Curl Noise Jittering

J Andreas Baerentzen, Jeppe Revall Frisvad, Jonàs Martínez

Curl Noise Jittering

J. Andreas Bærentzen
Technical University of Denmark
Lyngby, Denmark
janba@dtu.dk

Jeppe Revall Frisvad
Technical University of Denmark
Lyngby, Denmark
jerf@dtu.dk

Jonàs Martínez
Université de Lorraine, CNRS, Inria, Loria
Nancy, France
jonas.martinez-bayona@inria.fr

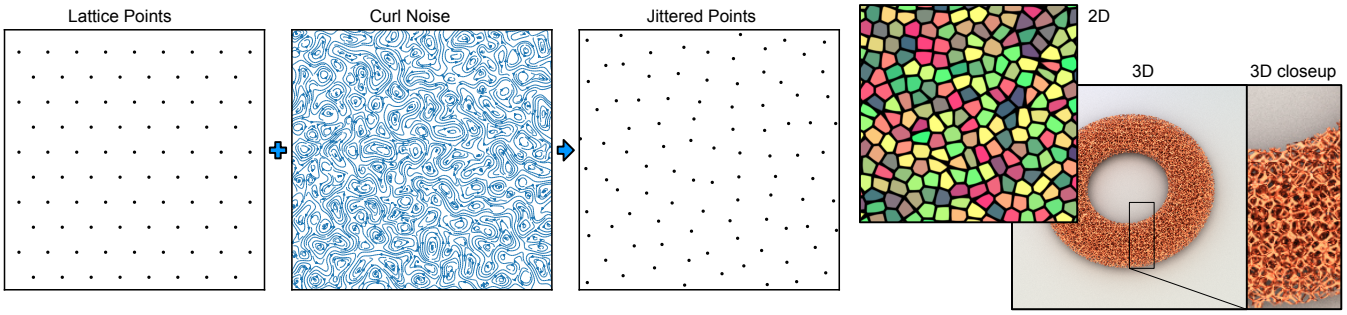

Figure 1: Points on a lattice are displaced by advecting them along a curl noise vector field. It is possible to efficiently find the closest jittered point to a given query point, and the jittered points have blue noise properties. The jittered points can be used for sampling, procedural texturing, or defining cellular materials (in 2D or 3D). The images on the right show a procedural texture based on Worley noise and a copper foam generated from our curl noise jittered points.

ABSTRACT

We propose a method for implicitly generating blue noise point sets. Our method is based on the observations that curl noise vector fields are volume-preserving and that jittering can be construed as moving points along the streamlines of a vector field. We demonstrate that the volume preservation keeps the points well separated when jittered using a curl noise vector field. At the same time, the anisotropy that stems from regular lattices is significantly reduced by such jittering. In combination, these properties entail that jittering by curl noise effectively transforms a regular lattice into a point set with blue noise properties. Our implicit method does not require computing the point set in advance. This makes our technique valuable when an arbitrarily large set of points with blue noise properties is needed. We compare our method to several other methods based on jittering as well as other methods for blue noise point set generation. Finally, we show several applications of curl noise jittering in two and three dimensions.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**.

KEYWORDS

jittered grid, curl noise, blue noise

ACM Reference Format:
J. Andreas Bærentzen, Jeppe Revall Frisvad, and Jonàs Martínez. 2023. Curl Noise Jittering. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610548.3618163

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0315-7/23/12…$15.00
https://doi.org/10.1145/3610548.3618163

1 INTRODUCTION

Sampling points in a 2D or a 3D domain is a common task in computer graphics. Several applications favor so-called *blue noise* sampling [Ulichney 1988]. Since colors of noise are defined in analogy with colors of light, *blue* means that the spectrum of the noise predominantly contains high frequencies. Spatially, this translates to point samples spread isotropically and evenly over the domain so that no two points are too close together. When using point collections for applications such as procedural generation of cellular materials or foams, distributing scattered objects, or sampling, broadly, blueness is an important property.

Poisson disk sampling [Cook 1986], also known as random sequential adsorption [Feder 1980], is a classical algorithm for computing a blue noise sampling. The principle is to generate random points but reject anyone too close to a previously added point. Figure 4 (rightmost column) shows an example of a Poisson disk sampling. Compared to jittering points on a regular grid, the corresponding frequency spectrum is "bluer", i.e., there is a wider gap between the central peak and the halo of noise surrounding it.

The rejection scheme used in the original Poisson disk sampling is slow - especially as the domain starts filling up, and more sampled points are likely to be rejected. Subsequent works have significantly improved computational efficiency and blue noise sampling

quality. However, almost all approaches require the entire point set to be computed and stored. This is unsuitable for applications where we only seek to determine a small subset of sample points or where storing the entire point set is infeasible. Applications that require this property usually resort to approaches based on jittering, where points within a grid are randomly perturbed within each grid cell [Cook 1986]. This makes it possible to find the closest sample by inspecting only the jittered positions of lattice points within a certain distance of the query point, and the positions of the rest of the points do not need to be computed.

In this paper, we refer to a method for generating a point collection as *implicit* when it provides the point closest to a query point without generating the entire collection and independently of the size of the collection. Our key insight is that displacing the points of a regular lattice along a *curl noise* [Bridson et al. 2007] vector field enhances the blue noise sampling quality (Figure 1). Since the displacements are small, our method can be considered a jittering approach, and it is possible to define an implicit method for generating point collections jittered by curl noise.

Curl noise is a divergence-free vector noise designed to emulate the velocity field of incompressible, turbulent fluids. Because curl noise is divergence-free, it contains neither sources nor sinks, which is important for our purposes. We carefully choose the curl noise parameters to achieve a balanced tradeoff between point set quality and computational efficiency. Notably, our method significantly improves the blue noise quality compared to prior works based on jittered grid sampling (see Figure 4).

An implementation of our method can be found at https://github.com/jonasmb/curlnoisejittering.

In summary, we provide the following contributions:

- *Curl Noise Jittering* (CNJ): an implicit method for computing 2D and 3D blue noise point distributions.
- A study of the suitable parameters of CNJ to improve sampling quality while preserving computational efficiency.
- An evaluation of the benefits of CNJ for implicit material modeling and rendering.

1.1 Related Work

In this section, we review and compare prior methods and applications for point sampling in the context of computer graphics.

Poisson disk sampling. A widely used method considers a Poisson disk sampling in which any two points are separated by a minimum distance [Cook 1986] that results in a blue noise sampling. The standard dart-throwing algorithm consists of iteratively placing points that are separated from each other by a minimum distance. Several works have studied efficient algorithms for Poisson disk sampling [Dunbar and Humphreys 2006; Bridson 2007; Wei 2008; Ebeida et al. 2011; Corsini et al. 2012; Yuksel 2015], and most of them are summarized in surveys [Lagae and Dutré 2008; Yan et al. 2015]. Poisson disk sampling algorithms are typically iterative processes that do not lend themselves to efficiently determining a subset of points lying in a given subspace.

Optimization-based methods. A broad class of sampling methods seek to optimize the position of a set of points to obtain blue noise. A popular method is Lloyd's algorithm [Lloyd 1982], in which the points are iteratively moved to the centroid of their corresponding Voronoi cells. Lloyd's algorithm was enhanced with the capacity-constrained point distribution method [Balzer et al. 2009] and subsequently improved through the lens of optimal transport [De Goes et al. 2012; Qin et al. 2017]. Fattal [2011] introduced an interacting particles model based on statistical mechanics. Similarly, Jiang et al. [2015] proposed a sampling algorithm based on smoothed particle hydrodynamics. Heck et al. [2013] proposed constructing sampling patterns with prescribed spectral properties by iteratively updating the point positions. Recently, Ahmed et al. [2022] presented an optimization method based on placing a set of Gaussian kernels on the sample points and defining an objective function that can be minimized with gradient-descent and that leads to a blue noise distribution. While delivering high-quality results, optimization-based methods eventually become inefficient as the number of samples increases and do not allow efficient retrieval of the points in a given subspace.

Tile-based methods. Another set of methods partitions the 2D space with tiles to sample points. For instance, by subdividing a Penrose tiling [Ostromoukhov et al. 2004], by precomputing and optimizing a set of tiles containing points [Hiller et al. 2001; Lagae and Dutré 2005; Kopf et al. 2006; Ahmed et al. 2017], by tiling with polyominos [Ostromoukhov 2007], hexagons [Wachtel et al. 2014], or AA optimized patterns [Ahmed et al. 2015]. To the best of our knowledge, tiling approaches only considered the 2D case since its most typical application is digital half-toning and rendering.

