Computational Visual Media
https://doi.org/10.1007/s41095-021-0206-z

Research Article

Prime gradient noise

Sheldon Taylor[1,*], Owen Sharpe[1,*], and Jiju Peethambaran[1] (✉)

Abstract Procedural noise functions are fundamental tools in computer graphics used for synthesizing virtual geometry and texture patterns. Ideally, a procedural noise function should be compact, aperiodic, parameterized, and randomly accessible. Traditional lattice noise functions such as Perlin noise, however, exhibit periodicity due to the axial correlation induced while hashing the lattice vertices to the gradients. In this paper, we introduce a parameterized lattice noise called prime gradient noise (PGN) that minimizes discernible periodicity in the noise while enhancing the algorithmic efficiency. PGN utilizes prime gradients, a set of random unit vectors constructed from subsets of prime numbers plotted in polar coordinate system. To map axial indices of lattice vertices to prime gradients, PGN employs Szudzik pairing, a bijection $F : \mathbb{N}^2 \to \mathbb{N}$. Compositions of Szudzik pairing functions are used in higher dimensions. At the core of PGN is the ability to parameterize noise generation though prime sequence offsetting which facilitates the creation of fractal noise with varying levels of heterogeneity ranging from homogeneous to hybrid multifractals. A comparative spectral analysis of the proposed noise with other noises including lattice noises show that PGN significantly reduces axial correlation and hence, periodicity in the noise texture. We demonstrate the utility of the proposed noise function with several examples in procedural modeling, parameterized pattern synthesis, and solid texturing.

Keywords procedural noise; procedural texture; Perlin noise; prime numbers

1 Introduction

Visually pleasing 3D content is one of the core ingredients of successful movies, video games, and virtual reality applications. Unfortunately, creation of high quality virtual 3D content and textures is a labour-intensive task, often requiring several months to years of skilled manual labour. A relatively cheap yet powerful alternative to manual modeling is procedural content generation using a set of rules, such as L-systems [1] or procedural noise [2]. Procedural noise has proven highly successful in creating computer generated imagery (CGI) consisting of 3D models exhibiting fine detail at multiple scales. Furthermore, procedural noise is a compact, flexible, and low cost computational technique to synthesize a range of patterns that may be used to texture 3D models.

In general, procedural noise functions fall into three main categories: lattice noise, explicit noise, and sparse convolution noise [3]. A well-known example of lattice noise is the gradient noise introduced by Ref. [2], which is included in many commercial rendering engines such as Terragen [4] and Unity [5]. The popularity of Perlin noise can be attributed to its simplicity and efficiency. Perlin noise employs a hash function using a single random permutation table on the integer lattice to generate a fixed random unit gradient at each vertex. The generated gradients are then used to compute noise values for the entire domain [2, 6]. A sample image of Perlin noise and its Fourier transform is shown in Fig. 1(a).

A hash function used in lattice noise should ideally decorrelate the indices of the lattice vertices to avoid anisotropy and periodicity. However, Kensler et al. [7] found striations in the Fourier transform of Perlin noise, indicating axial correlation. They modified Perlin's 3D hash function to use three independent

* Sheldon Taylor and Owen Sharpe contributed equally to this work.

1 Graphics & Spatial Computing Lab, Dept. of Math & CS, Saint Mary's University, Halifax, NS, Canada, B3H3C3. E-mail: S. Taylor, sheldontaylor.7@gmail.com; O. Sharpe, owensharpe19@gmail.com; J. Peethambaran, jiju.poovvancheri@smu.ca (✉).

Manuscript received: 2021-01-15; accepted: 2021-01-18

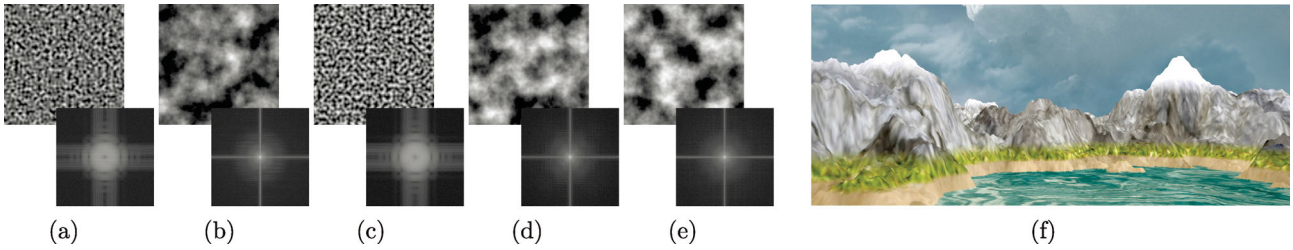


Fig. 1 Noise images (above) and Fourier transforms (below). (a) Perlin noise, (b) fractional Brownian motion (fBm) using Perlin noise, (c) prime gradient noise, (d) fBm using prime gradient noise, (e) fBm using additive cascades of parameterized PGN, (f) landscape generated using the noise texture in (e).

random permutation tables corresponding to each axis. Though the modified hashing scheme reduces the number of permutation table lookups from $2^{D+1}-2$ to $2D$ (where D is the dimension), the permutation tables consume three times as much space as the single permutation table in the original Perlin noise.

