Multi-Dimensional Procedural Wave Noise

Pascal Guehl, Rémi Allègre, Guillaume Gilet, Basile Sauvage, Marie-Paule Cani, Jean-Michel Dischler

Multi-Dimensional Procedural Wave Noise

PASCAL GUEHL, LIX, Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, France
RÉMI ALLÈGRE, ICube, Université de Strasbourg, CNRS, France
GUILLAUME GILET, Université de Sherbrooke, Canada
BASILE SAUVAGE, ICube, Université de Strasbourg, CNRS, France
MARIE-PAULE CANI, LIX, Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, France
JEAN-MICHEL DISCHLER, ICube, Université de Strasbourg, CNRS, France

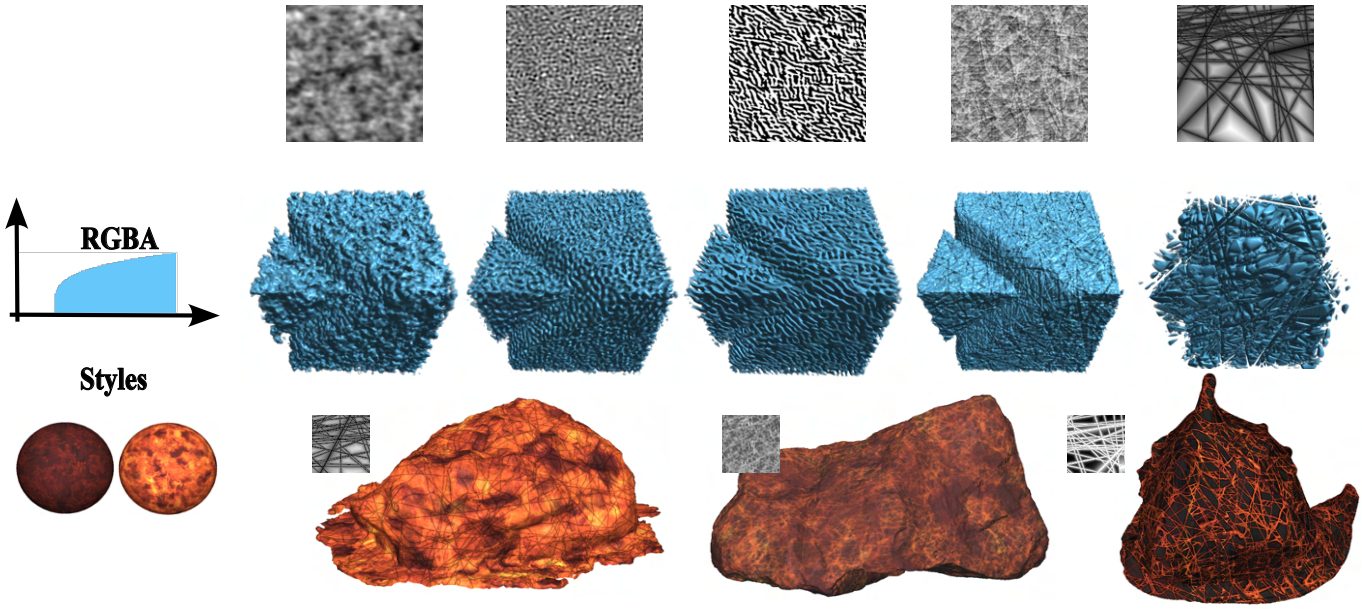

Fig. 1. Wave noise is a new procedural noise that encompasses sparse convolution procedural noises like resp. Lewis, Gabor, and Phasor noises providing more efficient computation in higher dimensions (top row, resp. left to center). It also introduces various novel patterns (top row, two rightmost images), including a new cellular noise distinct from Worley's cellular approach (right most). Retaining essential properties (infinite, resolution independent, compact [1KB per 3D example], fast to compute, and GPU-friendly), it supports diverse applications in Computer Graphics. Two examples are demonstrated: modeling volumetric materials with RGBA transfer functions (middle row), and surface materials mapped on arbitrary geometry with no UV coordinates using style transfer functions (bottom row). Transfer functions are on the left.

While precise spectral control can be achieved through sparse convolution, corresponding state of the art noise models are typically too expensive for solid noise. We introduce an alternative, wave-based procedural noise model, fast enough to be used in any dimension. We express the noise in the spectral domain and then apply an inverse Fourier transform (FT), requiring the computation of a multidimensional integral. Our contribution is a novel, efficient way to perform this computation, using a sum of precomputed complex-valued hyperplanar wave-functions, oriented in random directions. We show that using suitable wave profiles and combination operators, our model is able to extend to 3D a number of Gaussian and non-Gaussian noises, including Gabor, by-example and Phasor noises, as well as generate novel cellular noises. Our versatile and controllable solid noise model is very compact, a key feature for complex power spectrum and animated noises. We illustrate this through the design of 2D, 3D, and 3D+t materials using color, transparency and style transfer functions.

CCS Concepts: • **Computing methodologies** → **Texturing**.

Additional Key Words and Phrases: Procedural noise

ACM Reference Format:
Pascal Guehl, Rémi Allègre, Guillaume Gilet, Basile Sauvage, Marie-Paule Cani, and Jean-Michel Dischler. 2025. Multi-Dimensional Procedural Wave

Authors' Contact Information: Pascal Guehl, LIX, Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, France, pascal.guehl@polytechnique.edu; Rémi Allègre, ICube, Université de Strasbourg, CNRS, France, remi.allegre@unistra.fr; Guillaume Gilet, Université de Sherbrooke, Canada, Guillaume.Gilet@USherbrooke.ca; Basile Sauvage, ICube, Université de Strasbourg, CNRS, France, sauvage@unistra.fr; Marie-Paule Cani, LIX, Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, France, marie-paule.cani@polytechnique.edu; Jean-Michel Dischler, ICube, Université de Strasbourg, CNRS, France, dischler@unistra.fr.

ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3730928

Noise. *ACM Trans. Graph.* 44, 4 (August 2025), 15 pages. https://doi.org/10.1145/3730928

1 Introduction

Procedural noise has long been a key element in Computer Graphics pipelines [Ebert et al. 2003]. Noise functions aim to define resolution-free, unbounded stochastic fields, then used as a primitive tool for adding imperfections to an object's appearance or shape at low memory and computation cost. For example, a rough set of voxels can be turned into a compelling cloudscape by combining it with volumetric noises [Schneider 2023]. In this context, procedural noise can be distinguished from the more general notion of texture, since it is expected to respect a number of specific mathematical and computational properties. Some are enumerated in [Ebert et al. 2003], like for instance a high degree of randomness (i.e. no periodicity and no visual repetitions), perfect parallel computing (i.e. random access at constant complexity), unlimited spatial extent, extreme compactness and no fixed resolution, as opposed to images. For some applications, control of the statistical or spectral characteristics of the generated noise may be required. Unifying all these properties in the case of solid (full 3D) noise remains a challenge. Most recent work was carried out solely in the 2D case, while certain core procedural properties, including easy extension to higher dimensions and resolution independence, have been skipped in favor of either high computational efficiency [Heitz and Neyret 2018] or ease of use [Maesumi et al. 2024]. Many applications require noise to be 3D and furthermore animated, like the turbulences seen in clouds or in fluids [Bridson et al. 2007]. Some models may even require an additional noise dimension (4D for example), related to additional physical parameters. Turbulences seen in gazes may for example exhibit different frequency spans according to temperature.

We introduce a new paradigm for procedural noise synthesis that allows spectral control and can be implemented as a GPU shader program with little input data, while being easy to extend to 3D, and higher, where it preserves its efficiency as well as the above-mentioned procedural properties. Fundamentally, we replace the widely used sparse convolution operation by the use of superpositions of randomly oriented hyperplanar waves. This leads to a mathematical formulation where not only most of the current procedural noises can be reproduced, including Lewis, Gabor and Phasor, but also where a variety of other noise-like functions, including new categories of anisotropic patterns (next-to-last image on the right of top of Figure 1) and cellular noises, structurally distinct from Worley's approach [Worley 1996] (last image on the right of top of Figure 1 and last row).

Our algorithm generates noise using non-periodic, complex-valued hyperplanar waves oriented randomly in n-dimensional space. These waves are derived from a spectral-domain noise definition, where noise is computed in the spatial domain using an inverse Fourier Transform (FT). We efficiently compute the continuous inverse FT integral, by approximating it using a combined Monte Carlo (discrete sum) and pre-computation strategy. Randomness is introduced through random phases and wave orientations. Efficiency is achieved by precomputing the waves and storing them in compact 1D tables, enabling fast, parallel, on-the-fly computation with minimal memory use, even in higher dimensions. Our approach further makes it possible to obtain noises with an arbitrary power spectrum, by using different wave profiles along the different directions. Complex power spectral densities increase the number of waves to be stored and pre-computed, but the memory footprint remains a small fraction of that needed to store, for example, full 3D tiles. By modifying the spatial profile of the waves, it becomes possible to generate new types of non-Gaussian random fields, which are beyond what can be produced with Gabor and Phasor noises. By stacking waves with a Heaviside step-function profile, our approach can be used, moreover, to partition space into random polytopes. This permits the creation of new cellular noise functions distinct from Worley's cellular approach [Worley 1996]. We show that a recursive definition also allows us to imitate STIT-like (STable under Iterated Tessellation) [Cowan 2010] cellular patterns, while preserving all procedural characteristics.

Our noise can seamlessly integrate as a new node in modern procedural material modeling systems based on acyclic directed graphs, such as *Substance* and *Mari*. It enables the creation of infinite, non-repetitive, and resolution-free volumetric materials using 1D RGBA transfer functions, as shown in second row of Figure 1. It can also be used to generate unbounded surface materials, which can be mapped without the need for UV coordinates, using style transfer functions inspired by [Bruckner and Gröller 2007] (see examples Figure 1 bottom row). Materials may also be animated. We show examples, like rock freezing, cooling and progressive propagation of moss.

2 Related work

We review noise generation algorithms, focusing on procedural approaches, which we classify into 4 categories based on the computational model they are based on. We then discuss procedural texture basis functions (TBFs), which resemble, but are not exactly, noise in the strict sense of Gaussian random processes. Finally, we also briefly discuss texture-based non-procedural noises.

Lattice noise, based on interpolating values on lattices, is a foundational method for procedural textures [Ebert et al. 2003]. Perlin noise [Perlin 1985] uses hash-coded random vectors at integer lattice vertices, later improved with simplicial complexes [Perlin 2001] for better scaling in higher dimensions. [Taylor et al. 2021] reduces axial artifacts by generating random unit vectors from prime numbers. However, lattice noises lack spectral control and may suffer from visual artifacts depending on the interpolation method.

Sparse convolution noise [Lewis 1984, 1989] was introduced to get rid of interpolation artifacts of lattice noise. It uses random Poisson point processes to sum Gaussian kernels. An integer lattice was used to approximate the Poisson point process, and accelerate the search for closest points, essential in convolution computation. Anisotropic kernels were proposed to generate 1D "shot noise" and 2D "spot noise" [Van Wijk 1991]. Gabor noise [Galerne et al. 2012; Lagae et al. 2009] relied on phase-augmented kernels for spectral control, whereas a sinc kernel was used in [Gilet et al. 2012] to define almost ideal box filters in the spectral domain.