Low-discrepancy sequences. Another class of efficient methods is based on generating so-called low discrepancy sequences, which are of interest for Monte Carlo integration [Pilleboue et al. 2015]. Classical examples are the Halton [1964], Hammersley [1964] or Sobol [1967] sequences. Recent work considered scrambled Morton ordering [Ahmed and Wonka 2020], improving Sobol in lower dimensions [Paulin et al. 2021], or optimizing and exploring a space of sequences named dyadic nets [Ahmed and Wonka 2021; Ahmed et al. 2023]. It is possible to find the coordinates of a point in a given stratum of a $(0, m, 2)$ dyadic net containing 2^m points in $O(m)$ time. This entails that a logarithmic time algorithm for locating the closest point in a dyadic net is feasible, provided that the binary flags that determine the net can be computed in constant time. Unfortunately, this means that an optimized dyadic net, e.g. a Blue Net [Ahmed and Wonka 2021], does not seem to admit an efficient query for the closest lattice point.

Error-diffusion methods. Blue noise is intimately related to the application of digital halftoning [Ulichney 1988], which seeks to depict a grayscale image by varying the size or the distribution of tiny black dots arranged in a regular pattern. Different techniques based on error diffusion [Floyd and Steinberg 1976], in which a quantization error is distributed among pixels, have considered blue noise samplings [Lau et al. 2003; Ostromoukhov 2001]. Again, such algorithms are inherently iterative and do not allow for the implicit identification of points in a subset of the space.

Jittered-grid methods. Some approaches have explored how to enhance the original jittering method. For instance, to modify the jittering in order to improve the projected sampling onto the vertical or horizontal line with multi-jittering methods [Chiu et al. 1994;

Christensen et al. 2018]. Similarly, Kensler [2013] used correlated jittering of rows and columns. Klassen [2000] proposed to smooth the result by considering the average perturbation between the neighbors of a grid cell. Dammertz [2009] considered jittering rank-1 lattices with Lloyd's algorithm. Our method is also an improved jittering method, but it significantly improves the blue noise quality compared to Klassen [2000] and Kensler [2013], see Figure 4.

Jittered grid and procedural noises. Jittered grids have endured the test of time thanks to their efficient, implicit, and simple formulation, and numerous procedural texturing methods use jittering to implicitly retrieve a local neighborhood of samples around a query point. For instance, Worley noise [Worley 1996] and its descendants [Martínez et al. 2016] are examples of procedural cellular structures relying on a jittered grid. Our technique improves the quality of Worley noise, leading to better visual results while incurring a slight loss of efficiency (see Figure 5). Worley noise is one example among many other procedural methods based on jittering that may directly benefit from our method.

2 CURL NOISE JITTERING IN 2D

We define jittering as the process of displacing the points, P, of a regular lattice by a small amount to obtain a less regular set of points, P^*. Regular lattices are typically formed as the vertices of tilings of the plane. While quadrangular tilings are often used, we observe that the densest packing of circles [Fejes 1942] is obtained by associating a circle with each vertex of a triangular tiling, which also has six-fold symmetry rather than four-fold. Moreover, initial experiments gave better results for this type of lattice. Hence, we use lattices formed as the vertices of triangular tiling, as shown in Figure 1 (left).

We can obtain the jittered points in two ways. We can generate P^* *directly* by computing the displacement for each point $\mathbf{p} \in P$ and storing the jittered point $\mathbf{p}^* \in P^*$. This is the most efficient option if P is known in advance and of acceptable size. However, the benefit of methods based on jittering is that we can generate P^* *implicitly*. Namely, we can find the point $\mathbf{p}^* \in P^*$ closest to a given query point $\mathbf{x}$ without generating P^*.

For implicit evaluation, the portion of the lattice that we need to inspect depends on the largest possible jitter displacement, which we will denote the *jitter radius*, r_j. Say, $\mathbf{p} \in P$ is the lattice point closest to the query point $\mathbf{x}$. The distance to the corresponding jittered point, $\mathbf{p}^*$, is at most $r_j + \|\mathbf{x} - \mathbf{p}\|$. Thus, the closest jittered point must lie within a disk of radius $r_j + \|\mathbf{x} - \mathbf{p}\|$ centered at $\mathbf{x}$ (see Figure 2). We need to inspect all lattice points whose disks of radius r_j intersect the disk at $\mathbf{x}$. In practice, this set includes neighbors (i.e., adjacent lattice points), second-order neighbors, and, in rare cases, third-order neighbors of $\mathbf{p}$.

Jittering can be construed as moving the points of the initial lattice along a vector field. Expressed in this fashion, $\mathbf{p}^*$ is obtained by tracing a streamline starting at $\mathbf{p}$ from time $t = 0$ till $t = 1$,

$$\mathbf{p}^* = \mathbf{C}_\mathbf{p}(1)\,, \tag{1}$$

Closest lattice point in blue. The first-order neighbors are light grey, and the second-order neighbors are dark grey. The closest jittered point must lie within the disk centered on the closest lattice point.

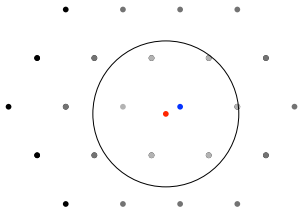

Figure 2: A query point shown (red) in a triangular lattice.

where the curve $\mathbf{C}_\mathbf{p}$ is the streamline defined in terms of the differential equation

$$\mathbf{C}_\mathbf{p}(0) = \mathbf{p} \tag{2}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{C}_\mathbf{p}(t) = \mathbf{V}(\mathbf{C}_\mathbf{p}(t))\,, \tag{3}$$

and $\mathbf{V}$ is the vector field determining how points are displaced. In the case of traditional random jittering, we can consider $\mathbf{V}$ constant within each subdomain. The subdomain would typically be the pixel or, more generally, the Voronoi region around the point. As the streamline tracing method, we choose a 4th order Runge-Kutta method (RK4) [Runge 1895] which improved on taking an Euler step (see Figure 7); a deeper investigation into streamline tracing methods could yield further improvements.

The advantage of interpreting jittering as streamline tracing is that we can now consider ways to obtain vector fields that lead to better blue noise quality. We posit that *divergence free* vector fields would be advantageous since such vector fields do not contain sources or sinks which, respectively, repel and attract the points, leading to uneven distributions of points. In addition to being divergence-free, the vector field we seek must be noisy in the sense that it has no manifest anisotropy, since this would likely show in the jittered points. In other words, we claim that a noisy but divergence-free vector field can lead to jittered point sets where the points are regularly distributed while the anisotropy of the original grid is greatly reduced. See Section 2.4 for further discussion.

Unfortunately, the requirement that the vector field be divergence-free rules out some apparent candidates. For instance, neither the gradient field of a noise function, nor smoothly interpolating random vectors would produce a divergence-free vector field.

2.1 Curl Noise

There is, however, a method for generating noisy yet divergence-free vector fields. This method, due to Bridson et al. [2007], is known as *curl noise*. The curl, $\nabla\times$, of a 3D vector field, $\mathbf{\Phi} = [\phi_x, \phi_y, \phi_z]$, is

$$\nabla \times \mathbf{\Phi} = \left[\frac{\partial \phi_z}{\partial y} - \frac{\partial \phi_y}{\partial z}, \frac{\partial \phi_x}{\partial z} - \frac{\partial \phi_z}{\partial x}, \frac{\partial \phi_y}{\partial x} - \frac{\partial \phi_x}{\partial y}\right].$$

For 2D curl noise, we will assume that $\mathbf{\Phi} = [0, 0, f]$ for some noise function $f : \mathbb{R}^3 \to \mathbb{R}$. This means that

$$\nabla \times \mathbf{\Phi} = \left[\frac{\partial \phi_z}{\partial y}, -\frac{\partial \phi_z}{\partial x}, 0\right] = \left[\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x}, 0\right] \quad .$$

In other words, since $\mathbf{\Phi}$ is orthogonal to the xy-plane, the curl, $\nabla\times\mathbf{\Phi}$, lies in the xy-plane. Moreover, $\nabla \times \mathbf{\Phi}$, is known to be divergence-free because of the equality of mixed partial derivatives (Clairaut's

Theorem). Due to this property, flows that follow $\nabla \times \Phi$ are volume-preserving, and for our purposes, this ensures that the vector field is free of sinks and sources.