To capture fine details at multiple scales, Ref. [2] constructs *fractal noise* by summing several weighted "octaves" of noise. A sample image of fractal noise based on Perlin noise and its Fourier transform is shown in Fig. 1(b). While fractal noise successfully captures the repetitions of an underlying shape at different scales, it is statistically homogeneous (invariant under translation) and isotropic (invariant under rotation) [8]. This makes fractal noise unsuitable for virtual terrains with features such as valleys, foothills, or jagged alpine mountains. Hybrid multifractals [8] provide a way to generate heterogenous fractal noise by scaling octave values up or down by previous local octave values, which in turn can be used to synthesize terrains with a high degree of roughness. However, scaling octave values may lead to the formation of undesirable structures in the models such as depressions on mountain slopes [8]. Furthermore, the construction of hybrid multifractals involves costly multiplicative cascades.

In this paper, we propose two improvements to Perlin noise which lower storage requirements and reduce axial correlation. We name our algorithm *prime gradient noise* (PGN); sample images of PGN and fractal PGN noise and their Fourier transforms are shown in Figs. 1(c) and 1(d). The main improvement is the construction of lattice gradients using the distribution of the prime numbers as a source of randomness. Each prime number corresponds to a unique angle when taken mod 2π. A theorem of Vinogradov on prime number distribution [9] implies that these angles are distributed randomly and uniformly on $[0, 2\pi)$. In 2D, each "prime gradient" is computed as the unit vector with a given prime angle, and in 3D, a related, slightly more complicated formula produces a uniform distribution of prime gradients on the unit sphere from pairs of primes.

Unlike the fixed set of gradients prescribed in Ref. [6], our algorithm is parameterized by the choice of range from which to sieve the primes. Incidentally, the parameterization facilitates the generation of fractal noise with varying levels of heterogeneity. Essentially, different base noise functions are employed in the additive cascade of multifractals to generate a fractal noise that is a blend of homogeneous fBm and hybrid multifractals. This approach avoids the computationally expensive multiplicative cascade [8] of traditional hybrid multifractal systems. A sample image of this additive heterogeneous noise and its Fourier transform are shown in Fig. 1(e) and a sample landscape generated from this noise is shown in Fig. 1(f).

The other improvement tackles the problem of axial correlation in Perlin noise with a simple hashing scheme using Szudzik pairing [10]. A pairing function uniquely encodes two natural numbers into another natural number; we use the Szudzik pairing to map the lattice vertex indices to gradient vectors in 2D. As pairing functions can be composed to create bijections between the natural numbers and higher dimensions, our technique extends to 3D hashing as well. Instead of storing a predefined set of random unit vectors as in Perlin's original implementation [2], we compute the gradients on the fly from prime numbers. An implementation representing each prime with two bytes uses 1/4 the storage of Perlin noise gradients which represents a 2D gradient with two four-byte floats. The trade-off is the cost of computing the gradient from the prime, but memory access is typically a larger bottleneck than arithmetic.

In summary, we make the following specific contributions:

- *Prime gradients:* Utilization of prime numbers to generate a family of densely and uniformly distributed lattice gradients for noise generation. A parameterized version of the proposed base noise function can be used to create fractal noise with varying levels of heterogeneity.
- *Szudzik hashing:* A hashing scheme using Szudzik pairing to minimize the axial correlation found in the lattice noises.
- *OpenSN*: An *open source synthetic noise* library containing implementations of various noise algorithms including the proposed noise function along with analysis tools such as amplitude distribution graphs and periodograms, and algorithms for virtual geometry and pattern synthesis.

2 Previous work

Lagae et al. [3] classify procedural noise algorithms into three categories: lattice noise, sparse convolution noise, and explicit noise. In lattice noise, noise values are computed as a function of position and pseudorandom values defined at nearby lattice vertices. A sparse convolution noise function is the convolution of a sparse Poisson process and a kernel. Explicit noise functions use precomputed data (unlike the other two procedural-only categories) to compute the final noise values. Any noise can be sampled at varying rates and offsets and summed with varying weights to obtain fractal noise [2]. Wang tiles can be used to avoid periodicity of noise functions [11, 12].