Texton noise [Galerne et al. 2017], stores the kernel in the form of a texture map, enabling control of the power spectral density (PSD) of the noise. Lastly, local spot-noise [Cavalier et al. 2019] defines

kernels as sums of anisotropic Gaussian functions. Despite these improvements, sparse convolution noises remain computationally intensive because of the search of closest points. This becomes a strong bottleneck when moving up to higher spatial dimensions.

Fourier series have a long history in Computer Graphics. In the early eighties, Gardner [Gardner 1985] introduced sums of cosines with phases modulated by other cosines to define turbulent clouds. Sakas [Sakas 1993] used spectral synthesis to model turbulent gaseous phenomena. *LRP noise* [Gilet et al. 2014] relied on "local" Fourier series, defined by cosines with random phases. Like these models, our work draws inspiration from the Fourier transform.

Tiling approaches for procedural noise synthesis have recently gained popularity. They arrange noise tiles similar to patch-based texture synthesis [Barnes and Zhang 2017]. Texton-noise [Galerne et al. 2017], akin to bombing textures [Fernando et al. 2004; Schachter and Ahuja 1979], sums randomly dropped tiles rather than stacking them. Wavelet noise [Cook and DeRose 2005] uses pre-computed, band-pass filtered white noise tiles, suggesting multiple, randomly selected tiles to avoid periodicity, but without a practical solution. Anisotropic noise [Goldberg et al. 2008] adds directional filtering but still faces periodicity issues. [Heitz and Neyret 2018] addresses this by blending three tiles randomly drawn from one noise image, with blending operations preserving histograms. Randomized tiling and blending (RTB) eliminates periodicity, supports non-Gaussian random fields (e.g., texture patterns) since it preserves some structural components like edges [Fournier and Sauvage 2024]. [Lutz et al. 2023] improves noise auto-covariance preservation through importance sampling. Cyclostationary noise [Lutz et al. 2021] introduces periodic statistics. RTB achieves high performance with few texture accesses to produce complex noises, but struggles with higher dimensions like 3D+t or 4D, which remain unachieved.

Noise–like functions share many procedural properties with previously discussed Gaussian noise. Worley [Worley 1996] introduces the first alternative to noise, called cellular texture basis function (TBF) but often referred to as "cellular noise" or "Voronoise". [Bridson et al. 2007] introduces curl noise, designed for turbulent fluid simulation, and [Gaillard et al. 2019] proposes a procedural function generating dendritic patterns for terrain modeling. Phasor noise [Grenier et al. 2022; Tricard et al. 2019] extends Gabor noise to generate a random phase field instead of an intensity field. A periodic function - such as a step function - is then applied to convert phases into intensities, leading to stripe patterns with perfect uniform local contrast. [Guehl et al. 2020] proposes a general formulation for point process texture basis functions (PPTBF), using collections of arbitrary point processes. Once thresholded, this model was shown to cover a large variety of natural random binary structures. Although deep learning and differential variants [Baldi et al. 2023] made it possible to find parameters on a simplified model, control remains one of the main issues, together with high computational cost.

Non-procedural approaches originate from texture synthesis, a broad research area beyond our focus. State-of-the-art techniques optimize images to match statistics or local similarity across scales or sample a latent texture space learned via deep neural networks [Akl et al. 2018]. Some algorithms extend texture synthesis to 3D, either from 3D data or 2D slices [Pietroni et al. 2010]. More recently, 3D texture synthesis from 2D inputs has been achieved using Neural Radiance Fields (NeRF) [Baatz et al. 2022; Huang et al. 2023, 2024]. Because of limited resolution, these methods generate 2.5D patches to be mapped onto 2D surfaces embedded in 3D, rather than large-scale volumetric textures. [Maesumi et al. 2024] trains a probabilistic diffusion denoising model to reproduce multiple 2D noises and interpolate them in a coherent manner. In summary, data-driven texture synthesis methods can mimic noise but are limited to finite-resolution images. Unlike procedural models [Ebert et al. 2003], they cannot evaluate noise at arbitrary positions in infinite space with minimal memory and constant cost.

3 Expected procedural noise properties

Noise generally refers to a function $\mathcal{N}(\mathbf{x})$ returning a random value at any position $\mathbf{x} \in \mathbb{R}^n$. When $n = 3$, $\mathcal{N}$ is often called "solid" noise. We use the term *procedural* as defined in [Ebert et al. 2003]. Accordingly, $\mathcal{N}(\mathbf{x})$ should be a randomly accessible and inherently parallel function of 2D, 3D, or higher-dimensional space. In addition, we are seeking for the following desirable mathematical and algorithmic properties: $\mathcal{N}$ should be **(a)** a function defined from $\mathbb{R}^n$ to complex numbers $\mathbb{C}$ that **(b)** can be computed at constant complexity, **(c)** from a small amount of data and **(d)** without periodicity or repetitions, at any location $\mathbf{x}$ and of **(e)** an *infinite* n-D space, **(f)** independently from computation at any other location $\mathbf{x}' \neq \mathbf{x}$. Note that, inspiring from [Tricard et al. 2019], we generate procedural noises that return complex values: indeed, either the real part, the modulus or the phase of $\mathcal{N}$ can then be used as a real value (e.g. a density of material or a color intensity) to display.

In this paper, we are primarily interested in noises that implement some Gaussian random process. In this case, the noise can be fully characterized by its power spectral density (PSD), which provides intuitive user-control [Lagae et al. 2010]. Thus, spectral control **(g)** can be yet another desired, though not mandatory property. Properties (b), (c), (d) and (f) lead to looking for a noise $\mathcal{N}(\mathbf{x})$ that can be readily implemented in a programmable GPU shader (fragment or compute) such as, for example, Shadertoy [Jeremias and Quilez 2014]. With a few rare exceptions, such as random tiling and blending (RTB), this excludes data-driven texture synthesis techniques, such as those based on neural networks or optimization, as highlighted in Section 2.

In the following, we first introduce our new procedural noise model for Gaussian solid noise. We then explore its extensions to other dimensions, time-varying noise, and non-Gaussian noise.

4 Gaussian Solid Wave-noise

Sparse convolution generates Gaussian noises (e.g. noises that approximate a Gaussian random process) by sampling spatial and frequency domains using kernel functions centered on random points, often requiring many samples to cover a wide frequency range, especially with Gabor kernels, which capture only a single frequency. In contrast, our approach defines noise as the sum of randomly oriented hyperplanar waves, similar to electromagnetic waves. These waves inherently contain a full spectrum of frequencies, and by precomputing them into 1D lookup tables, we achieve fast random

access while maintaining all desired procedural properties. This method can be somewhat seen as a kind of multidimensional ocean surface, where waves from various directions and frequencies combine to create a dynamic, turbulent pattern.

4.1 Procedural noise from Hyperplanar wave functions

White light is composed of a continuous spectrum of uncorrelated electromagnetic waves across all frequencies. This physical concept inspired the term white noise in signal processing, which refers to a type of noise with uniform power across the frequency spectrum. Drawing from this analogy, we define our procedural noise $\mathcal{N} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{C}$ as a continuous superposition of electromagnetic-like waves over all spatial frequencies, resulting in a stochastic function that varies across both space and time (n-D + t). An individual electromagnetic wave can be described as: $A(\xi)e^{i(2\pi\,\xi\cdot\mathbf{x}-ct+\phi)}$, where this formulation naturally generalizes to n-D space. Here, $\mathbf{x} \in \mathbb{R}^n$ denotes the position, $t \in \mathbb{R}$ is time, $\xi \in \mathbb{R}^n$ is the spatial frequency vector, c the wave speed (celerity), $\mathbf{A}$ the amplitude, and ϕ a phase offset. Randomness arises from assigning random phases ϕ, while spectral control is provided by amplitudes A:

$$\begin{aligned}\mathcal{N}(\mathbf{x},t) &= \frac{1}{F}\int_{\mathbb{R}^n} A(\xi)e^{i(2\pi\,\xi\cdot\mathbf{x}+\phi(\xi)-ct)}d\xi \\ &= \frac{1}{F}\int_{\Omega}\int_{0}^{\infty} A(f\omega)e^{i(2\pi f\mathbf{x}\cdot\omega-ct+\phi(f\omega))}\,|\mathcal{J}(f\omega)|\,df d\omega,\end{aligned} \tag{1}$$

F is a normalization factor used to control the noise values' range. Note how the first equation mathematically matches a multidimensional inverse Fourier Transform of a spectral domain defined by $A(\xi)e^{i(\phi(\xi)-ct)}$. The second equation derives our actual wave noise formulation from the first one by change of variable. It expresses $\xi = f\omega$ as a scalar frequency $f \in [0, +\infty[$ scaling a unit direction vector $\omega \in \Omega$, where Ω is the unit n-sphere of $\mathbb{R}^n$ (a circle in 2D, a sphere in 3D). $|\mathcal{J}|$ is the determinant of the Jacobian matrix introduced by change of variable ξ to $f\omega$, which is separable in 2D, 3D and 4D: $|\mathcal{J}(f\omega)| = \mathcal{J}_f(f)\,\mathcal{J}_\omega(\omega)$.

To compute $\mathcal{N}$ efficiently, our main insight is to regard the inner integral as a complex-valued hyperplanar wave, which can be pre-integrated and tabulated, while using importance sampling to evaluate the outer integral, similarly to the sparse convolution approach for point processes. By denoting

$$\mathcal{S}_\omega(x,t) = \int_0^{+\infty} A(f\omega)\,e^{i(2\pi f x\,-\,ct\,+\,\phi(f\omega))}\,\mathcal{J}_f(f)\,df, \tag{2}$$

we get

$$\mathcal{N}(\mathbf{x},t) = \frac{1}{F}\int_{\Omega^+} \mathcal{S}_\omega(\mathbf{x}\cdot\omega,t)\;\mathcal{J}_\omega(\omega)\;d\omega, \tag{3}$$

where the complex-valued hyperplanar wave $\mathcal{S}_\omega(\mathbf{x}\cdot\omega,t)$ propagates in the direction ω. We call the modulus $|\mathcal{S}_\omega(x,t)|$ the **intensity** of the wave, which is constant on hyperplanes orthogonal to ω. We also restrict the integration domain to Ω^+ as directions ω and $-\omega$ perfectly align. Figure 2 shows an example of wave.

Our goal is to provide a computational implementation of $\mathcal{N}$ in 2D/3D and even higher dimensions. For clarity, we derive all the equations and approximations in 3D for $t = 0$, i.e. for a static solid noise. The simpler, 2D case (surface noise) will be discussed in Section 5, where we also discuss how to scale up to 4D, and reintegrate time.

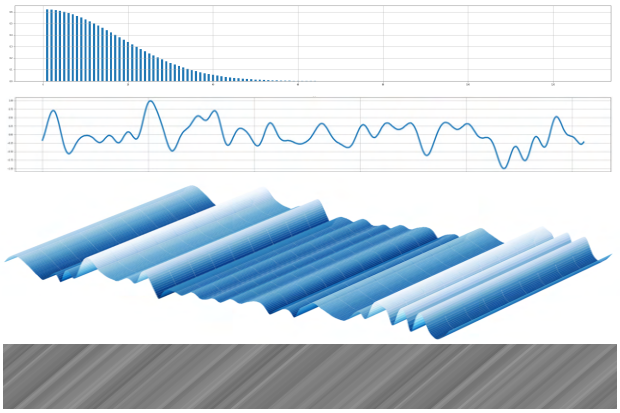

Fig. 2. Given a direction ω, a Gaussian amplitude distribution $A(\cdot,\omega)$ (top) and random phases $\phi(\cdot,\omega)$ generate a wave intensity $|\mathcal{S}_\omega(x,t)|$ (middle) corresponding to a 1D noise. Bottom: modulus of the associated planar wave $\mathcal{S}_\omega(\mathbf{x}\cdot\omega,t)$, respectively displayed as heightfield and as greyscale image.