As an aside, we note that in 2D, the streamlines of $\nabla \times \Phi$ are simply the iso-contours of f, which means that any particle following such a trajectory, with sufficient numerical precision, will eventually return to its starting point.

2.2 Curl Noise Jittering

We can now formulate curl noise jittering operationally,

$$\mathrm{cnj}[N, s, t](\mathbf{x}) = \mathbf{C}_{\mathbf{x}}(t) \ , \tag{4}$$

where

$$\mathbf{C}_{\mathbf{x}}(0) = \mathbf{x} \ , \tag{5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{C}_{\mathbf{x}} = \left[\frac{\partial N(x/s, y/s)}{\partial y}, -\frac{\partial N(x/s, y/s)}{\partial x}\right] \ . \tag{6}$$

Thus, cnj is governed by three parameters: the noise function N, the noise scale s, and finally the time t for which we trace along the streamline.

2.3 Noise Functions

The choice of underlying noise function is essential when implementing curl noise and, by extension, curl noise jittering. It is important that the noise function is band limited, stationary, and isotropic [Perlin 1985; Lagae et al. 2010]; it is also a requirement that we can efficiently compute derivatives of the noise function.

We have tested three different noise functions that adhere to these requirements:

- Sparse convolution noise (SC) [Lewis 1984, 1989].
- Sum of sines noise (SoS) [Schachter 1981; Max 1981].
- Perlin noise [Perlin 1985, 2002].

For Perlin noise, we adapt an implementation by Iñigo Quilez[1] that includes computation of derivatives. Our implementation of sparse convolution noise is based on a version that can be evaluated locally [Frisvad and Wyvill 2007], and that has been described in a closed formula [Luongo et al. 2020] for which it is easy to find derivatives. Sine functions are easily differentiable. The principle of SoS is to construct an nD function by superposing nD waves. Max [1981] discusses how this can be used to construct models of water waves, and Schachter [1981] describes a similar model for narrow-band noise. In our formulation, SoS noise is based on waves of the form

$$w[a, \mathbf{d}, \phi](\mathbf{x}) = a \sin\left(\mathbf{d}^T\mathbf{x} + \phi\right), \tag{7}$$

where a is the amplitude and ϕ is the phase of the wave, while $\mathbf{d} = \frac{2\pi}{\lambda}\mathbf{n}$ is a frequency vector in which λ is the wavelength and $\mathbf{n}$ is the unit length direction of the wave. SoS noise is defined by

$$N(\mathbf{x}) = \sum_{(a,\mathbf{d},\phi)\in W} w[a, \mathbf{d}, \phi](\mathbf{x}), \tag{8}$$

where W is the set of parameter tuples. For each tuple, we generate the parameters by first sampling the random variables, $s \sim \mathcal{N}(0, \sigma)$ from a normal distribution and $\alpha, \phi \sim \mathcal{U}(0, 2\pi)$ from a uniform distribution. Next, we compute $\mathbf{d} = (\sigma + s)[\cos(\alpha), \sin(\alpha)]$ and $a = \exp(-s^2/\sigma^2)$. Thus, the amplitude is largest for waves with a frequency vector close to the mean. For the 2D experiments in this paper, we use $|W| = 256$ and $\sigma = 7$.

[1] https://iquilezles.org/articles/gradientnoise/

2.4 Quality Measures and Parameter Selection

Curl noise jittering is motivated by the premise that displacing the points of a lattice along a curl noise vector field will imbue the points with blue noise properties. This hypothesis is based on the observation that flow along a curl noise vector field is volume-preserving because the field is divergence-free. While volume preservation does not imply preservation of point distances, it does guarantee that there are no sinks or sources which, respectively, attract or repel the points when transported along the field for a short time. This leads us to assume that the point spacing of the lattice would be preserved to some extent for (short) flows along the vector field. Yet, the regularity of the lattice, and hence anisotropy, is diminished because the vector field is a noise function.

However, it appears hard to provide a formal argument. Instead, we offer a quantitative metric for blue noise properties. This metric is used to assess the quality of CNJ as well as competing methods and to tune the parameters of our method. Specifically, it is important to select an appropriate time step (as discussed above) and noise scale. If the noise scale is extremely coarse, the lattice is locally translated, and the lattice structure remains visible. In contrast, if the scale is too fine (i.e., the frequency is too high), the method devolves to random jittering.

Our quantitative analysis is based on the point set analysis framework (PSA) [Schlömer and Deussen 2011; Heck et al. 2013], which is commonly used in related work, e.g. [Ahmed et al. 2022]. We briefly recall the different metrics in the following. We refer the reader to Heck et al. [2013] for further details.

The *power spectrum* of a signal, such as a point process, is the Fourier transform of the autocorrelation function of the signal, and it can also be computed directly as the absolute square of the Fourier transform of the signal. Useful statistics can be derived by partitioning the power spectrum into concentric annuli of width Δ, and then averaging the spectrum samples within each annulus ξ_r of central radius r [Schlömer and Deussen 2011]. The *radially averaged power spectrum* $P(\xi_r)$ measures the average of the power spectrum in some annulus ξ_r. The *anisotropy* is defined by $a(\xi_r) = \frac{V^2(\xi_r)}{P(\xi_r)}$, where $V^2(\xi_r)$ is the sample variance of the power spectrum in some annulus ξ_r. Lower anisotropy values indicate that the power spectrum is close to being radially symmetric; thus, the resulting point process is close to isotropic. We define the power-weighted mean anisotropy,

$$\bar{a} = \sum_{r>0} a(\xi_r)P^2(\xi_r) = \sum_{r>0} V^2(\xi_r)P(\xi_r) \ , \tag{9}$$

where the first annulus $r = 0$ containing the zero frequency is excluded. The largest frequency v so that the average energy in the power spectrum up to v stays below 0.1 is called the effective Nyquist frequency and denoted v_{eff}. Intuitively, frequencies below v_{eff} can be sampled and reconstructed with little error, while frequencies above lead to aliasing. Now, we define the quality of a point set as the ratio of the effective Nyquist frequency to mean

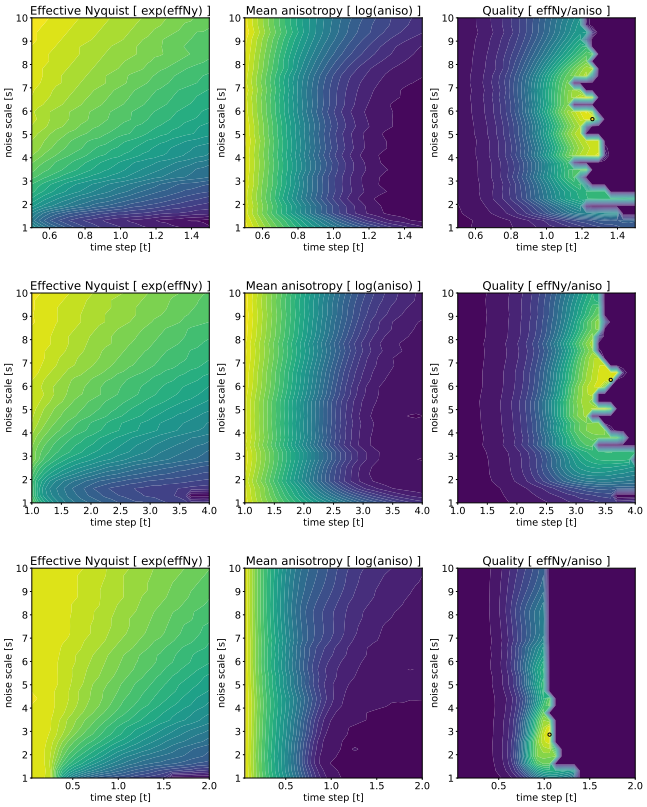

Figure 3: From left to right, each row shows the effective Nyquist frequency, the anisotropy, and the derived quality measure, which is simply their ratio, for curl noise jittering based on sparse convolution noise (top), sum-of-sines noise (middle), and Perlin noise (bottom).

anisotropy,

$$q = \begin{cases} \frac{v_{\text{eff}}}{\bar{a}} & r_j < \delta \\ 0 & r_j \geq \delta \end{cases} , \tag{10}$$

where δ is the edge length of the regular lattice, i.e., the distance between two neighboring points. Thus, q is proportional to the effective Nyquist frequency and inversely proportional to the average anisotropy – unless the maximum displacement due to jittering exceeds the edge length of the lattice. Assigning $q = 0$ for $r_j \geq \delta$ is important for the implicit generation of P^*: the greater r_j, the more grid points could beget the closest jittered point. If $r_j < \delta$, q balances the two requirements that the jittered points are well separated (v_{eff} large) and isotropy ($\bar{a}$ small).

