at deployment to an l' with the required quality.

On-line and off-line point patterns. Depending on the memory and time requirements of the application, our method can be used both to pre-compute and cache a single realization off-line, as well as to quickly compute many realizations on-line, e. g., for interactive exploration of different object placements or using a different sample pattern in every pixel, requiring millions of realizations. The set of tunable parameters is compact, typically requiring orders of magnitude less storage and bandwidth than a point pattern.

Computational speed. Our method can be much slower than specialized solutions for specific problems, such as Halton or Sobol, but

faster than previous point pattern design work and is so without any low-level optimizations.

Generality. Our work as a whole is more general than previous work as it can address many different objectives. A specific learned instance of our pipeline however, can be less general, in the sense that it is specific to a target or a certain class of integrands..

Point sets and sequences. Our approach produces point sets, not point sequences, i. e., points are produced in no specific order. In particular, this for now prevents producing progressive results, e. g., a point set where the first $m/2$ points have a spectrum similar to the one of the full m points. Working on ordered, but unstructured data is a promising avenue of future work, not limited to point pattern correlation design.

High- and multi-dimensional point patterns. Our approach does not yet scale to higher dimensions (>10-D) or larger point clouds (>50 k). However, many important integration problems in rendering require much higher numbers of dimension. The first is due to the separable construction: the effort is still linear in the number of dimensions. While doing so allows filters to operate in such spaces, it is no guarantee that composing them into a pipeline is effective: distances in high-dimensional spaces often behave differently and indeed we see increased difficulty for high-dimensional patterns. Achieving properties across all the combinatorically many possible subspaces is also difficult, which is why we demonstrate correlation in a selection of subspaces and (radially-averaged) in the full space.

Metrics. In this work we restrict ourselves to using the common and simple metrics $\mathcal{L}_1$ and $\mathcal{L}_2$. However, observing their notably different influence on the results, we see the use of more advanced metrics (e. g., Wasserstein, total variation) and their combinations as promising future work.

Continuous vs. discrete. Our approach operates on continuous point patterns by design, but the gridding function enables the system to work on (partially) discrete point patterns like dither masks. While we demonstrate the ability to successfully produce blue-noise masks, our filter-based architecture is not the best fit for this type of application. We therefore expect specialized approaches [Georgiev and Fajardo 2016] to perform better on this task.

Generative modeling. Different from many other generative models we do not aim at modeling the underlying distribution of target point sets, but rather exclusively focus on correlations. We consider a combination of our approach with more advanced learnable architectures, such as Real NVP [Dinh et al. 2017] or normalizing flow [Rezende and Mohamed 2015] promising directions to explore.

8 CONCLUSION

We have proposed a framework to optimize for methods that turn uniform random points into point patterns with properties relevant for computer graphics tasks. Other than previous work that requires mathematical derivation and implementation effort, we simply state the forward model as a loss and rely on modern back-propagation software to come up with a point pattern generation method. The methods resulting from our approach are versatile: As we have shown several previous patterns can be emulated and in some cases even surpassed in terms of quality and/or computation speed (in multiple dimension, anisotropy, subspaces). We share execution efficiency with classic CNNs that require only a few passes across the input and complete GPU-friendly data-parallelism. Point sets of ten-thousands are generated in a second.

Still many questions remain to be answered. While we state the loss and hope for modern optimizers to find good solution, at the one hand, we lack any theoretical guarantees. On the other hand, most mathematical derivations also do not provide proofs, such as we are unaware of proofs that Lloyd relaxation converges in high dimensions. Future work will need to investigate a detailed convergence analysis for different losses. Ultimately we would want to ask if any point pattern can be learned as we here have only shown a small, but important subset.

We think that our approach can be categorized as a generative model that is trained in a semi-supervised fashion, specifically through inexact supervision [Zhou 2017]. Clearly, we do not provide supervision in form of pairs of input and output that sample a mapping. Instead, we learn filters, that, when applied to originally random data are free to do to those points what they please, as long as they introduce structure in the form of point correlation.

Ultimately, we hope that our approach will support exploration of new point patterns, make their application easier in practice, and finally move forward their theoretical understanding.