Spherical coordinates $\omega(\varphi,\theta) = (cos(\varphi)sin(\theta), sin(\varphi)sin(\theta), cos(\theta))$ lead to $\mathcal{J}_\omega = sin(\theta)$ and $\mathcal{J}_f = f^2$. The integral (2) spans an infinite range in frequency space, so we need to limit the frequency band. Therefore, without loss of generality, we set the maximum frequency to $f_{max} = 1$ meaning that $A = 0$ for $f > 1$. We can then rewrite

$$\mathcal{S}_\omega(x,t) = \int_0^1 A(f\omega)\,e^{i(2\pi f x\,-\,ct\,+\,\phi(f\omega))}\;f^2\,df \tag{4}$$

and

$$\mathcal{N}(\mathbf{x},t) = \frac{1}{F}\int_0^{2\pi}\int_0^{\pi} \mathcal{S}_\omega(\mathbf{x}\cdot\omega,t)\;\sin(\theta)\;d\theta\,d\varphi. \tag{5}$$

The Monte Carlo technique is a classical way to numerically estimate integrals by sampling the integration domain. Applying it to both variables f and ω would however require sampling a too large space. We avoid this issue by precomputing the integral on f, and only applying the Monte Carlo approximation on ω, as follows:

$$\mathcal{N}(\mathbf{x},t) \approx \frac{1}{FN_\omega}\sum_{k=1}^{N_\omega}\frac{\tilde{\mathcal{S}}_{\omega_k}(\mathbf{x}\cdot\omega_k,t)\;\mathcal{J}_\omega(\omega_k)}{pdf(\omega_k)}, \tag{6}$$

where N_ω is a user defined number of directions $\omega_k \in \Omega^+$, which are randomly sampled according to a probability density function (pdf) denoted as $pdf(\omega)$. Here, $\tilde{\mathcal{S}}_{\omega_k}$ is a pre-computed approximation of the planar wave.

Let us now see how to define $\tilde{\mathcal{S}}_{\omega_k}$ and $pdf(\omega)$, first in the isotropic case and then in the anisotropic case.

4.2 Isotropic case

An efficient implementation of Equation (6) requires an approximation $\tilde{\mathcal{S}}_{\omega_k}$ of the planar wave, and a sampling strategy $pdf(\omega)$.

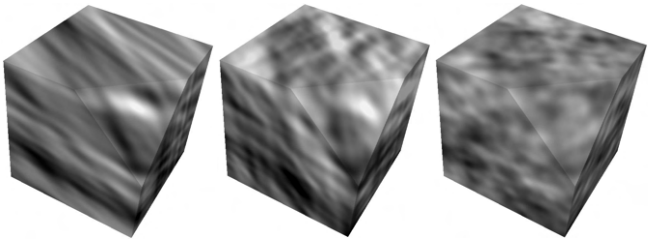

Fig. 3. Procedural solid wave noise is generated by partitioning Ω^+ into N_ω regions of equal weight and superimposing corresponding waves $\tilde{\mathcal{S}}_{\omega_k}$. Here $N_\omega = N_\varphi N_\theta = 12 \times 5 = 60$ regions. From left to right: Real part of an accumulation of 4, 30 and 60 complex valued planar waves.

Pre-computation of the planar wave. The goal is to approximate the integral in (4) where the amplitude A is known, while the phase ϕ is random. We pre-compute a 1D table

$$T[j] = \frac{1}{N_f} \sum_{l=0}^{N_f-1} A\left(\frac{l}{N_f}\omega_k\right) e^{i\left(2\pi \frac{l}{N_f}\frac{j}{N_x} + \phi_{l,k}\right)} \left(\frac{l}{N_f}\right)^2 \quad (7)$$

of size N_x, where $\phi_{l,k}$ are random phases uniformly drawn in $[0, 2\pi]$. It approximates the integral (4) at time $t = 0$ and at discrete positions $x = j/N_x$, by sampling the integrand at discrete harmonic frequencies $f = l/N_f$. In accordance with the Shannon-Nyquist theorem, the spatial sampling should be at least twice the highest frequency: we use $N_x = mN_f$, $m \geq 2$. Due to the use of discrete harmonic frequencies l/N_f and a table size that is a multiple of N_f, the table's content is periodic.

In theory, T should depend on k. However, we don't want to store one table per orientation ω_k, which would be too expensive. First, note that, in the isotropic case, A actually does not depend on ω_k. Second, $\phi_{l,k}$ samples $\phi(\frac{l}{N_f}\omega_k)$ in Equation (4). Using the same wave $\tilde{\mathcal{S}}_{\omega_k}$ for all orientations might create correlation patterns. To avoid this while removing the dependence of T on k, we use the following strategy: A single table T is computed, but we add a random offset $\zeta_k \in [0, 1]$. Ultimately, our approximation is

$$\tilde{\mathcal{S}}_{\omega_k}(\mathbf{x} \cdot \omega_k, t = 0) = T[i_x], \quad i_x = N_x \times frac(\mathbf{x} \cdot \omega_k + \zeta_k) \quad (8)$$

where $frac()$ is the fractional part, used to deal with table index overflow, without introducing visual artifacts thanks to the periodicity of T. i_x is not an integer index. We linearly interpolate between consecutive values in T. This is natively available on GPUs when the table is stored as a 1D texture.

Monte-Carlo sampling. A sampling strategy $pdf(\omega)$ must be chosen for approximating Equation (6). We choose

$$pdf(\omega) = \mathcal{J}_\omega(\omega)/\left|\Omega^+\right|,$$

with $\left|\Omega^+\right| = 2\pi$ the surface of the half-unit sphere. This choice simplifies Equation (6) by making $\mathcal{J}_\omega$ disappear.

In practice, we apply stratified sampling: we partition Ω^+ into $N_\omega = N_\varphi N_\theta$ regions of equal weight (with respect to the pdf), and draw one direction $\omega_k = (\varphi_i, \theta_j)$, where $(i, j) \in [0, N_\varphi[\times[0, N_\theta[$, in each region k according to the pdf. In practice $\varphi_i = 2\pi(i + \iota_i)/N_\varphi, \theta_j = \cos^{-1}((j + \chi_j)/N_\theta)$, where ι_i, χ_j are random variables uniformly drawn in $[0, 1]$. The arccos is important to draw θ_j according to the pdf, which is depending on $sin(\theta)$.

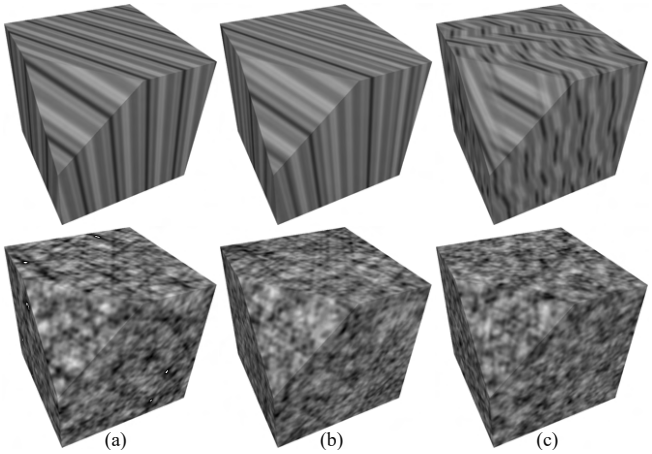

Fig. 4. Slicing allows the use of a coarser sampling ($N_\omega = 40$). (a) The approximation $\tilde{\mathcal{S}}$ of $\mathcal{S}$ defines a periodic wave (top), which may cause salient alignment artifacts in the generated solid wave noise when the table T contains high frequencies (bottom). (b) We disrupt periodicity by introducing random offsets within slices (here of average size $1:4$), but some artifacts remain (bottom). (c) We further use randomly varying wave directions inside slices to help removing the artifacts.

A noise generation example using Monte Carlo sampling is illustrated in Figure 3. Since we used the amplitude distribution of Figure 2 to precompute T, it results in a solid noise (see right) featuring a Gaussian power spectrum similar to Lewis' sparse convolution noise [Lewis 1989].

Slicing. We noticed that when only a few directions N_ω are used along with a high frequency content with respect to the viewing scale, salient alignment artifacts may appear. See Figure 4.a. To avoid this issue, we subdivide $\mathbb{R}^3$ into irregularly spaced slices. The advantage of slicing is twofold: 1) we avoid the alignments caused by the periodicity of $\tilde{\mathcal{S}}_{\omega_k}$ by drawing ζ_k not only according to the direction but also according to the slice, and 2) it allows us to introduce random variations of the direction within each slice, which effectively avoids the correlation patterns and persistent alignments that would be caused by a constant wave intensity across planes perpendicular to its direction. See Figure 4, (b) and (c). Slicing, however, requires to manage smooth transitions from one slice to the next. We do this by blending the wave real and imaginary values at the boundary between two consecutive slices. In practice, slices are generated in $\mathbb{R}^3$ by first defining a straight line from the origin in a specified direction (orthogonal to ω). Points are regularly placed along the line, and further jittered (to add randomness). Orthogonal planes are then threaded through these points to create the slices. A key parameter is the average distance between the points, which determines the thickness of the slices. For example, in Figure 4 the ratio $1:4$ means 4 slices for the cube's edge. This parameter is analyzed in the supplemental material.

Instead of slicing, other subdivision and blending schemes, such as simplicial complexes, could be used, as was done for simplex noise [Perlin 2001]. However, in 3D, this would require 4 table accesses and blends (due to the tetrahedral simplex), a number further increasing linearly with dimension. Slicing, in contrast, keeps this number equal to 2 regardless of dimension, which makes it a much better choice in terms of computational complexity.

Algorithm 1 provides some pseudo-code. The functions $seed()$ and $rnd() \in [0, 1[$ correspond respectively to the initialization and generation of random numbers. $ortho$ computes an orthogonal direction, and $interpol$ returns a weight used for interpolation according to the regularly placed and jittered points $slpos$ that define the slices. We treat the complex-valued lookup table T like a 1D texture, e.g. like a function from $[0, 1]$ to $\mathbb{R}^2$, following the typical GPU texture implementation approach.