In Figure 3, v_{eff}, $\bar{a}$, and q are plotted as a function of time step t and noise scale s for the three noise functions considered. Based on the underlying (30×30) grids of quality values, the maximum was found for each function, and the results are presented in Table 1. Perlin noise has the highest quality and is less computationally demanding than the other noise functions. Hence, it was selected for our 2D experiments. Note that the scale s is the ratio of the side length of the Perlin noise grid to the distance between lattice points.

Table 1: Effective Nyquist limit (v_{eff}), anisotropy ($\bar{a}$), and quality (q) for different noise functions and selected pairs of parameters, time step t and noise scale s. The best values are in boldface.

Noise type	t	s	v_{eff}	$\bar{a}$	q
SC	1.25862	5.65517	**0.745**	81.8	0.0091
SoS	3.58621	6.27586	0.7	72.6	0.0096
Perlin	1.05862	2.86207	0.705	**71.9**	**0.0098**

2.5 Implementation

The direct 2D method for curl noise jittering was implemented in Python for all three noise functions. Our code interfaces to the PSA library[2], which is used for the quality metric. Our code also interfaces to the CCVT library[3] and the dyadic nets[4] library, both of which were used for comparison. The performance of the direct methods was evaluated on a single Apple M1 core.

The implicit 2D curl noise jittering method is implemented in GLSL on the ShaderToy platform. This implementation employs only Perlin noise and runs at 2.04 ms per frame in 1080p resolution when using an NVIDIA RTX 3090 GPU. We found that in practice, we do not need to check the distances between lattice points and query points if we jitter all neighbors up to the second order.

2.6 Comparisons

Like all methods based on jittering, CNJ places each point without considering the positions of nearby points and can hence be evaluated implicitly. We compare CNJ to two other methods based on jittering and three methods that consider the local neighborhood. The jittering-based methods are clearly the most similar to ours, while methods that consider neighboring points can provide very high quality. We also included *iterative* curl noise jittering ICNJ in this comparison. The iterative method applies jittering with a time step that is halved in each iteration (and we used 64 iterations). The noise scale is the same for all iterations, but the noise is different each iteration (since we add an offset).

From the category of jittering based methods, we compare against Kensler's correlated multi-jittered (CMJ) sampling approach [Kensler 2013]. The principle is that each row and each column is jittered with a sequence of perpendicular displacements. Since the displacements are identical for all rows (columns), neighbors along each row (column) are never closer than the edge length of the grid. We also compare against the smoothed jittering (SJ) of a triangular lattice proposed by Klassen et al. [2000].

From the category of non jittering-based methods, we compare against the Blue Nets (BN) method [Ahmed and Wonka 2021], capacity-constrained Voronoi tessellations (CCVT) [Balzer et al. 2009] and the fast Poisson disk sampling (PDS) method proposed by Robert Bridson [2007].

For each of these seven methods, we show the point clouds and their associated power and radial spectra in Figure 4. We also computed our quality measure, $D*$, the L2 norm of star-discrepancy [Wang and Sloan 2008], and timings, all averaged over 100 runs.

[2] https://github.com/nodag/psa
[3] https://github.com/michaelbalzer/ccvt
[4] http://abdallagafar.com/publications/dyadic-nets/

Table 2: Average results for 100 runs of the tested methods. The statistics shown are: effective Nyquist, v_{eff}, anisotropy, $\bar{a}$, quality, q, discrepancy, $D*$, and time. Methods above the double line can be evaluated implicitly, and methods below cannot. The best values within each of these two categories are in boldface.

Method	v_{eff} ↑	$\bar{a}$ ↓	q ↑	D* ↓	time(s) ↓	language
CNJ	0.7	**71.81**	0.0097	**0.002**	0.0621	Python
ICNJ	0.74	72.81	**0.01**	0.0021	2.22	Python
CMJ	**0.82**	960.6	0.00086	0.0023	**0.00218**	Python
SJ	0.44	442.7	0.001	0.0034	0.00788	Python
BN	**0.83**	88.27	0.0095	**0.00083**	47.9	C++
CCVT	**0.83**	78.85	**0.011**	0.002	20.8	C++
PDS	0.26	**70.63**	0.0036	0.0047	**0.332**	Python

These results are shown in Table 2, which also includes the programming language used.

CCVT attains the highest quality. This is unsurprising since this method uses information about the neighborhood to optimize positions. ICNJ and regular CNJ are second and third, respectively. In most cases, the slightly increased quality of ICNJ would not justify the increased computational cost. Blue Nets is only in fourth place due to a comparatively high anisotropy. However, the discrepancy of the Blue Nets method is less than half that of its closest competitor. Low discrepancy is a trait that appears to be shared by $(0, m, 2)$ nets in general; such nets can be generated very quickly using the Netshuffle algorithm of Ahmed and Wonka [2021], albeit dyadic nets do not have blue noise properties unless they are optimized.

CNJ and ICNJ are the only methods that can be evaluated implicitly (all based on jittering) with a low anisotropy. In general, isotropy is challenging to achieve with jittering since a regular lattice exhibits high anisotropy. It is perhaps more surprising that while Poisson disk sampling has the lowest anisotropy of all methods, its v_{eff} is also very low, and the overall quality is inferior to CNJ.

2.7 Applications and Extensions

Steven Worley [1996] proposed a method for texturing based on distances to randomly scattered points, which can be used to model the appearance of objects with cracked or segmented surfaces. In Worley's original formulation, points were computed on the fly by hashing the value of a grid cell to a random point within each cell. With curl noise jittering, we can produce a more even distribution of points leading, in turn, to a more regular cellular structure than in the original method, as shown in Figure 5. All images in Figure 5 were generated in ShaderToy using the implicit method. In the top row, the color is based purely on the distance to the closest point. In the bottom row, the difference between the distance to the closest and second closest point is used to generate the line color. The position of the closest jittered point is hashed to cell color.

We implemented the direct method in a Blender script to generate a distribution of boxes, see Figure 6 (right). A similar distribution generated by using Blender's built-in Poisson disk sampling is shown on the left. While the differences are subtle, our method seems to produce a slightly more even distribution.

3 CURL NOISE JITTERING IN 3D

An important difference in 3D is that we need a vector-valued noise (a 3D vector field is needed to take the curl in 3D). In the case of Perlin noise, this means evaluating the function three times with different offsets. For the sum of sines, the three evaluations with different offsets are easily performed in the same sum. Finally, since sparse convolution noise is based on the convolution of randomly placed random impulses, we just need to use random vector-valued impulses, which entails little extra cost.

The initial grid to use for curl noise jittering depends on the desired packing density of the point distribution. If we start from a regular octahedral grid (body centered cubic lattice [Theußl et al. 2001], obtained by inserting a point at the center of each voxel of a cubic grid and using the faces of the cells in the cubic grid as the bases of regular octahedra), we get a denser point distribution than if we start from a cubic grid. The properties of the noise functions also come into play when we work in 3D. We ran a parameter study by rendering out 3D point distributions as collections of spheres. We visually inspected the results to qualitatively assess whether the findings in 2D generalize to 3D. The behavior with different parameters is a bit different in 3D. Perlin noise and SoS seem to more strictly require using the RK4 method to avoid regularities. SoS can occasionally exhibit ripples in the point distribution, but this seems avoidable with well-chosen parameters. Sparse convolution noise is, in that sense, easier to find suitable parameters for in 3D, and it seems to perform reasonably well with an Euler step. If combined with RK4, sparse convolution noise becomes more expensive to evaluate than the other methods.

3.1 Implementation

We implemented 3D curl noise jittering in CUDA for NVIDIA OptiX (v7.6) [Parker et al. 2010] and render metals by path tracing of specular materials with a complex index of refraction [Pharr et al. 2023]. Sphere tracing [Hart 1996] is used for ray-surface intersection in a signed distance field (SDF). We rendered images using the previously mentioned RTX 3090 GPU. The rendering time for a one sample per pixel (1spp) frame of resolution 720×720 is provided for different examples in Figures 8 and 9.