ACKNOWLEDGMENTS

We would like to thank all the reviewers for their detailed and constructive feedback. This work was supported by the Fraunhofer Society and the Max Planck Society as a cooperation program within the German Pact for Research and Innovation (PFI).

REFERENCES

Abadi et al. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proc. USENIX.*

Pankaj K Agarwal, Jeff Erickson, et al. 1999. Geometric range searching and its relatives. *Contemp. Math.* 223 (1999).

Abdalla GM Ahmed, Hélène Perrier, David Coeurjolly, Victor Ostromoukhov, Jianwei Guo, Dong-Ming Yan, Hui Huang, and Oliver Deussen. 2016. Low-discrepancy blue noise sampling. *ACM Trans. Graph.* 35, 6 (2016).

Luke Anderson, Tzu-Mao Li, Jaakko Lehtinen, and Frédo Durand. 2017. Aether: An Embedded Domain Specific Sampling Language for Monte Carlo Rendering. *ACM Trans. Graph.* 36, 4 (2017).

Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point Convolutional Neural Networks by Extension Operators. *ACM Trans. Graph.* 37, 4 (2018).

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4 (2017).

Michael Balzer, Thomas Schlömer, and Oliver Deussen. 2009. Capacity-constrained point distributions: a variant of Lloyd's method. *ACM Trans. Graph.* 28, 3 (2009).

Gilbert Louis Bernstein, Chinmayee Shah, Crystal Lemire, Zachary Devito, Matthew Fisher, Philip Levis, and Pat Hanrahan. 2016. Ebb: A DSL for Physical Simulation on CPUs and GPUs. *ACM Trans. Graph.* 35, 2 (2016).

John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. 2010. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph.* 29, 6 (2010).

Luca Brandolini, Leonardo Colzani, and Andrea Torlaschi. 2001. Mean square decay of Fourier transforms in Euclidean and non Euclidean spaces. *Tohoku Math J* 53, 3 (2001).

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph. (Proc. SIGGRAPH)* 36, 4 (2017).

Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong

Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. *arxiv:1512.03012* (2015).
Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. 2012. Variational blue noise sampling. *IEEE Trans. Vis Comp. Graph.* 18, 10 (2012).
Kenneth Chiu, Peter Shirley, and Changyaw Wang. 1994. Graphics Gems IV. Chapter Multi-jittered Sampling.
Per Christensen, Andrew Kensler, and Charlie Kilpatrick. 2018. Progressive multi-jittered sample sequences. *Comp. Graph. Forum (Proc. EGSR)* 37, 4 (2018).
Sebastian Claici, Edward Chien, and Justin Solomon. 2018. Stochastic Wasserstein Barycenters. *arxiv:1802-05757* (2018).
Richard Condit, Peter S. Ashton, Patrick Baker, Sarayudh Bunyavejchewin, Savithri Gunatilleke, Nimal Gunatilleke, Stephen P. Hubbell, Robin B. Foster, Akira Itoh, James V. LaFrankie, Hua Seng Lee, Elizabeth Losos, N. Manokaran, R. Sukumar, and Takuo Yamakura. 2000. Spatial Patterns in the Distribution of Tropical Tree Species. *Science* 288, 5470 (2000).
Robert L Cook. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph.* 5, 1 (1986).
Ken Dahm and Alexander Keller. 2017. Learning Light Transport the Reinforced Way. *arXiv:1701.07403* (2017).
Sabrina Dammertz and Alexander Keller. 2008. Image Synthesis by Rank-1 Lattices. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*.
Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Trans. Graph.* 31, 6 (2012).
Zachary Devito, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Niessner. 2017. Opt: A Domain Specific Language for Non-Linear Least Squares Optimization in Graphics and Imaging. *ACM Trans. Graph.* 36, 5 (2017).
Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. In *ICLR*.
David P. Dobkin, David Eppstein, and Don P. Mitchell. 1996. Computing the Discrepancy with Applications to Supersampling Patterns. *ACM Trans. Graph.* 15, 4 (1996).
Carola Doerr and François-Michel De Rainville. 2013. Constructing Low Star Discrepancy Point Sets with Genetic Algorithms. In *Proc. GECCO*.
Ron O Dror, Thomas K Leung, Edward H Adelson, and Alan S Willsky. 2001. Statistics of real-world illumination. In *CVPR*.
Raanan Fattal. 2011. Blue-noise point sampling using kernel density model. *ACM Trans. Graph.* 30, 4 (2011).
Iliyan Georgiev and Marcos Fajardo. 2016. Blue-noise Dithered Sampling. In *ACM SIGGRAPH 2016 Talks*.
Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*.
John H. Halton. 1960. On the Efficiency of Certain Quasi-random Sequences of Points in Evaluating Multi-dimensional Integrals. *Numer. Math.* 2, 1 (1960).
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.
Daniel Heck, Thomas Schlömer, and Oliver Deussen. 2013. Blue noise sampling with controlled aliasing. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 3 (2013).
A.S. Hedayat, N.J.A. Sloane, and John Stufken. 1999. *Orthogonal Arrays: Theory and Applications*. Springer New York.
Felix Heide, Steven Diamond, Matthias Niessner, Jonathan Ragan-Kelley, Wolfgang Heidrich, and Gordon Wetzstein. 2016. ProxImaL: Efficient Image Optimization Using Proximal Algorithms. *ACM Trans. Graph.* 35, 4 (2016).
Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. 2018. Monte Carlo Convolution for Learning on Non-uniformly Sampled Point Clouds. *ACM Trans. Graph (Proc. SIGGRAPH Asia)* 37, 5 (2018).
JS Hicks and RF Wheeling. 1959. An efficient method for generating uniformly distributed points on the surface of an n-dimensional sphere. *Comm. ACM* 2, 4 (1959).
Wojciech Jarosz, Afnan Enayet, Andrew Kensler, Charlie Kilpatrick, and Per Christensen. 2019. Orthogonal array sampling for Monte Carlo rendering. *Computer Graphics Forum (Proceedings of EGSR)* 38, 4 (2019).
Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. 2015. Blue noise sampling using an SPH-based method. *ACM Trans. Graph.* 34, 6 (2015).
S. Joe and F. Kuo. 2008. Constructing Sobol Sequences with Better Two-Dimensional Projections. *SIAM J Scientific Comp.* 30, 5 (2008).
Bhavya Kailkhura, Jayaraman J Thiagarajan, Peer-Timo Bremer, and Pramod K Varshney. 2016. Stair blue noise sampling. *ACM Trans. Graph.* 35, 6 (2016).
Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4 (2015).
Simon Kallweit, Thomas Müller, Brian Mcwilliams, Markus Gross, and Jan Novák. 2017. Deep Scattering: Rendering Atmospheric Clouds with Radiance-predicting Neural Networks. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 36, 6 (2017).
Alexander Keller, Simon Premoze, and Matthias Raab. 2012. Advanced (Quasi) Monte Carlo Methods for Image Synthesis. In *ACM SIGGRAPH 2012 Courses*. Article 21.
Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arxiv:1412.6980* (2014).
Diederik P Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv:1312.6114* (2013).
Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang tiles for real-time blue noise. *ACM Trans. Graph. (Proc. SIGGRAPH)* 25, 3 (2006).
Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*.
Christopher Kulla, Alejandro Conty, Clifford Stein, and Larry Gritz. 2018. Sony Pictures Imageworks Arnold. *ACM Trans. Graph.* 37, 3 (2018).
Frances Kuo. 2007. Lattice rule generating vectors. web.maths.unsw.edu.au/~fkuo. Accessed: 2019-07-12.
Ares Lagae and Philip Dutre. 2008. A Comparison of Methods for Generating Poisson Disk Distributions. *Comp. Graph. Forum* 27, 1 (2008).
Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. 2018. Differentiable Programming for Image Processing and Deep Learning in Halide. *ACM Trans. Graph.* 37, 4 (2018).
Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans Inform. Theory* 28, 2 (1982).
Benoit B Mandelbrot. 1983. *The fractal geometry of nature*. WH Freeman New York.
Michael McCool and Eugene Fiume. 1992. Hierarchical Poisson disk sampling distributions. In *Proc. Graphics interface*.
Don P. Mitchell. 1992. Ray Tracing and Irregularities of Distribution. In *In Third Eurographics Workshop on Rendering*.
Scott A. Mitchell, Mohamed S. Ebeida, Muhammad A. Awad, Chonhyon Park, Anjul Patney, Ahmad A. Rushdi, Laura P. Swiler, Dinesh Manocha, and Li-Yi Wei. 2018. Spoke-Darts for High-Dimensional Blue-Noise Sampling. *ACM Trans. Graph.* 37, 2 (2018).
Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Nov'ak. 2018. Neural Importance Sampling. *arXiv:1808.03856* (2018).
Jeffrey B Mulligan and Albert J Ahumada. 1992. Principled halftoning based on human vision models. In *Human Vision, Visual Processing, and Digital Display III*, Vol. 1666.
Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen-Space Shading. *Comp. Graph. Forum (Proc. EGSR)* 36, 4 (2017).
Harald Niederreiter. 1978. Quasi-Monte Carlo methods and pseudo-random numbers. *Bull. Amer. Math. Soc.* 84, 6 (1978).
H. Niederreiter. 1992. *Random Number Generation and Quasi-Monte-Carlo Methods*. SIAM.
Dirk Nuyens. 2013. The construction of good lattice rules and polynomial lattice rules. *arXiv:1308.3601* (2013).
Deussen Oliver, Hiller Stefan, Van Overveld Cornelius, and Strothotte Thomas. 2001. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum* 19, 3 (2001).
Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.* 23, 3 (2004).
Art B. Owen. 1997. Monte Carlo Variance of Scrambled Net Quadrature. *SIAM J. Numer. Anal.* 34, 5 (1997), 27.
A Cengiz Öztireli. 2016. Integration with stochastic point processes. *ACM Trans. Graph.* 35, 5 (2016).
A Cengiz Öztireli and Markus Gross. 2012. Analysis and synthesis of point distributions based on pair correlation. *ACM Trans. Graph.* 31, 6 (2012).
Hélène Perrier, David Coeurjolly, Feng Xie, Matt Pharr, Pat Hanrahan, and Victor Ostromoukhov. 2018. Sequences with Low-Discrepancy Blue-Noise 2-D Projections. *Comp. Graph. Forum (Proc. Eurographics)* 37, 2 (2018).
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory To Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. 2015. Variance analysis for Monte Carlo integration. *ACM Trans. Graph.* 34, 4 (2015).
Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3D classification and segmentation. *CVPR* (2017).
Hongxing Qin, Yi Chen, Jinlong He, and Baoquan Chen. 2017. Wasserstein Blue Noise Sampling. *ACM Trans. Graph.* 36, 5 (2017).
Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. 2016. Projective Blue-Noise Sampling. *Comp. Graph. Forum* 35, 1 (2016).
Peiran Ren, Yue Dong, Stephen Lin, Xin Tong, and Baining Guo. 2015. Image Based Relighting Using Neural Networks. *ACM Trans. Graph.* 34 (2015).
Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. *arXiv:1505.05770* (2015).
Christian Schmaltz, Pascal Gwosdek, Andres Bruhn, and Joachim Weickert. 2010. Electrostatic Halftoning. *Comp. Graph. Forum* (2010).
David W Scott. 1979. On optimal and data-based histograms. *Biometrika* 66, 3 (1979).
Adrian Secord. 2002. Weighted Voronoi stippling. In *Proc. NPAR*.
Peter Shirley. 1991. Discrepancy as a quality measure for sample distributions. In *Proc. Eurographics*.