ALGORITHM 1: `Isotropic wave noise`

Input: (x, y, z), table T, dir. (N_φ, N_θ), slice thickness W, norm. fact.F
`sum` $:= (0.0, 0.0)$, $x := x \cdot W$, $y := y \cdot W$, $z := z \cdot W$
foreach $(i, j) \in ([0, N_\varphi - 1], [0, N_\theta - 1])$ **do**
  `seed`$(i, j, 0)$, $(\alpha, \beta) := ortho\left(\frac{2\pi(i+\text{rnd}())}{N_\varphi+1}, \arccos(\frac{j+\text{rnd}()}{N_\theta+1})\right)$
  $p := x \sin\beta\cos\alpha + y \sin\beta\sin\alpha + z\cos\beta$ `// projection`
  `seed`$(i, j, \lfloor p \rfloor)$, `slpos` $:= 0.3 + 0.4 \cdot$ `rnd()` `// slice position`
  $u := interpol(slpos, frac(p))$ `//` $u \in [0, 1]$
  foreach `s` $\in \{0, 1\}$ **do**
    `seed`$(i, j, \lfloor p \rfloor + \text{s})$,$(\alpha, \beta) := (\frac{2\pi(i+\text{rnd}())}{N_\varphi+1}, \arccos\left(\frac{j+\text{rnd}()}{N_\theta+1}\right))$
    $p' := (x \sin\beta\cos\alpha + y \sin\beta\sin\alpha + z\cos\beta + \text{rnd}())/\text{W}$
    `sum` $+= [(1 - \text{s}) \cdot (1 - u) + \text{s} \cdot u] \cdot T(frac(p'))$
  end
end
return `sum/F`

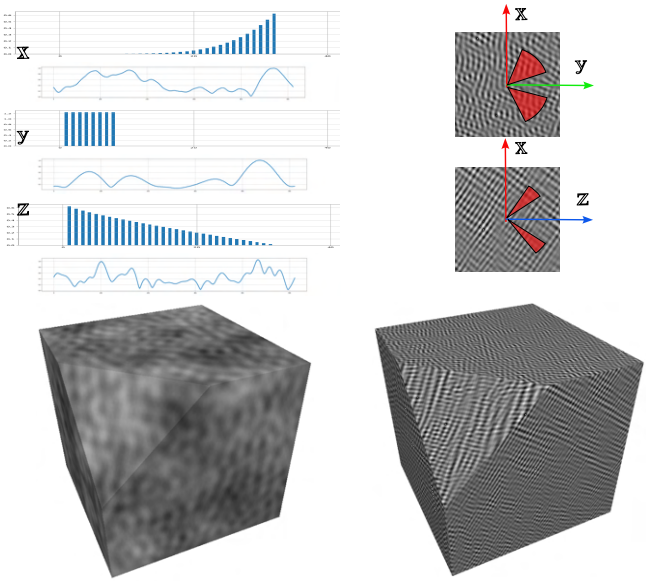

Fig. 5. Anisotropic 3D wave noises. Left: Different amplitude distributions (bar-charts on top) are used along the three coordinate axes and stored in separate tables T. Spatial waves (blue curves show real part) are precomputed using identical, random phases, to allow interpolation. Right: Null amplitudes are used in the frequency domain except within the solid angles in red (top), where the same amplitude distribution, stored in single table T, is defined. Since the solid angles (here, pyramids) are narrower along the z-axis, the resulting, anisotropic noise patterns differ in XY and XZ planes. The resulting solid noises are shown on bottom.

4.3 Anisotropic case

Anisotropy is achieved by making the wave amplitudes vary with respect to their direction.

General case: To handle anisotropy, a specific table T needs to be defined in each direction ω. Our model would then allow to handle arbitrary power spectra. As an infinite number of tables spanning the full space Ω cannot be defined, a practical solution consists in partitioning Ω^+ into a finite set of solid angles, each with frequency content stored in a table T_ω, and then applying directional interpolation. In this context, adapting the partitioning to the desired Power Spectral Density (PSD), i.e., using smaller angles in high-energy regions, is more effective than a straight regular partitioning of Ω. A challenge is that direct interpolation of complex-valued waves is not straightforward. We are looking for linear interpolation of amplitudes, but this cannot be obtained through interpolation of waves, because the sum of two complex numbers does not equal the sum of their amplitudes. When waves are out of phase, they may cancel out partially or completely, which is known in wave theory as *interference*. This fundamentally limits the ability to interpolate wave amplitudes by simply interpolating their complex representations. We distinguish different scenarios to circumvent this issue.

Uni-phase waves with different frequency contents: The use of an identical phase for each frequency allows for interference-free intensity interpolation, as two complex numbers can have their amplitudes directly summed when their phases align. This means that all tables T_ω must be precomputed with the same set of random phases. During noise generation the random offset ζ_k drawn inside a given slice must also be identical when interpolating according to the direction. The goal is to guarantee that phases keep being aligned. A basic example using only three tables associated to each of the coordinate axes is shown in Figure 5 (left). During noise generation, interpolation is performed by computing the dot product: $\tilde{\mathcal{S}}_{\omega_k}(x) = \omega_k \cdot (T_x[i_x], T_y[i_x], T_z[i_x])$, with i_x, same as previously defined in Equation 8.

More complex power spectra can be defined using more input tables, storing amplitudes in specific directions.

Waves of same frequency content in specific directions: This is a simpler anisotropy case. It uses a null amplitude everywhere except in a few directions defined by solid angles, for which the same amplitude, stored in a single table T, is used. See Figure 5 (right) as an example of two distinct, non-zero amplitude angles in 3D.

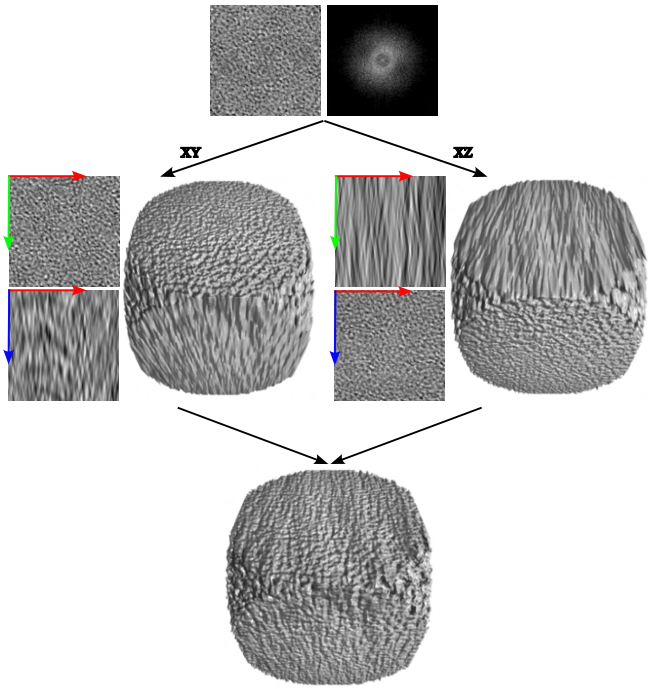

Fig. 6. Anisotropic 3D wave noise generated from a single 2D example (top). Middle: using the 2D PSD of the example (top right) respectively on XY (left) and XZ (right) planes. 3D extension is obtained by extending 2D angular sectors to 3D pyramids, as previously shown in Figure 5 (top). Bottom: combining the two previous sets of angles.

4.4 Designing solid wave noise from 2D noise images

In practice, designing noise directly in the frequency domain can be a difficult task, which is why noise "by example" was introduced in [Galerne et al. 2012]. This approach simplifies the process by allowing the user to provide a sample noise image. Similarly, we extend this idea to generate solid wave noises using a single 2D noise image as input (see Figure 6). We first decompose the 2D input spectral domain computed from the input noise image by Fast Fourier Transform (FFT) into angular sectors, each with its own precomputed table (and identical phases to allow interpolations). Then, we extend the sectors to pyramids in 3D. We have three choices to do so, as the input can be considered as either corresponding to the XY, XZ or YZ planes of a volume. Two of these results are shown in Figure 6 (middle), where the 3D noise naturally matches the example in the corresponding planes. Further merging such sets of solid angles improves spectral coverage (bottom), enabling enhanced, yet not full 3D spectral control.

We leave this aside for further work. Our simpler fusion method, offering intuitive control over the generated solid noise through one or several 2D noise images, proved sufficient to generate all the examples in this paper.

5 Wave noise in other dimensions

In this section, we explain how the procedural wave noise model we just defined for solid noise can be extended to 2D, 4D, as well as to animated noise. We show that specific simplifications can be made in the 2D case and that consistent models for the generation of animated n-D+t noises can be set up by extending a n-D noise rather than using more general, yet costly, (n+1)D wave noise.

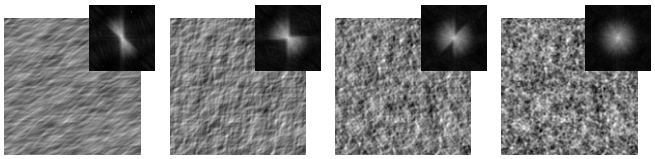

Fig. 7. 2D noise generation using the real part of an accumulation of 5, 10, 15 and 20 complex valued $\tilde{\mathcal{S}}_{\omega_k}$. Upper right is the corresponding power spectrum. Ω^+ was decomposed into 20 equally sized angular sectors.

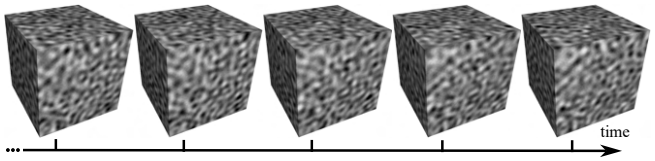

Fig. 8. Time varying solid wave noise ($t = 0$ to 0.4), using a speed $v = 0.01$. The power spectral density (PSD) is perfectly preserved over time.

5.1 2D noise

The wave noise function in Equation (1) can be reused in 2D by substituting the 3D direction vector with a 2D vector $\omega = (cos(\alpha), sin(\alpha))$. In this case, $\mathcal{J}_\omega = 1$ and $\mathcal{J}_f = f$. Since $\mathcal{J}_\omega$ is now constant, the sampling strategy $pdf(\omega)$ defined in Equation (6), consists in partitioning Ω^+ into N_ω equally sized angular sectors while drawing one random direction *uniformly* in each. In 2D, Ω^+ represents a half-unit disk, with $\left|\Omega^+\right| = \pi/2$ its surface. The previous slicing method can be reused, with 1D lines instead of planar slices. A surface noise example using 20 angular sectors is depicted in Figure 7.

5.2 4D and higher dimensional noises

Higher-dimensional noises can be useful in applications involving phenomena with multiple stochastic parameters, e.g. if non-uniform values for temperature or pressure needed to be initialized over a 3D density field. Let us detail our solution in the 4D case:

$$\omega = (sin(\psi)cos(\varphi)sin(\theta), sin(\psi)sin(\varphi)sin(\theta), sin(\psi)cos(\theta), cos(\psi))$$

leading to $\mathcal{J}_\omega = sin^2(\psi)sin(\theta)$ and $\mathcal{J}_f = f^3$. In this case, Ω^+ represents a half-unit 4D-sphere, also called 3-sphere in mathematics. We still can set $pdf(\omega) = \mathcal{J}_\omega(\omega)/\left|\Omega^+\right|$, with $\left|\Omega^+\right| = \pi^2$. Technically, there is no issue in scaling up from 3D to 4D, except that N_ω must be increased to ensure adequate directional coverage. However, in 4D, importance sampling involves drawing ψ based on $sin^2(\psi)$, whose integral lacks a closed-form inverse. A numerical approximation is required. Cost is further increased because of wave interpolations, doubling table accesses from 4 in 3D to 8 in 4D.