3.2 Applications

The ability of our method to produce a point set with blue noise properties makes it suitable for approximate modeling of the jammed hard-particle packings observed in granular materials [Torquato and Stillinger 2010; Meng et al. 2015]. As an example, we made a bunny out of tiny spheres and rendered it with the camera at different distances from the object, see Figure 8. We used a regular octahedral lattice as our initial grid, with a total of around 1.3 billion spheres in the entire volume of the bunny. Our implicit method was used with the sum of sines (and RK4) to get an SDF for a stochastic sphere packing with little overlapping of the spheres.

The production of stochastic foam by a random bubble-forming chemical process results in material structures represented well by Voronoi cells [Bogunia et al. 2022]. Based on our discussion of the Worley method (see Section 2.7), we believe our method can be used advantageously for the 3D modeling of foams. As an example, we modeled the structure of open-cell aluminum and

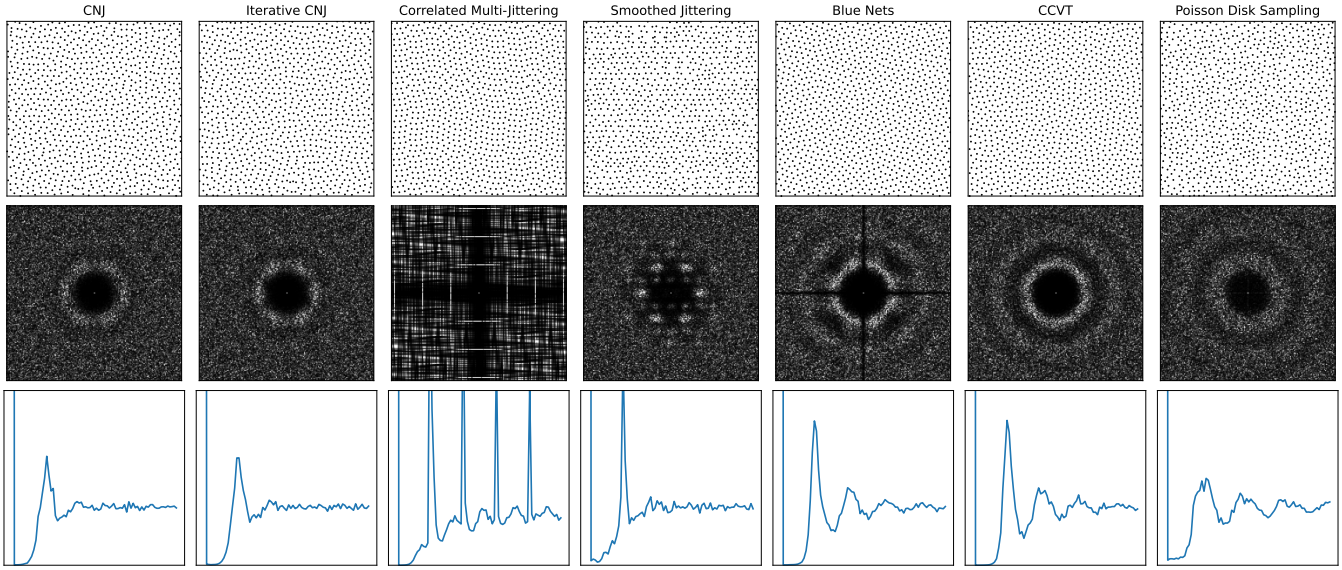

Figure 4: Comparison of our method (far left) to the methods discussed in Section 2.6. We show point sets (top) as well as the power spectrum (zoomed in $\times 10$, middle) and the radial spectrum for each point set (bottom).

copper foams (see Figure 9). For the rendering of the metals, we use available measured complex indices of refraction of oxidized aluminum [Querry 1985] and copper [Glassner 1995]. We use the Worley method and find the distances to the nearest points in our point distribution by inspecting the positions of the grid nodes in the first-order neighborhood after being curl noise jittered. Let d_i, $i = 1, \ldots, 4$, denote distances from the point of interest $\mathbf{x}$ in the volume with d_1 being the distance to the closest point and $d_1 < d_2 < d_3 < d_4$. For the more porous aluminum foam, we use a regular grid and the following formula for conversion to a non-Euclidean signed distance field:

$$\begin{aligned} \mathrm{sdf}(\mathbf{x}) \quad = \quad & (d_2(\mathbf{x}) - d_1(\mathbf{x}))^\alpha + (d_3(\mathbf{x}) - d_2(\mathbf{x}))^\alpha \\ & + (d_3(\mathbf{x}) - d_1(\mathbf{x}))^\alpha - \varepsilon\,. \end{aligned} \tag{11}$$

The parameters in this model are α, which is related to the enclosedness of the cavities, and ε, which is the thickness of the geometric features. We used $\alpha = 1.55$ and $\varepsilon = 0.1$ for the aluminum foam. In the curl noise jittering, we used sparse convolution noise with an Euler step for this material.

For the copper foam, we use a *trabeculum* formula by Fabrice Neyret[5]. The non-Euclidean signed distance field is then

$$\mathrm{sdf}(\mathbf{x}) = \left(\frac{1}{d_3(\mathbf{x}) - d_1(\mathbf{x})} + \frac{1}{d_4(\mathbf{x}) - d_1(\mathbf{x})} \right)^{-1} - \varepsilon\,. \tag{12}$$

In this case, we used $\varepsilon = 0.06$ and CNJ with the octahedral grid as the initial grid (as for the bunny). Figure 10 compares the CNJ of the cubic grid versus the octahedral grid. Visual inspection of the foams as compared with foams obtained using random jitter sampling of one point per voxel in the grid (Figure 9) confirms that our curl noise jittering effectively reduces clustering of points and thus achieves a better model of the real metal foams.

[5]https://www.shadertoy.com/view/MlB3Wt

4 DISCUSSION AND LIMITATIONS

We have demonstrated that creating point sets with blue noise properties is possible using a simple jittering approach. Going forward, this type of point set can more readily be used in real-time applications or where large domains need to be sampled. Of course, applications that need a blue noise sampling large enough that either generating or storing the points would be a limiting factor will benefit the most. However, we note that our method is simple to implement and can be used as a drop-in replacement for applications that already employ jittering.

CNJ works in both 2D and 3D, and while we have focused on the 2D case in our analysis, our 3D results indicate that CNJ can be useful for procedural materials with very fine granularity where precomputing blue noise point sets would lead to prohibitive memory consumption. In the future, we are also interested in extending curl noise jittering to other domains. For instance, time-varying noise functions could be useful in certain scenarios.

Our work presupposes that a lattice is given. This does not preclude adaptive curl noise jittering, where the point density depends on, for instance, an underlying image's intensity. However, this would require that the lattice adapts to the intensity, which is not a part of our investigation but might be an avenue for future work. Finally, we would also like to explore the use of other noise functions and streamline tracing methods.

ACKNOWLEDGMENTS

This project was partially supported by the Villum Foundation through the Villum Investigator Project InnoTop.