Eero P Simoncelli and Bruno A Olshausen. 2001. Natural image statistics and neural representation. *Ann. Review Neuroscience* 24, 1 (2001).
Gurprit Singh and Wojciech Jarosz. 2017. Convergence Analysis for Anisotropic Monte Carlo Sampling Spectra. *ACM Trans. Graph. (Proc. SIGGRAPH)* 36, 4 (2017).
Gurprit Singh, Cengiz Oztireli, Abdalla G.M. Ahmed, David Coeurjolly, Kartic Subr, Oliver Deussen, Victor Ostromoukhov, Ravi Ramamoorthi, and Wojciech Jarosz. 2019a. Analysis of Sample Correlations for Monte Carlo Rendering. *Comp. Graph Form. (Proc. EGSR)* 38, 2 (2019).
Gurprit Singh, Kartic Subr, David Coeurjolly, Victor Ostromoukhov, and Wojciech Jarosz. 2019b. Fourier Analysis of Correlated Monte Carlo Importance Sampling. *Comp. Graph. Forum* 38, 1 (2019).
I.H. Sloan and S. Joe. 1994. *Lattice methods for multiple integration.* Clarendon Press.
I. M. Sobol. 1967. The distribution of points in a cube and the approximate evaluation of integrals. *U. S. S. R. Comput. Math. and Math. Phys.* 7 (1967).
Kartic Subr and Jan Kautz. 2013. Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration. *ACM Trans. Graph.* 32 (2013).
Robert A Ulichney. 1988. Dithering with blue noise. *Proc. IEEE* 76, 1 (1988).
Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando De Goes, Mathieu Desbrun, and Victor Ostromoukhov. 2014. Fast tile-based adaptive sampling with user-specified Fourier spectra. *ACM Trans. Graph.* 33, 4 (2014).
Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. 2018b. Deep parametric continuous convolutional neural networks. In *CVPR.*
Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2018a. Dynamic Graph CNN for Learning on Point Clouds. *arxiv:1801.07829* (2018).
Li-Yi Wei and Rui Wang. 2011. Differential domain analysis for non-uniform sampling. *ACM Trans. Graph.* 30, 4 (2011).
Dong-Ming Yan, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka. 2015. A survey of blue-noise sampling and its applications. *J Comp. Sci. and Tech.* 30, 3 (2015).
John I Yellott. 1983. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science* 221, 4608 (1983).
Quan Zheng and Matthias Zwicker. 2018. Learning to Importance Sample in Primary Sample Space. *arxiv:1808.07840* (2018).
Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. 2012. Point sampling with general noise spectrum. *ACM Trans. Graph.* 31, 4 (2012).
Zhi-Hua Zhou. 2017. A brief introduction to weakly supervised learning. *National Science Review* 5, 1 (2017).