5.3 Time-varying wave noise

3D+t noise could be generated as 4D noise, but at high computational costs. To reduce complexity and provide easier control, a

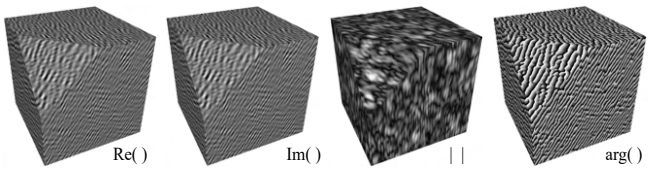

Fig. 9. Solid wave noise is complex valued. From left to right: real and imaginary parts of an anisotropic wave noise, modulus and phase, these two respectively showing typical ridged noise and Phasor patterns.

key-framing approach could also be envisioned, i.e. temporal interpolation between a set of predefined solid noises. This would, however, cause undesirable "ghosting" effects, leading to poor visual results. Instead, we draw inspiration from electromagnetic or ocean waves, allowing waves to propagate through space at their own speed.

Time was introduced in our noise model in Equations (2) and (6) using a pulsation c, which is a classic parameter in electro-magnetic wave theory, but all the approximations discussed in Section 4 were developed for $t = 0$. It is easy to see that our approximation of $\tilde{\mathcal{S}}$, based on a precomputed table T according to Equation (7), no longer holds when $t > 0$. In fact, each time step results in a new $\tilde{\mathcal{S}}$ and would require its own table to be precomputed. To avoid the need for generating such time-varying set of tables we input a wave speed parameter v and we substitute $\tilde{\mathcal{S}}_{\omega_k}(\mathbf{x}\cdot\omega_k, t)$ for $\tilde{\mathcal{S}}_{\omega_k}(\mathbf{x}\cdot\omega_k - vt, 0)$. The effect of time is now to introduce in each direction, a synchronized shift of the entire $\tilde{\mathcal{S}}$ along the direction ω_k, rather than applying oscillations to all its individual frequency components as given by Equation (4). This enables the precomputation of the table T, in the same manner as defined in Equation (7), while motion is achieved at no additional computational cost by simply shifting the table access indices: Equation (8) becomes

$$\tilde{\mathcal{S}}_{\omega_k}(\mathbf{x}\cdot\omega_k, t) = T[N_x \times frac(\mathbf{x}\cdot\omega_k - vt) + \zeta_k]. \tag{9}$$

Since each of the N_ω waves propagates in a different direction, we experienced that such directionally-synchronized motion produces a rich-enough movement, without visual artifact, provided that we use directions in the entire Ω, and not only Ω^+. Moreover, user control is intuitive thanks to explicit speed control in each direction.

As results show (see the companion video), this approach maintains visual and spectral consistency of the noise over time, without any additional computational or memory cost. Figure 8 illustrates time-dependent solid noise, showing its evolution for 5 consecutive time steps. Note that generating 4D+t noise is also straightforward with our approach, using the same technique but for extending 4D noise.

6 Extension to Non-Gaussian wave noises

Our noise is defined by summing complex-valued waves, oriented in random directions. So far, we used the real part of the waves, with random phases to mimic Gaussian processes and control via amplitude distributions, similarly to the Gabor kernels used to control the PSD of sparse convolution noise. But our model is more general, and can also capture a variety of non-Gaussian noises.

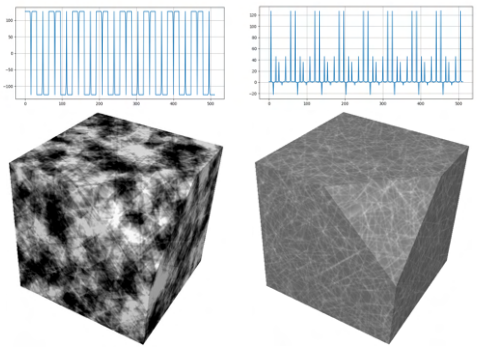

Fig. 10. Non-gaussian 3D wave noise. Top shows wave intensities, bottom resulting noises. Local intensity peaks can generate either crystal-like structures (left) or scratches and webs (right).

6.1 Phasor and Ridged noises

Instead of the real part, we may use the modulus or the phase of a wave noise to generate some density or color value. Since the phase belongs to a random phase field, using it is equivalent to using Phasor noise [Tricard et al. 2019].

Figure 9 shows an anisotropic wave noise example, serving as basis for our wave-based Phasor noise. While the latter is obtained by using the Phase (bottom right), the modulus of wave noise (bottom left) resembles ridged noise, a non-Gaussian noise often used for modeling fractal terrains.

6.2 Crystal-like and Wired noises

To generate other real-valued noises with non-Gaussian statistics, one can also define custom waveforms $\tilde{\mathcal{S}}_k$, that are not constrained to Eq. (4). Two examples of arbitrary custom waves are shown in Figure 10. On the left, $\tilde{\mathcal{S}}_k$ is characterized by several steps, providing a crystal-like noise appearance. On the right, $\tilde{\mathcal{S}}_k$ features localized Dirac impulses, creating a web-like or wired noise pattern. Both examples were produced without modifying Algorithm 1; only the values of T were precomputed using a custom approach.

It is important to note that such patterns cannot be produced with Phasor noise, even if the notion of wave seems similar at first glance to the one defined in [Tricard et al. 2019]. In our case, we sum multiple waves, whereas Phasor composes a periodic function with its random phase field, thus generating uniformly contrasted, mainly stripe like, patterns.

6.3 Cellular noise

Yet another way of creating non-Gaussian noises from our wave-based model is to substitute the sum of waves with another operator. Worley pioneered the use of n-th closest distances to random points, instead of summing kernel functions centered on them. Similarly, we can substitute the sum by a “min” operator. In practice, this modifies Algorithm 1 by initializing sum to $+\infty$ instead of 0 and replacing additions by: $sum \coloneqq \min(sum, T(\text{frac}(d)))$. Substituting table T by a reversed triangle function, $T(x) = |x - \text{slpos}|$, centered on slices yields cellular patterns with aligned borders, differing from Worley noise. See Fig. 11(a). The resulting pattern closely mimics a

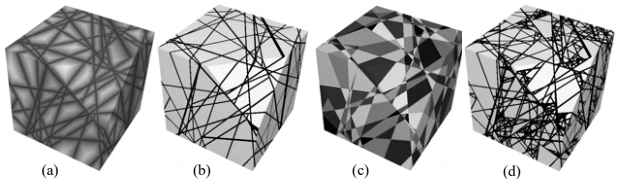

Fig. 11. Cellular wave noise. (a) using a min operator generates cellular noise, as inspired by Worley's cellular function, our cells having a different structure. (b) visualizing a binary hash code obtained by stacking boolean values resulting from a step function wave. (c) using a $1-\delta$ wave intensity. (d) further using an iterative cell subdivision, imitating STIT patterns.

random hyperplane process (RHP), which is a popular mathematical basis for generating random polytopes [Hug and Schneider 2024]. Figure 11(b) is same as (a), but using a Dirac delta function: $T(x) = 1-\delta(x-slpos)$. In both cases, blending is disabled across slice borders, causing visible discontinuities. To conceal them, a slice-aligned wave is added, effectively drawing both, slices and waves. Algorithm 2 reflects this: (1) we loop over both, slices ($k=0$) and waves ($k=1$) and (2) blending is replaced by two min operations, e.g. one on either side of the corresponding hyperplane.

ALGORITHM 2: `Isotropic cellular wave noise`

```
sum := +∞
foreach (i, j) ∈ ([0, N_φ − 1], [0, N_θ − 1]) do
    for k := 0 to 1 do
        seed(i, j, k · (id + 1 − χ))
        (α, β) := ( 2π(i+rnd()) / (N_φ+1) , arccos( (j+rnd()) / (N_θ+1) ) )
        p := x sin β cos α + y sin β sin α + z cos β   // projection
        id := ⌊p⌋, o := frac(p)
        seed(i, j, id), slpos := 0.3 + 0.4 · rnd()
        sum := min(sum, |slpos − o|)                  // no T needed
        χ := step(slpos − o)        // 1 if o < slpos, else 0
        seed(i, j, id − χ + (1 − χ)), slp := 0.3 + 0.4 · rnd()
        sum := min (sum, χ(1 + o − slp) + (1 − χ)(1 − o + slp))
    end
end
return sum/F
```

Figure 11(c) still shows the same cellular noise, visualized differently. Now, instead of a Dirac, we use a step function $T(x) = step(x-slpos)$ (0 if $x < slpos$ else 1), partitioning space into half-spaces. Each point thus receives a binary value, and stacking these for all slices and waves forms a binary code identifying a unique polytope. In (c), this code is mapped to grayscale via hashing to reveal the random polytope structure.

By introducing an iterative process that adds more polytopes inside a given polytope according to some probability, we can furthermore imitate STIT tessellations (where STIT means STable with respect to ITerations), another well-studied mathematical model for random tessellations, often applied to the modeling of crack patterns [Nagel et al. 2008]. Figure 11(d) illustrates a result obtained with a probability of 70% and stopped after five iterations. Note how only some "initial" polytopes (shown in (b)) are further subdivided. We would like to stress that our cellular noise is only a procedural approximation of true RHP and STIT. Because of slicing, our approach enforces some parallelism that does not exist in mathematical random processes.

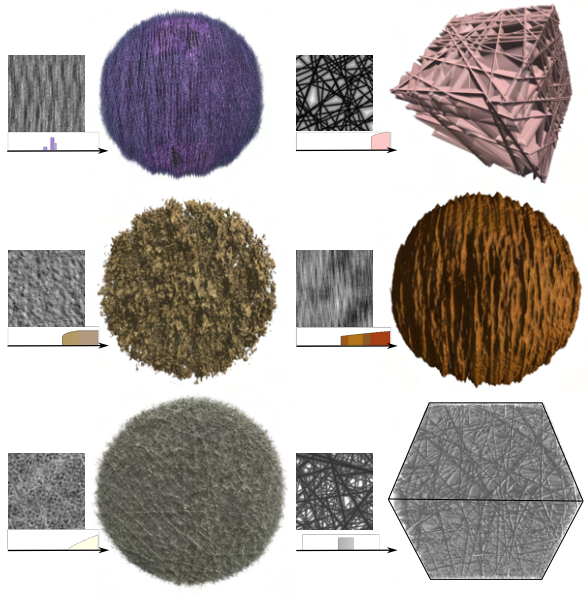

Fig. 12. Modeling volumetric data or micro-materials using a classical RGBA Transfer Function (TF). Left: 2D slice of the noise and TF. Right: corresponding volumetric material.

7 Results and Evaluation

Many results were already shown in Sections 4, 5 and 6. We discuss below the applications of wave noise, and provide an in-depth comparison with previous procedural noise models in terms of performances, expressivity and quality. See also the companion video.

7.1 Applications

Generating volumetric data: Firstly, our model can directly be used to generate volumetric data-sets, either structured or representing unstructured, micro-material. This is achieved by applying transfer functions for color and transparency to our model (see Figure 12).

A result based on by-example design (see Section 4.4) is shown in Figure 13. Note the achieved visual similarity between the generated volumetric wave noise data (shown both as a slice and in 3D) and the input images.

Generation of PBR materials: A second major application is the generation of Physically-Based Rendering (PBR) materials to be used on surfaces. Being procedural, wave noise can be seamlessly integrated within any existing procedural texture generation tool (i.e. a node graph system used to design resolution-free materials), where it would provide a versatile new node. Solid wave noise can also directly be used as follows.