REFERENCES

Abdalla G. M. Ahmed, Markus Hadwiger, Mikhail Skopenkov, and Peter Wonka. 2023. Analysis and synthesis of digital dyadic sequences. https://doi.org/10.48550/arXiv.2306.06925 arXiv:2306.06925 [cs.GR]

Abdalla G. M. Ahmed, Hui Huang, and Oliver Deussen. 2015. AA patterns for point sets with controlled spectral properties. *ACM transactions on graphics* 34, 6 (2015), 212:1–212:8. https://doi.org/10.1145/2816795.2818139

Abdalla G. M. Ahmed, Till Niese, Hui Huang, and Oliver Deussen. 2017. An adaptive point sampler on a regular lattice. *ACM Transactions on Graphics* 36, 4 (2017), 138:1–138:13. https://doi.org/10.1145/3072959.3073588

Abdalla G. M. Ahmed, Jing Ren, and Peter Wonka. 2022. Gaussian blue noise. *ACM Transactions on Graphics* 41, 6 (2022), 260:1–260:15. https://doi.org/10.1145/3550454.3555519

Abdalla G. M. Ahmed and Peter Wonka. 2020. Screen-space blue-noise diffusion of Monte Carlo sampling error via hierarchical ordering of pixels. *ACM Transactions on Graphics* 39, 6 (2020), 244:1–244:15. https://doi.org/10.1145/3414685.3417881

Abdalla G. M. Ahmed and Peter Wonka. 2021. Optimizing dyadic nets. *ACM Transactions on Graphics* 40, 4 (2021), 141:1–141:17. https://doi.org/10.1145/3450626.3459880

Michael Balzer, Thomas Schlömer, and Oliver Deussen. 2009. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics* 28, 3 (2009), 86:1–86:8. https://doi.org/10.1145/1531326.1531392

Lukas Bogunia, Stefan Buchen, and Kerstin Weinberg. 2022. Microstructure characterization and stochastic modeling of open-cell foam based on μCT-image analysis. *GAMM-Mitteilungen* 45, 3-4 (2022), e202200018. https://doi.org/10.1002/gamm.202200018

Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *SIGGRAPH 2007 Sketches*. 22. https://doi.org/10.1145/1278780.1278807

Robert Bridson, Jim Houriham, and Marcus Nordenstam. 2007. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics* 26, 3 (2007), 46:1–46:3. https://doi.org/10.1145/1276377.1276435

Kenneth Chiu, Changyaw Wang, and Peter Shirley. 1994. Multi-jittered sampling. In *Graphics Gems*. Elsevier, 370–374.

Per Christensen, Andrew Kensler, and Charlie Kilpatrick. 2018. Progressive multi-jittered sample sequences. *Computer Graphics Forum* 37, 4 (2018), 21–33. https://doi.org/10.1111/cgf.13472

Robert L Cook. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics* 5, 1 (1986), 51–72. https://doi.org/10.1145/7529.8927

Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. 2012. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics* 18, 6 (2012), 914–924. https://doi.org/10.1109/TVCG.2012.34

Sabrina Dammertz. 2009. *Rank-1 Lattices in Computer Graphics*. Ph. D. Dissertation. Universität Ulm. https://doi.org/10.18725/OPARU-1068

Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Transactions on Graphics* 31, 6 (2012), 171:1–171:11. https://doi.org/10.1145/2366145.2366190

Daniel Dunbar and Greg Humphreys. 2006. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics* 25, 3 (2006), 503–508. https://doi.org/10.1145/1141911.1141915

Mohamed S. Ebeida, Andrew A. Davidson, Anjul Patney, Patrick M. Knupp, Scott A. Mitchell, and John D. Owens. 2011. Efficient maximal Poisson-disk sampling. *ACM Transactions on Graphics* 30, 4 (2011), 49:1–49:12. https://doi.org/10.1145/2010324.1964944

Raanan Fattal. 2011. Blue-noise point sampling using kernel density model. *ACM Transactions on Graphics* 30, 4 (2011), 48:1–48:12. https://doi.org/10.1145/2010324.1964943

Jens Feder. 1980. Random sequential adsorption. *Journal of Theoretical Biology* 87, 2 (1980), 237–254. https://doi.org/10.1016/0022-5193(80)90358-6

L. Fejes. 1942. Über die dichteste Kugellagerung. *Mathematische Zeitschrift* 48, 1 (1942), 676–684. https://doi.org/10.1007/BF01180035

Robert W. Floyd and Louis Steinberg. 1976. An adaptive algorithm for spatial gray-scale. *Proceedings of the Society of Information Display* 17, 2 (1976), 75–77.

Jeppe Revall Frisvad and Geoff Wyvill. 2007. Fast high-quality noise. In *GRAPHITE 2007*. 243–248. https://doi.org/10.1145/1321261.1321305

Andrew S. Glassner. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann. Two volumes.

John H. Halton. 1964. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM* 7, 12 (1964), 701–702. https://doi.org/10.1145/355588.365104

J. M. Hammersley and D. C. Handscomb. 1964. *Monte Carlo Methods*. Chapman and Hall.

John C. Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545. https://doi.org/10.1007/s003710050084

Daniel Heck, Thomas Schlömer, and Oliver Deussen. 2013. Blue noise sampling with controlled aliasing. *ACM Transactions on Graphics* 32, 3 (2013), 25:1–25:12. https://doi.org/10.1145/2487228.2487233

Stefan Hiller, Oliver Deussen, and Alexander Keller. 2001. Tiled blue noise samples. In *Vision, Modeling, and Visualization (VMV)*.

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. 2015. Blue noise sampling using an SPH-based method. *ACM Transactions on Graphics* 34, 6 (2015), 211:1–211:11. https://doi.org/10.1145/2816795.2818102

Andrew Kensler. 2013. *Correlated Multi-Jittered Sampling*. Technical Memo 13-01. Pixar. https://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf

R Victor Klassen. 2000. Filtered jitter. *Computer Graphics Forum* 19, 4 (2000), 223–230. https://doi.org/10.1111/1467-8659.00459

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics* 25, 3 (2006), 509–518. https://doi.org/10.1145/1179352.1141916

Ares Lagae and Philip Dutré. 2005. A procedural object distribution function. *ACM Transactions on Graphics* 24, 4 (2005), 1442–1461. https://doi.org/10.1145/1095878.1095888

Ares Lagae and Philip Dutré. 2008. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum* 27, 1 (2008), 114–129. https://doi.org/10.1111/j.1467-8659.2007.01100.x

Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, and Matthias Zwicker. 2010. A survey of procedural noise functions. *Computer Graphics Forum* 29, 8 (2010), 2579–2600. https://doi.org/10.1111/j.1467-8659.2010.01827.x

Daniel L. Lau, Robert Ulichney, and Gonzalo R. Arce. 2003. Blue and green noise halftoning models. *IEEE Signal Processing Magazine* 20, 4 (2003), 28–38. https://doi.org/10.1109/MSP.2003.1215229

John-Peter Lewis. 1984. Texture synthesis for digital painting. *Computer Graphics (SIGGRAPh '84)* 18, 3 (1984), 245–252. https://doi.org/10.1145/800031.808605

John-Peter Lewis. 1989. Algorithms for solid noise synthesis. *Computer Graphics (SIGGRAPH '89)* 23, 3 (1989), 263–270. https://doi.org/10.1145/74333.74360

Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

Andrea Luongo, Viggo Falster, Mads Brix Doest, Macarena Mendez Ribo, Eyþór Rúnar Eiríksson, David B Pedersen, and Jeppe Revall Frisvad. 2020. Microstructure control in 3D printing with digital light processing. *Computer Graphics Forum* 39, 1 (2020), 347–359. https://doi.org/10.1111/cgf.13807

Jonàs Martínez, Jérémie Dumas, and Sylvain Lefebvre. 2016. Procedural Voronoi foams for additive manufacturing. *ACM Transactions on Graphics* 35, 4 (2016), 44:1–44:12. https://doi.org/10.1145/2897824.2925922

Nelson L Max. 1981. Vectorized procedural models for natural terrain: Waves and islands in the sunset. *Computer Graphics (SIGGRAPH '81)* 15, 3 (1981), 317–324. https://doi.org/10.1145/965161.806820

Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. 2015. Multi-scale modeling and rendering of granular materials. *ACM Transactions on Graphics* 34, 4 (2015), 49:1–49:13. https://doi.org/10.1145/2766949

Victor Ostromoukhov. 2001. A simple and efficient error-diffusion algorithm. In *SIGGRAPH 2001*. 567–572. https://doi.org/10.1145/383259.383326

Victor Ostromoukhov. 2007. Sampling with polyominoes. *ACM Transactions on Graphics* 26, 3 (2007), 78:1–78:6. https://doi.org/10.1145/1276377.1276475

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics* 23, 3 (2004), 488–495. https://doi.org/10.1145/1015706.1015750

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics* 29, 4 (2010), 66:1–66:13. https://doi.org/10.1145/1778765.1778803

Loïs Paulin, David Coeurjolly, Jean-Claude Iehl, Nicolas Bonneel, Alexander Keller, and Victor Ostromoukhov. 2021. Cascaded Sobol' sampling. *ACM Transactions on Graphics* 40, 6 (2021), 275:1–275:13. https://doi.org/10.1145/3478513.3480482

Ken Perlin. 1985. An image synthesizer. *Computer Graphics (SIGGRAPH '85)* 19, 3 (1985), 287–296. https://doi.org/10.1145/325165.325247

Ken Perlin. 2002. Improving noise. In *SIGGRAPH 2002*. 681–682. https://doi.org/10.1145/566570.566636

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically Based Rendering: From Theory to Implementation* (fourth ed.). MIT Press.

Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. 2015. Variance analysis for Monte Carlo integration. *ACM Transactions on Graphics* 34, 4 (2015), 1–14. https://doi.org/10.1145/2766930

Hongxing Qin, Yi Chen, Jinlong He, and Baoquan Chen. 2017. Wasserstein blue noise sampling. *ACM Transactions on Graphics* 36, 5 (2017), 168:1–168:13. https://doi.org/10.1145/3119910

Marvin R. Querry. 1985. *Optical Constants*. Technical Report CRDC-CR-85034. University of Missouri.

Carl Runge. 1895. Über die numerische Auflösung von Differentialgleichungen. *Math. Ann.* 46, 2 (1895), 167–178. https://doi.org/10.1007/BF01446807

Bruce Schachter. 1981. Long crested wave models. In *Image Modeling*. Elsevier, 327–341. https://doi.org/10.1016/B978-0-12-597320-5.50024-3

Thomas Schlömer and Oliver Deussen. 2011. Accurate spectral analysis of two-dimensional point sets. *Journal of Graphics, GPU, and Game Tools* 15, 3 (2011), 152–160. https://doi.org/10.1080/2151237X.2011.609773

Il'ya Meerovich Sobol'. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 7, 4 (1967), 784–802. https://doi.org/10.1016/0041-5553(67)90144-9

Thomas Theußl, Torsten Möller, and Meister Eduard Gröller. 2001. Optimal regular volume sampling. In *Proceedings of Visualization 2001 (VIS'01)*. IEEE, 91–98 + 546. https://doi.org/10.1109/VISUAL.2001.964498

Salvatore Torquato and Frank H. Stillinger. 2010. Jammed hard-particle packings: From Kepler to Bernal and beyond. *Reviews of Modern Physics* 82, 3 (2010), 2633–2672. https://doi.org/10.1103/RevModPhys.82.2633

Robert A Ulichney. 1988. Dithering with blue noise. *Proc. IEEE* 76, 1 (1988), 56–79. https://doi.org/10.1109/5.3288

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 Transactions on Graphics* 33, 4 (2014), 56:1–56:11. https://doi.org/10.1145/2601097.2601107

Xiaoqun Wang and Ian H. Sloan. 2008. Low discrepancy sequences in high dimensions: How well are their projections distributed? *J. Comput. Appl. Math.* 213, 2 (2008), 366–386. https://doi.org/10.1016/j.cam.2007.01.005

Li-Yi Wei. 2008. Parallel Poisson disk sampling. *ACM Transactions on Graphics* 27, 3 (2008), 20:1–20:9. https://doi.org/10.1145/1360612.1360619

Steven Worley. 1996. A cellular texture basis function. In *SIGGRAPH '96*. 291–294. https://doi.org/10.1145/237170.237267

Dong-Ming Yan, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka. 2015. A survey of blue-noise sampling and its applications. *Journal of Computer Science and Technology* 30, 3 (2015), 439–452. https://doi.org/10.1007/s11390-015-1535-0

Cem Yuksel. 2015. Sample elimination for generating Poisson disk sample sets. *Computer Graphics Forum* 34, 2 (2015), 25–32. https://doi.org/10.1111/cgf.12538

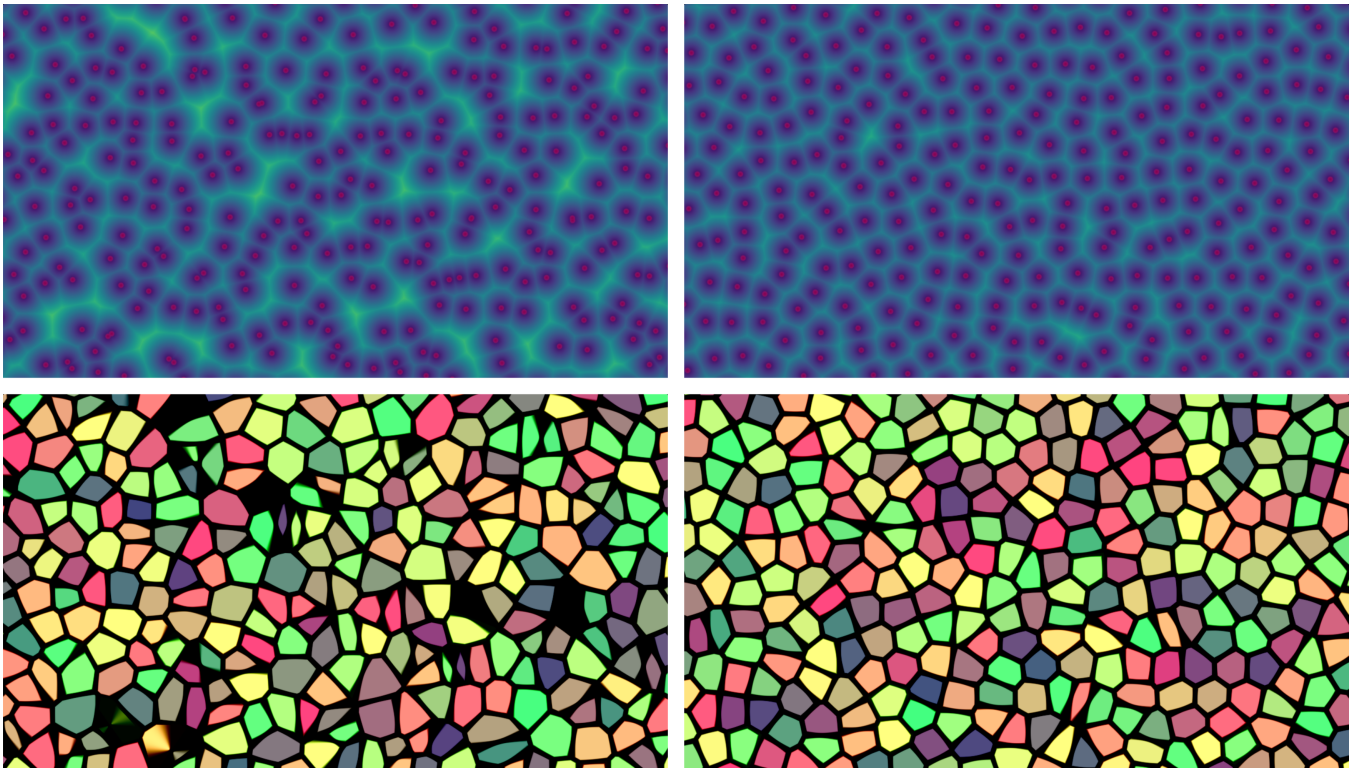

Figure 5: Two different Worley noise textures generated using random displacements (left) and CNJ (right).

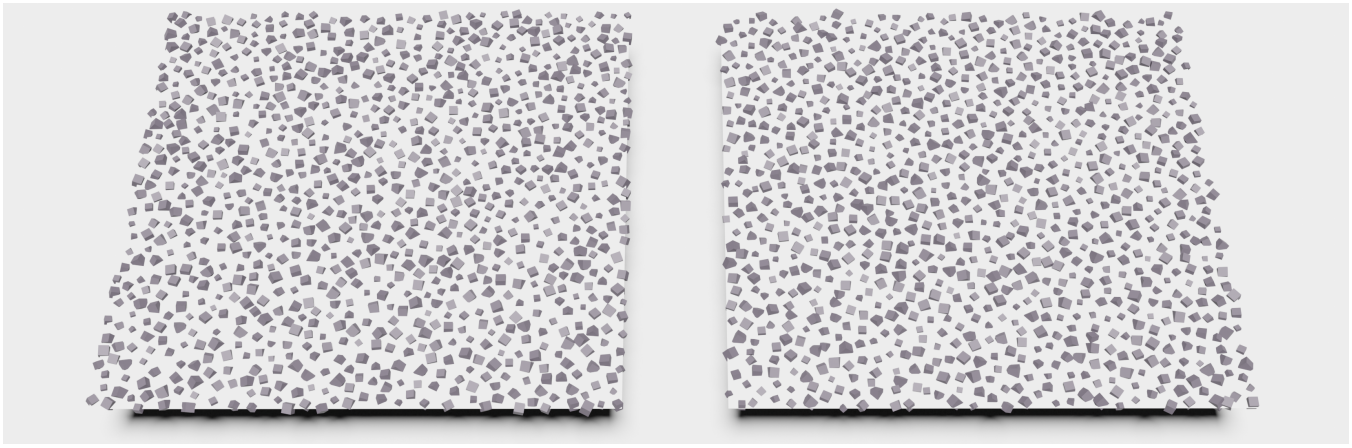

Figure 6: In this example, randomly oriented and scaled boxes were placed on a square using Blender's built-in Poisson disk sampling (left) and our CNJ method (right).