A GRADIENTS

Our unstructured filters (Eq. 3) can be used directly in a trivial TensorFlow implementation. However, we found this to execute much slower than a hand-crafted C++ implementation and most of all required memory in the order of $O(m^2)$. We here give the derivation required to implement our filter as a custom operation: a forward pass to filter points ($\mathcal{F}$ Eq. 3) and the derivative passes in respect to kernel weights parameters ($\partial\mathcal{L}/\partial\theta$, Eq. 3) and input point positions ($\partial\mathcal{L}/\partial\mathbf{x}$, Eq. 3).

Both the *forward pass* and the *backward pass* are implemented parallel over all points $\mathbf{x}_i$. Each point sequentially iterates all neighbors $\mathbf{x}_{i,j}$. Distances d_{ij} are computed and the weights θ are indexed at $\lceil d_{ij} \rceil$ and $\lfloor d_{ij} \rfloor$, interpolated, and applied to the offsets $\mathbf{x}_i \ominus \mathbf{x}_j$.

The *backward pass* requires the derivative of the loss $\mathcal{L}$ of Eq. 3 in respect to the l-th weight θ, which is

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \sum_i \sum_{j \neq i} \frac{\partial g(d_{ji}|\theta)}{\partial \theta_l} \frac{1}{d_{ij}} \left\langle \mathbf{x}_i \ominus \mathbf{x}_j, \frac{\partial \mathcal{L}}{\partial (\mathcal{F}(\mathbf{x}|\theta))_i} \right\rangle$$

a b-dimensional vector, where $d_{ab} = ||\mathbf{x}_a \ominus \mathbf{x}_b||$. Here,

$$\frac{\partial}{\partial \theta_l} g(d|\theta) = \begin{cases} 1 - \text{frac}(\hat{d}) & \text{if } \lfloor \hat{d} \rfloor = l \\ \text{frac}(\hat{d}) & \text{if } \lceil \hat{d} \rceil = l \\ 0 & \text{else} \end{cases},$$