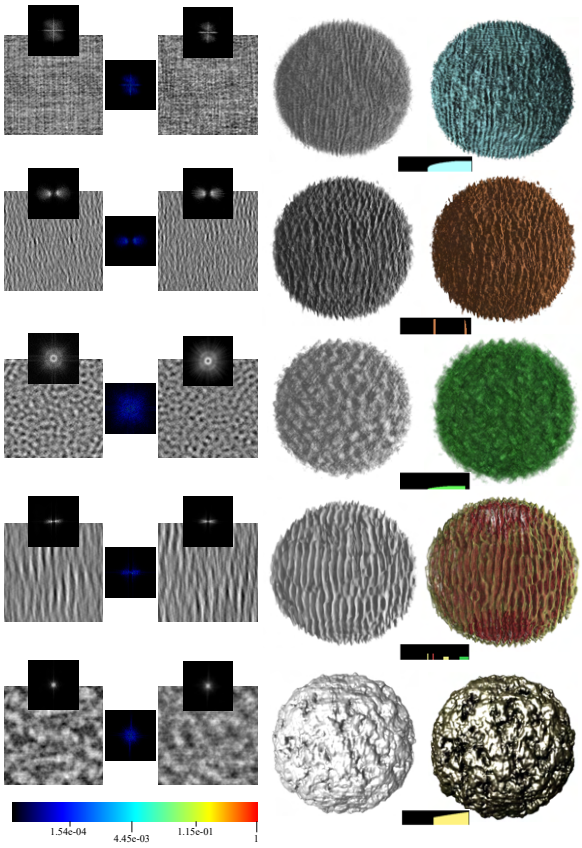

Fig. 13. Solid wave noise by example. Left column: (left most) 2D input exemplar with corresponding power spectral density (PSD), (middle) measured PSD error using L2 norm, (right) 2D slice of the generated wave noise with corresponding average PSD. Note the very low spectral error (see logarithmic scale on bottom). Right column: rendering of a corresponding 3D material with transparency (left) and RGBA transfer function (right).

PBR materials on arbitrary surfaces can be generated using an adaptation of the method of [Bruckner and Gröller 2007], introduced in the field of direct volume rendering: We first generate various material styles by transferring attributes such as colors, normals, albedo, reflectance, and ambient occlusion from a material database. The resulting material is stored as a "style transfer function" represented as a cube map (capturing position) projected onto a sphere (capturing normals), unlike [Bruckner and Gröller 2007], which uses only a lit sphere projection. The material can then be reapplied at any 3D point of a surface, based on the wave noise value, without the need for UV coordinates. See Figure 14.

In addition, noise can serve as a guide for optimization-based data-driven texture synthesis methods, an approach known as “textures by numbers”, and extended to PBR material generation in [Guehl et al. 2020]. In Figure 15, wave noise is used to generate a guidance map (top) for the synthesis of structured materials, making use of the exemplars of texture layers at the left.

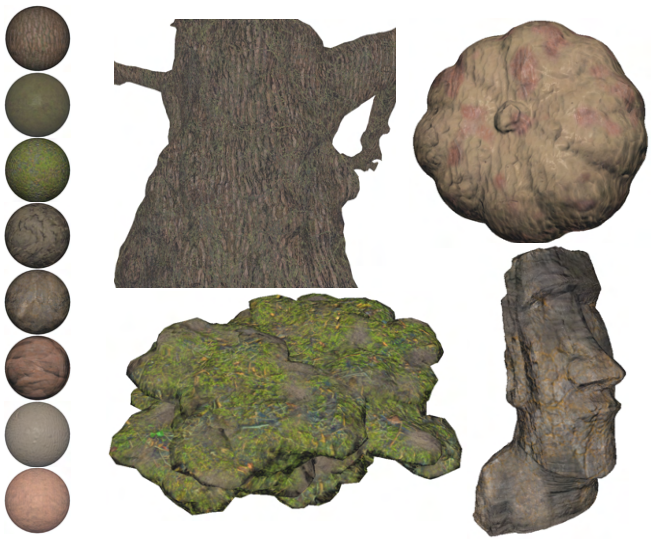

Fig. 14. Modeling surface materials using 3D wave noise and style transfer functions (left), which we mapped on surfaces without UV coordinates.

Generation of animated materials: The last application is animation. Figure 16 illustrates this through four inspiring examples: the transformation of a surface into an icy rock, the cooling of a glowing stone, the gradual growth of moss on a rock, and the formation of crack patterns on a clay statue. The first three rows showcase 3D+t noise, enabling patterns to move and deform rather than simply fading in or out, as static 3D noise interpolation would. This dynamic behavior adds a sense of realism (refer to the video for a detailed view). In the last row, the probability of subdivision of 3D cellular wave noise increases over time, gradually revealing more fractures.

Smooth transitions: Our noise relies on real-valued parameters that support linear interpolation. Precomputed waves can also be interpolated (assuming phase alignment) to smoothly change frequency content. As shown in Fig. 17, this enables the noise to smoothly evolve across the volume, from one cube corner to the opposite (not just across faces).

7.2 Implementation and performances

Our wave noise generator was implemented in C++.

Performances were computed on a desktop computer, Windows 11, Intel, i7, 8 cores, 32 Go memory, NVidia GeForce RTX 3090, 24Go VRAM GDDR6X, 10496 cores, and OpenGL *shader language*. We analyzed the average framerates for a viewport of size 1024×1024. Framerates in the companion video might differ because we used an interface with editable parameters for the video, whereas measurements have been done using fixed parameters for the figures.

Table 1 compares performance for generating 2D/3D textures of various sizes using compute shaders. It includes Perlin and Worley fractal noises, Gabor noise with varying kernel counts, and wave noise with varying direction counts and dimensions.

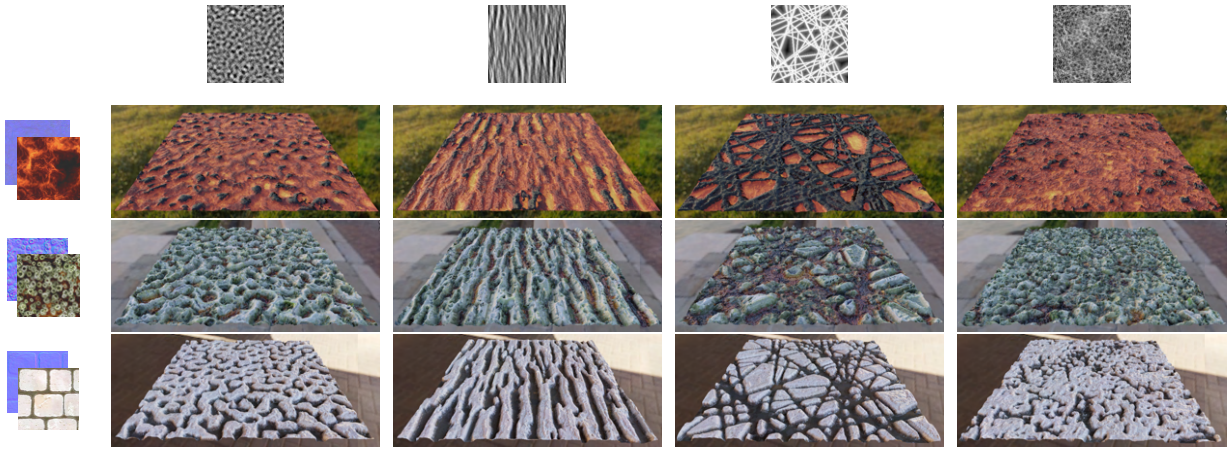

Fig. 15. By-example PBR material modeling using wave noise (top row) as “guidance”, and input exemplars of texture layers, here normal and albedo maps (left column). The resulting surface materials are rendered using Blender Cycles, with environment-map lighting for photo-realistic visualization.

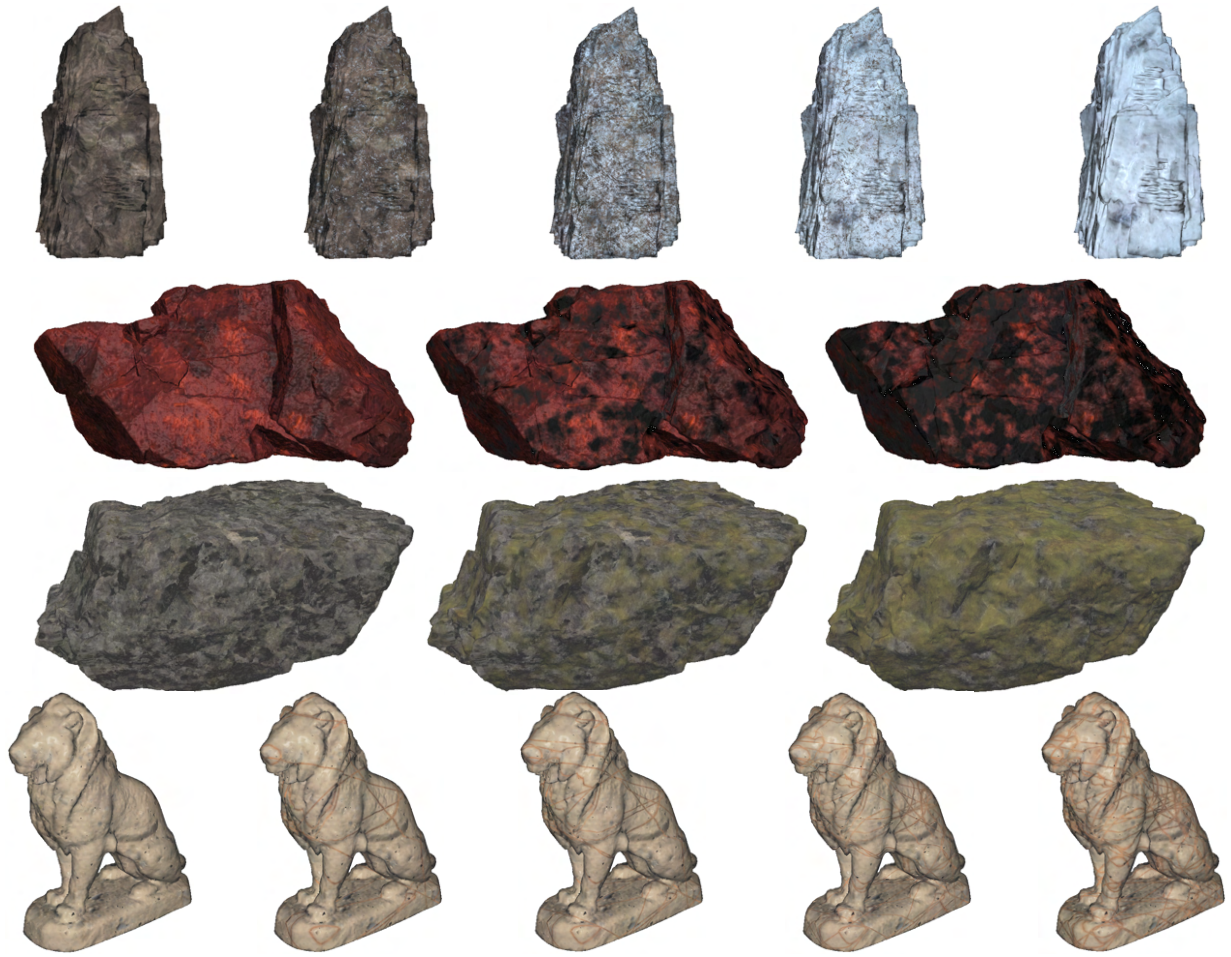

Fig. 16. Examples of 3D+t wave noise for material animation. Each row respectively shows a freezing rock, the cooling of a glowing stone, moss growth on a rock and cracks forming on a clay statue.