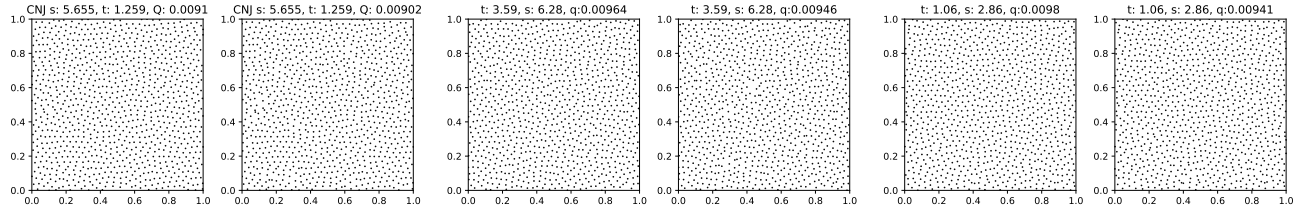

Figure 7: CNJ realizations corresponding to SC noise (left), SoS noise (middle), and Perlin noise (right) for the s and t values shown. For each pair, the left is with RK4 tracing and the right with Euler stepping.

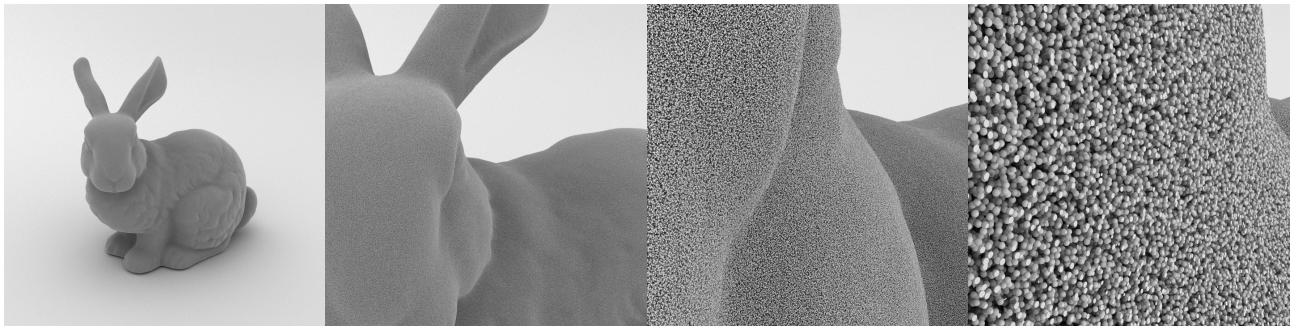

Figure 8: Moving closer and closer to a seemingly dusty version of the Stanford bunny (http://graphics.stanford.edu/data/3D-scanrep/). This illustrates our ability to distribute a very large number of points using CNJ w. SoS and RK4. Rendering time for a 1spp frame (left to right): 2.5 s, 4.5 s, 5.4 s, 6.4 s. Rendered here with 3000 samples per pixel.

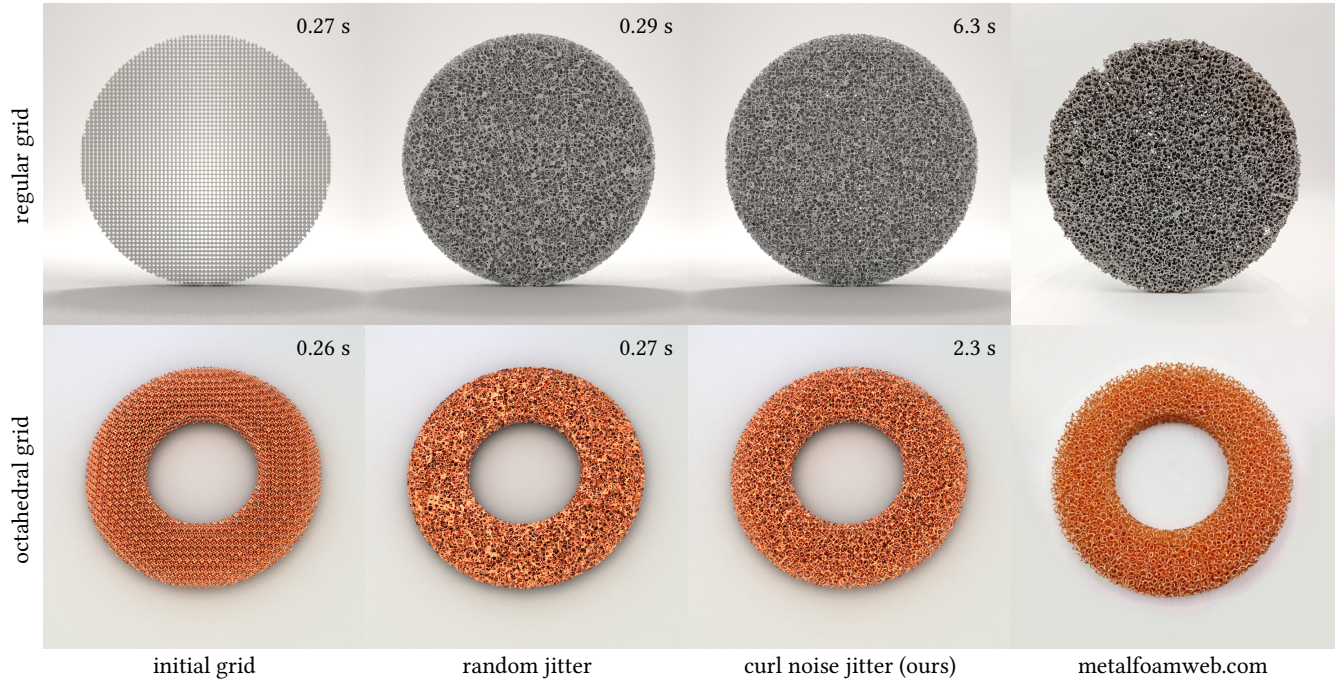

Figure 9: Modeling open cell aluminum foam (top) and copper foam (bottom). We used a cubic grid for the aluminum foam due to its higher porosity and an octahedral grid for the copper foam. The time in the corner is for a 1spp frame. The blue noise properties of our method helps it better avoid point clustering issues. The photos in the rightmost column, courtesy of Beihai Composite Materials (https://www.metalfoamweb.com/), provide some intuition on the appearance of real metal foams.

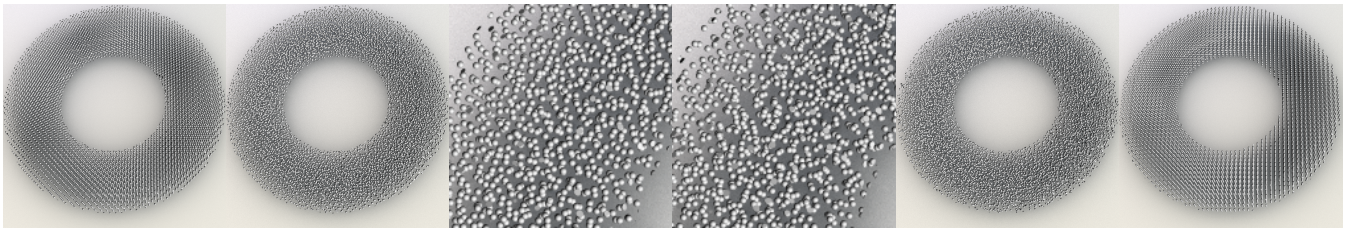

Figure 10: To indicate the difference between the cubic grid (□) and the octahedral grid (8Δ), we here show the point distributions for the copper foam scene when using the two different types of initial grid. We tried to obtain a similar result with both grids. Due to the larger distance between the nodes in the cubic grid, we made the grid smaller, applied a larger time step, and decreased the scale in CNJ. The distribution of points is more uneven for the cubic grid.