where $\hat{d} = d \cdot b/r$ is the index scaled by receptive field and bin count and $\text{frac}(\hat{d})$ returns the fractional part of a real $\hat{d}$. The derivative of Eq. 3 in respect to the k-th dimension of the i-th input point is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{i,k}} = \frac{\partial \mathcal{L}}{(\partial \mathcal{F}(\mathbf{x}|\theta))_{i,k}} + \left\langle \sum_{j \neq i} \frac{\partial \otimes_{ji}}{\partial \mathbf{x}_{i,k}}, \frac{\partial \mathcal{L}}{(\partial \mathcal{F}(\mathbf{x}|\theta))_i} \right\rangle + \sum_{j \neq i} \left\langle \frac{\partial \otimes_{ij}}{\partial \mathbf{x}_{i,k}}, \frac{\partial \mathcal{L}}{(\partial \mathcal{F}(\mathbf{x}|\theta))_j} \right\rangle$$

where $\otimes_{ba}$ is the pairwise interaction between point b and a

$$\otimes_{ba} = g(||\mathbf{x}_b \ominus \mathbf{x}_a||_2|\theta) \frac{\mathbf{x}_b \ominus \mathbf{x}_a}{||\mathbf{x}_b \ominus \mathbf{x}_a||}$$

that has the derivative

$$\frac{\partial \otimes_{ba}}{\partial \mathbf{x}_{b,k}} = \frac{1}{d_{ba}^2} \left(\frac{1-b}{r} (w_{\lceil \hat{d} \rceil} - w_{\lfloor \hat{d} \rfloor})(\mathbf{x}_{b,k} \ominus \mathbf{x}_{a,k})(\mathbf{x}_b \ominus \mathbf{x}_a) + g(d_{ba}|\theta) \cdot \left(\frac{(\mathbf{x}_{b,k} \ominus \mathbf{x}_{a,k})(\mathbf{x}_b \ominus \mathbf{x}_a)}{d_{ba}} - \mathbb{1}_k d_{ba} \right) \right),$$

where $\mathbb{1}_k$ is a one-hot vector. Finally, $\partial \otimes_{ba} / \partial \mathbf{x}_{a,k} = -\partial \otimes_{ba} / \partial \mathbf{x}_{b,k}$.

B SYMBOLS

Table 1. Symbols used in this work.

Symbol	Meaning	Domain
b	Number of kernel bins	$\mathbb{Z}^+$
$\mathbf{d}$	Offset	$\mathbb{R}^n$
$\mathcal{F}$	Filter	$\mathbb{R}^n \to \mathbb{R}^n$
g	Kernel	$\mathbb{R}_{\geq 0} \to \mathbb{R}$
h	Unique kernel	$\mathbb{R}_{\geq 0} \to \mathbb{R}$
l	Number of filters	$\mathbb{Z}^+$
l_u	Number of unique kernels	$\mathbb{Z}^+$
$\mathcal{L}$	Loss	$\mathbb{R}^{n \times m} \to \mathbb{R}_{\geq 0}$
$\mathcal{H}$	Histogram	$\mathbb{R}_{\geq 0}^{n_\text{c}}$
κ	Correlation function	$(\mathbb{R}^n \times \mathbb{R}^n) \to \mathbb{R}$
m	Number of points	$\mathbb{Z}^+$
M	Weight matrix	$\mathbb{R}^{s \times s}$
n	Number of dimensions	$\mathbb{Z}^+$
n_a	Radial average resolution	$\mathbb{Z}^+$
n_c	Point correlation resolution	$\mathbb{Z}^+$
	Normal distribution	$\mathcal{N}$
	Hypersphere	Ω
P	Point correlation	$\left(\mathbb{R}^{n \times m} \times \mathbb{R}^n\right) \to \mathbb{R}_{\geq 0}$
$\mathcal{P}$	Pipeline	$\mathbb{R}^{n \times m} \to \mathbb{R}^{n \times m}$
$\mathbf{q}$	Correlate	$\mathbb{R}^n$
r	Receptive field	$\mathbb{R}_{\geq 0}$
ρ	Correlation radius	$\mathbb{R}_{\geq 0}$
s	Number of subspaces	$\mathbb{Z}^+$
$\mathbf{S}$	Spectrum	$\mathbb{R}_{\geq 0}^{n_\text{c}}$
θ	Learnable parameters	$\mathbb{R}^{b \times l_\text{u} \times s}$
	Uniform distribution	$\mathcal{U}$
$\mathbf{x}$	Point	$\mathbb{R}^n$
X	Point set	$\mathbb{R}^{n \times m}$