3D Textures	Perlin Noise (fractal)	Worley Noise (fractal)	Gabor Noise			Wave Noise 3D/3D+t/4D		
			10 Kernels	50 Kernels	90 Kernels	10 Dir	50 Dir	100 Dir
256^3	5.6	15.15	40.86	175.86	315.2	2.6/2.74/7.45	11/11.7/25.8	22/23.3/44.6
512^3	44	131	311.3	1 438	2 584	20.8/22/53.1	88.1/93.2/206.6	175/185.6/361.3
1024^3	346	1 047	2 552	116 000	206 000	167/176.4/429	706/749.6/1 719	1 410/1 502/2 970
2D Textures								
1024^2	0.328	0.94	2.23	11.0	22.12	0.14	0.62	1.28
2048^2	0.92	2.75	6.37	32.9	61.9	0.39	1.82	3.72

Table 1. Performance (in ms) for the generation of textures of varying resolution using compute shaders. Each result is an averaged timing over 100 texture generations. The last column indicates timing for the generation of a 3D texture with our method using a 3D, 3D+t or 4D noise. The last lines show timings for rendering a screen-covering quad using a fragment shader of 2D noises.

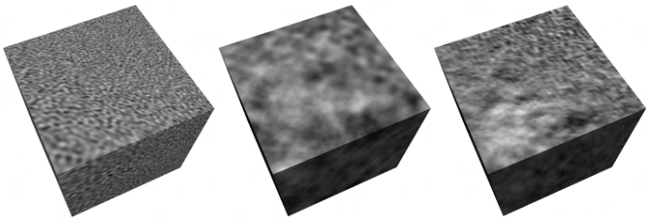

Fig. 17. Smooth transitions: (left) varying frequency range; (middle) varying frequency contents, i.e. the amplitude variations within the range; (right) varying frequency contents and anisotropy.

7.3 Model parameters

Our noise model is based on a compact set of parameters made accessible via a graphical user interfaces (GUI). See Figure 18. More details are available in the supplemental material. For clarity, we distinguish the following groups of parameters:

Spatial Transformation controls 3D noise position and scale through translation (`X`, `Y`, `Z`) and global zoom factor.

Quality versus Framerates allows to balance both, by setting slice size (parameter W in Algorithm 1), and the number of directions, which we reduced to a single value $NDir = N_\varphi \cdot N_\theta$, by fixing $N_\theta = 4$, as this proved to be an appropriate choice in most cases.

Frequency Range shapes the spectral domain via anisotropy spread, and a global band pass filter defined by frequency bounds (`FreqMin`, `FreqMax` $\in [1/N_f, 1]$,with $N_f = 64$). Outside this range amplitudes are null. This affects only Gaussian noises.

Waveforms and Noise types allow users to choose from various preset waveforms, wave combinations `Operator`, and degrees of anisotropy. It includes options for: 1) Gaussian noises characterized by amplitude variations within the previous range [`FreqMin`, `FreqMax`], commonly referred to by color names such as blue (increasing) and brown (decreasing) or other variations, 2) non-Gaussian noises, classified according to visual characteristics (e.g., the crystal- and scratch-like patterns in Figure 10), and cellular noises as in Figure 11. While we provide a limited selection of waveform presets, a potential extension would be the integration of an interactive 1D function editor allowing users to design custom waveforms more freely.

Cellular Noise Settings allow to set the probability and maximum recursion depth of cellular noise, as in Figure 11(d).

Post-Processing and Display define what value is rendered (as in Figure 9), scaled by `contrast` used to compute normalization factor F in Algorithm 1.

Temporal and Animation Controls add motion to the noise using a timeline position and wave speed as in Figure 16.

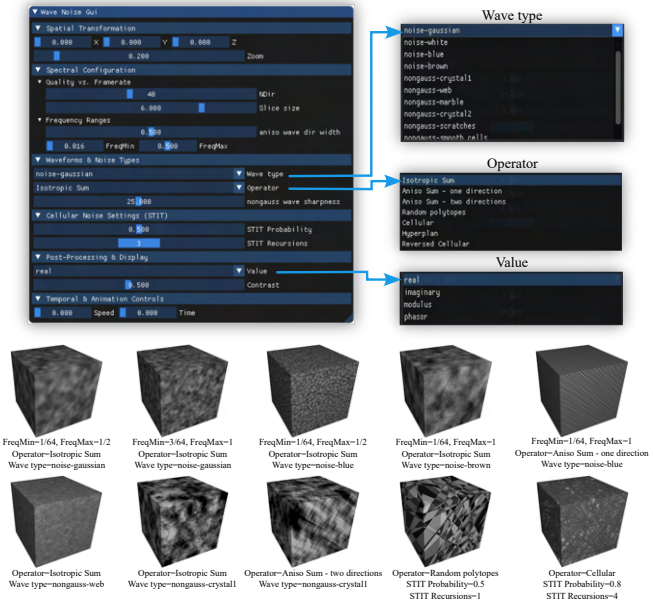

Fig. 18. Top: Graphical User Interface (GUI). Bottom: Some different parameter settings with corresponding noise result.

7.4 Comparison with previous models

Our goal was to develop a generic and efficient procedural noise, usable in high-dimensions (3D, 4D, and 3D+t). We compare with the noises with similar procedural properties, namely lattice noises [Perlin 1985, 2001], sparse convolution noises [Lagae et al. 2009; Lewis 1989], Phasor noise [Tricard et al. 2019] and Worley's cellular noise [Worley 1996].

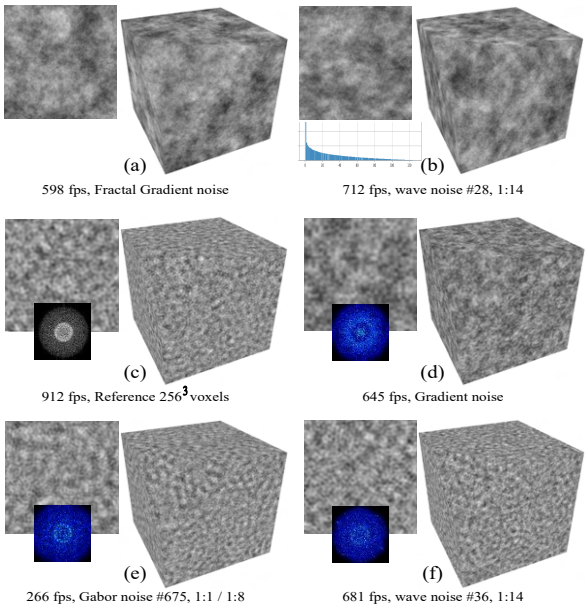

Fig. 19. Comparison with Gaussian noises, with 2D cut of XY plane (left) and 3D views (right): (a) Perlin noise: fractal sum of 8 octaves, each scaled by 0.8 and weight halved at each level. (b) Wave noise with $1/f^n$ decreasing amplitudes (see bar chart) resulting in a fractal noise. (c) Reference noise 256^3 computed using inverse FFT and rendered as 3D texture, (d) Perlin noise struggling to replicate the reference, (e) Gabor noise better matching the reference thanks to spectral control, but requiring 2 lattices (1 : 1 and 1 : 8 sizes), resp. 10 and 15 points, i.e. 675 kernels. (f) Wave noise achieving a good match with 36 directions and 14 slices per direction, at lower costs.

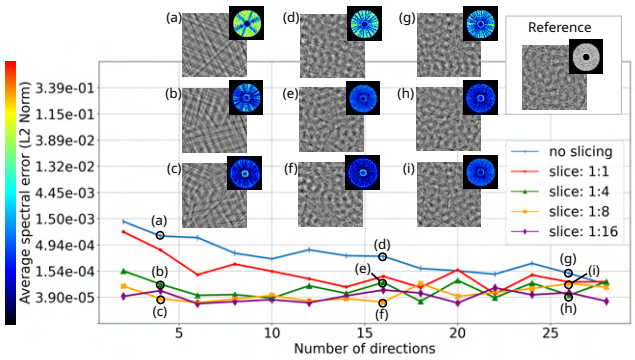

Fig. 20. Quantitative evaluation of wave noise: Top-right shows the reference and its PSD. The chart plots average spectral error (L2 norm) vs. number of directions for varying slice thicknesses. Slicing reduces error with few directions (see a, b, c), but causes spectral leakage as directions increase: the error no longer decreases. It may even increase (see f and i).

Comparison with Gaussian noises: Figure 19 presents side by side comparisons between our wave noise and solid fractal Perlin noise, a widely used noise for modeling natural phenomena, such as volumetric clouds, and whose framerate scales with the number of octaves. For the comparison, we used our isotropic wave noise, where amplitude decreases as a function of frequency f in the form $1/f^n$ to imitate the fractal behavior. Our method relies on precomputed waves, so performance only depends on the number of random directions, not on the number of scales. This leads to fairly similar visual output for comparable framerates. But, additionally, our wave noise supports animation without further impacting performance, a significant advantage over such a similar-looking lattice noise. We finally provide a ground truth noise texture in (c) generated by inverse 3D FFT, representing isotropic band-pass filtered white noise across two frequency bands, and show that Perlin noise fails to replicate it. Indeed, summing scales in Perlin noise creates "Brownian fractal" spectra, unsuitable for band-pass filtering. We compare to spectral density of the reference shown in (c) using the same logarithmic scale and color ramp as in Figure 6. Note that the error in (d) is highest in the low frequency range (spot in center).

Next we compare with Gabor sparse convolution noise [Lagae et al. 2009], able to better match the reference thanks to spectral control (see error, mostly dark blue): its FT is a sum of Gaussians enabling arbitrary power spectra to be generated. However, precise control is difficult for spectra like in (c) without many kernels. Two lattices (1 : 1 and 1 : 8 sizes) with resp. 10 and 15 points require a total sum of 675 kernels. Using more kernels and finer grids could improve accuracy but also increase costs. Gabor noise speeds up searches by jittering points within lattices, but the number of neighbors (and points) scales exponentially with dimension (9 in 2D, 27 in 3D, 81 in 4D). Covering frequency ranges with high precision requires many points on a single lattice. [Galerne et al. 2012] addresses this by splitting the spectrum into bands with tailored lattices. In the end, a Monte Carlo approach over spatial and frequency domains is used, which explains high costs. Our method reduces sampling via pre-computations, reproducing the ground truth more efficiently (see Figure 19.(f). Figure 20 further shows a quantitative evaluation, outlining how the spectral error evolves according to the number of directions and the slice thickness.

Comparison with non-Gaussian noises: Figure 21 provides a side-by-side comparison with previous non-Gaussian noises. Comparison with Phasor noise shows the efficiency of our model, twice as fast, for similar visual results. We then provide comparisons with cellular wave noise, even while our model generates, by design, a structurally different pattern (see Section 6). Like sparse convolution, Worley noise uses random points jittered inside an integer lattice, so its performance is directly linked to their number. For cellular wave noise, we implemented the triangular function profile directly as program code, thus requiring no table T and no memory at all. We can observe that our wave noise provides a new class of cellular structures, resembling hyperplanar random processes (HRP).

7.5 Limitations

While our wave noise model is efficient enough for high dimensions and allows to capture a large range of existing noises, plus some new cellular noises, its current formulation brings several limitations.

A first difficulty is related to high frequency content, which may require increased slicing to mitigate alignment artifacts, thus reducing spectral accuracy. Another issue is real-time filtering, that is evaluating the noise at different resolutions, to avoid artifacts at distance rendering. For linearly combined waves, it is possible to

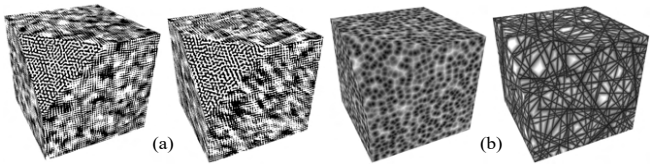

Fig. 21. Comparison with non-Gaussian noises: (a) Phasor noise [Tricard et al. 2019] (left) vs. wave noise (right) for similar frequency spectrum (b) Worley's cellular function [Worley 1996] using 3 points per lattice cell (left) versus cellular wave noise (right).

pre-filter the tabulated waves (e.g. to MIPmap the tables T) and to fetch them at the proper resolution. Conversely, when the combination is non linear (e.g. in cellular noise), or a transfer function is applied, it remains for a future work. Moreover, while it achieves procedural noise synthesis from a single 2D image, our design solution to create solid wave noise is limited to Gaussian processes. This limitation arises from its reliance on random phases, which disrupt structural components (added only via specific transfer functions) and prevent the generation of structured 3D textures from examples. Additionally, our by-example formulation considers frequencies only on axis-aligned planes, leading to wavy artifacts in isotropic cases. Lastly, while 2D-exemplar-based 3D texture synthesis could help precomputing a full 3D PSD via FFT and improve frequency coverage, this is left for future work.

8 Conclusion

We introduced a novel procedural model for complex valued, non-periodic noises. We used hyperplanar waves randomly oriented in n-D space, stored in 1D tables for efficient computation and minimal memory use. It enables custom power spectra by varying waveforms and supports novel non-Gaussian random fields and cellular patterns beyond Gabor, Phasor, and Worley's approaches. Our wave noise seamlessly integrates into procedural material modeling tools, facilitating infinite, resolution-independent volumetric and surface materials to be designed and applied to surfaces without UV coordinates. Wave noise allows spectral control and is more efficient in 2D, 3D and higher dimensions compared to Gabor sparse convolution, since it relies on pre-computations.

Our insight of combining Monte Carlo and precomputation-based approaches represents an initial step toward enhancing the performance of multidimensional noises while maintaining low memory usage and with precise spectral control. As performance still remains higher when accessing an entire volume stored as a 3D texture, future work could explore incorporating additional precomputation stored in 2D tables instead of 1D tables, e.g. still without relying on full 3D tiles. Additionally, extending the example-based design to the support of structural components would be an excellent extension. Improving animations to combine them with physical simulations would last be a really interesting area for future extension.

References

Adib Akl, Charles Yaacoub, Marc Donias, Jean-Pierre Da Costa, and Christian Germain. 2018. A survey of exemplar-based texture synthesis methods. Computer Vision and Image Understanding 172 (2018), 12–24.

H. Baatz, J. Granskog, M. Papas, F. Rousselle, and J. Novák. 2022. NeRF-Tex: Neural Reflectance Field Textures. Computer Graphics Forum 41, 6 (2022), 287–301. doi:10.1111/cgf.14449

Guillaume Baldi, Rémi Allègre, and Jean-Michel Dischler. 2023. Differentiable point process texture basis functions for inverse procedural modeling of cellular stochastic structures. Computers & Graphics 112 (2023), 116–131.

Connelly Barnes and Fang-Lue Zhang. 2017. A survey of the state-of-the-art in patch-based synthesis. Computational Visual Media 3, 1 (March 2017), 3–20.

Robert Bridson, Jim Houriham, and Marcus Nordenstam. 2007. Curl-noise for procedural fluid flow. ACM Transactions on Graphics (ToG) 26, 3 (2007), 46–es.

Stefan Bruckner and M Eduard Gröller. 2007. Style transfer functions for illustrative volume rendering. In Computer Graphics Forum, Vol. 26. Wiley, 715–724.

Arthur Cavalier, Guillaume Gilet, and Djamchid Ghazanfarpour. 2019. Local spot noise for procedural surface details synthesis. Computers & Graphics 85 (2019), 92–99.

Robert L Cook and Tony DeRose. 2005. Wavelet noise. ACM Transactions on Graphics (TOG) 24, 3 (2005), 803–811.

Richard Cowan. 2010. New Classes of Random Tesselations Arising from Iterative Division of Cells. Advances in Applied Probability 42, 1 (2010), 26–47. http://www.jstor.org/stable/25683805

David S Ebert, F Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2003. Texturing & modeling: a procedural approach. Morgan Kaufmann.

Randima Fernando et al. 2004. GPU gems: programming techniques, tips, and tricks for real-time graphics. Vol. 590. Addison-Wesley Reading.

Romain Fournier and Basile Sauvage. 2024. Mix-Max: A Content-Aware Operator for Real-Time Texture Transitions. Computer Graphics Forum 43, 6 (2024).

Mathieu Gaillard, Bedrich Benes, Eric Guérin, Eric Galin, Damien Rohmer, and Marie-Paule Cani. 2019. Dendry: A procedural model for dendritic patterns. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 1–9.

Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor noise by example. ACM Transactions on Graphics (ToG) 31, 4 (2012), 1–9.

Bruno Galerne, Arthur Leclaire, and Lionel Moisan. 2017. Texton noise. In Computer Graphics Forum, Vol. 36. Wiley Online Library, 205–218.

Geoffrey Y Gardner. 1985. Visual simulation of clouds. In Proceedings of the 12th annual conference on Computer graphics and interactive techniques. 297–304.

Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. 2012. Multiple kernels noise for improved procedural texturing. The Visual Computer 28 (2012), 679–689.

Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. 2014. Local random-phase noise for procedural texturing. ACM Transactions on Graphics (ToG) 33, 6 (2014), 1–11.

Alexander Goldberg, Matthias Zwicker, and Frédo Durand. 2008. Anisotropic noise. ACM Transactions on Graphics (TOG) 27, 3 (2008), 1–8.

Charline Grenier, Basile Sauvage, J-M Dischler, and Sylvain Thery. 2022. Color-mapped noise vector fields for generating procedural micro-patterns. In Computer Graphics Forum, Vol. 41. Wiley Online Library, 477–487.

Pascal Guehl, Rémi Allegre, J-M Dischler, Bedrich Benes, and Eric Galin. 2020. Semi-Procedural Textures Using Point Process Texture Basis Functions. Computer Graphics Forum 39, 4 (2020), 159–171.

Eric Heitz and Fabrice Neyret. 2018. High-performance by-example noise using a histogram-preserving blending operator. Proceedings of the ACM on Computer Graphics and Interactive Techniques 1, 2 (2018), 1–25.

Yi-Hua Huang, Yan-Pei Cao, Yu-Kun Lai, Ying Shan, and Lin Gao. 2023. NeRF-Texture: Texture Synthesis with Neural Radiance Fields. In ACM SIGGRAPH 2023 Conference Proceedings (Los Angeles) (SIGGRAPH '23). Association for Computing Machinery, New York, NY, USA, Article 43, 10 pages. doi:10.1145/3588432.3591484

Yi-Hua Huang, Yan-Pei Cao, Yu-Kun Lai, Ying Shan, and Lin Gao. 2024. NeRF-Texture: Synthesizing Neural Radiance Field Textures. IEEE Transactions on Pattern Analysis and Machine Intelligence (2024), 1–15. doi:10.1109/TPAMI.2024.3382198

Daniel Hug and Rolf Schneider. 2024. Poisson Hyperplane Tessellations. Springer Cham. doi:10.1007/978-3-031-54104-9 Published: 24 May 2024, Hardcover. Softcover ISBN: 978-3-031-54106-3 (Due: 07 June 2025).

Pol Jeremias and Inigo Quilez. 2014. Shadertoy: learn to create everything in a fragment shader. In SIGGRAPH Asia 2014 Courses. ACM. doi:10.1145/2659467.2659474

A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, and M. Zwicker. 2010. A Survey of Procedural Noise Functions. Computer Graphics Forum 29, 8 (2010), 2579–2600. doi:10.1111/j.1467-8659.2010.01827.x

Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. 2009. Procedural noise using sparse Gabor convolution. ACM Transactions on Graphics (TOG) 28, 3 (2009), 1–10.

John-Peter Lewis. 1984. Texture synthesis for digital painting. In Proceedings of the 11th annual conference on Computer graphics and interactive techniques. 245–252.

John-Peter Lewis. 1989. Algorithms for solid noise synthesis. In Proceedings of the 16th annual conference on Computer graphics and interactive techniques. 263–270.

Nicolas Lutz, Basile Sauvage, and Jean-Michel Dischler. 2021. Cyclostationary Gaussian noise: theory and synthesis. In Computer Graphics Forum, Vol. 40. Wiley Online

Library, 239–250.
Nicolas Lutz, Basile Sauvage, and Jean-Michel Dischler. 2023. Preserving the autocovariance of texture tilings using importance sampling. In Computer Graphics Forum, Vol. 42. Wiley Online Library, 347–358.
Arman Maesumi, Dylan Hu, Krishi Saripalli, Vladimir G. Kim, Matthew Fisher, Sören Pirk, and Daniel Ritchie. 2024. One Noise to Rule Them All: Learning a Unified Model of Spatially-Varying Noise Patterns. ACM Transactions on Graphics (TOG) (2024). doi:10.1145/3658195
Werner Nagel, MECKE JOSEPH, Joachim Ohser, and Viola Weiss. 2008. A tessellation model for crack patterns on surfaces. Image Analysis and Stereology 27 (06 2008). doi:10.5566/ias.v27.p73-78
Ken Perlin. 1985. An image synthesizer. In Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85). Association for Computing Machinery, New York, NY, USA, 287–296. doi:10.1145/325334.325247
Ken Perlin. 2001. Noise Hardware. In Real-Time Shading SIGGRAPH Course Notes, M. Olano (Ed.). Chapter 9.
Nico Pietroni, Paolo Cignoni, Miguel Otaduy, and Roberto Scopigno. 2010. Solid-Texture Synthesis: A Survey. IEEE Comput. Graph. Appl. 30, 4 (jul 2010), 74–89. doi:10.1109/MCG.2009.153
Georgios Sakas. 1993. Modeling and animating turbulent gaseous phenomena using spectral synthesis. The Visual Computer 9 (1993), 200–212.
Bruce Schachter and Narendra Ahuja. 1979. Random pattern generation processes. Computer Graphics and Image Processing 10, 2 (1979), 95–114.
Andrew Schneider. 2023. NUBIS3 Methods (and madness) to model and render immersive real-time voxel-based clouds.. In Advances in Real-Time Rendering in Games Course (SIGGRAPH '23). Association for Computing Machinery, Los Angeles, USA.
Sheldon Taylor, Owen Sharpe, and Jiju Peethambaran. 2021. Prime gradient noise. Computational Visual Media 7, 3 (Feb. 2021), 349–362. doi:10.1007/s41095-021-0206-z
Thibault Tricard, Semyon Efremov, Cédric Zanni, Fabrice Neyret, Jonàs Martínez, and Sylvain Lefebvre. 2019. Procedural phasor noise. ACM Transactions on Graphics (TOG) 38, 4 (2019), 1–13.
Jarke J Van Wijk. 1991. Spot noise texture synthesis for data visualization. In Proceedings of the 18th annual conference on Computer graphics and interactive techniques. 309–318.
Steven Worley. 1996. A cellular texture basis function. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 291–294.