2.1 Lattice noise

Probably the best-known procedural noise algorithm is Perlin noise [2]. It uses a hash function to define pseudorandom gradients at each point on the integer lattice and interpolates their dot products with difference-from-point vectors to obtain the noise value at each point. The hash function and interpolation function of Perlin noise are improved in later papers [6, 7], and hardware versions of Perlin noise are designed and/or implemented by Refs. [13–17]. Flow noise [18] and curl noise [19] use Perlin noise to animate flowing liquid. Ref. [20] vastly generalizes Perlin noise to a family of polynomial noise algorithms.

There are other lattice noises which use differently-shaped lattices. Wyvill and Novins [21] present a skewed, more dense lattice. Simplex noise [22] uses a triangular or tetrahedral lattice. Worley noise [23] randomly fills space with "feature points" via a Poisson process. Space is then partitioned into regions whose points share the closest feature point, and each noise value is computed as a function of distance to the nearest few feature points.

2.2 Sparse convolution noise

Sparse convolution noise was introduced by Refs. [24–26]. The choice of kernel is straightforward given a description of the desired power spectrum, for the spectra of the kernel and noise differ only by a constant multiple [26]. Spot noise [27] is produced by kernels whose graphs are disks, ellipses, and more complicated shapes. It is common to provide the sparse Poisson process of spot noise via a table of precomputed random values rather than procedurally. Ref. [28] defines a sparse convolution noise based on a cubic filter which uses a lattice to generate "source" points. Local random-phase noise [29] is another sparse convolution noise using a lattice; local regions with distinct noise parameters are defined based on the lattice points. Texton noise [30] introduces another kernel defined from a small texture which reproduces "Gaussian textures".

Gabor noise [31, 32] is a recent, well-known sparse convolution noise using the Gabor kernel:

$$G(x,y)=K\mathrm{e}^{-\pi a^2(x^2+y^2)}\cos(2\pi F_0(x\cos\omega_0+y\sin\omega_0)) \tag{1}$$

to generate both isotropic and anisotropic noise. Variations on Gabor noise include random phase Gabor noise [33], bandwidth-quantized Gabor noise [34], and NPR Gabor noise [35]. A very recent development based on Gabor noise is phasor noise [36], the main idea of which is to separate the intensity and phase of Gabor noise and control them separately.

2.3 Explicit noise

Wavelet noise [37] is an explicit noise designed to bandlimit the power spectrum. The output noise image is computed as the difference between a white noise image R and a blurred copy of R. Anisotropic noise [38] filters the Fourier transform of a white noise image to obtain several orientable sub-bands, which are then used to construct a noise image with the desired power spectrum and orientation. Stochastic

subdivision [39, 40] is a technique to approximate Brownian noise via interpolation of pre-computed samples.

Example-based texture synthesis is the generation of new textures which emulate the noise characteristics of given example input. Several researchers [41, 42] have used "energy" to measure how well a texture matches another. A generative adversarial network is employed by Ref. [43] to generate larger textures from smaller ones. A PCA-based convolutional network is used by Ref. [44] to identify and reproduce texture features. Semi-procedural textures [45] are generated from a combination of explicit structure parametrization and colour information taken from a sample texture. An algorithm has been designed by Ref. [46] to select a small gallery of textures based on a short training process by a user.

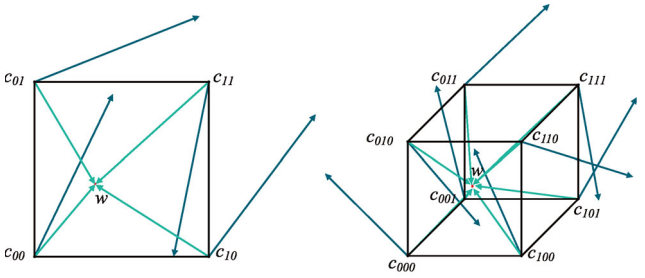
Table 1 Mathematical notation

Symbol	Definition
$\mathbb{N}$	The set of natural numbers
$\mathbb{P}$	The set of prime numbers
$\mathbb{Z}$	The set of integers
$\mathbb{R}$	The set of real numbers
$\oplus$	bitwise XOR

3 Prime gradient noise

3.1 Perlin noise

We start by reviewing Perlin noise [2, 6], one of the best-known procedural noise algorithms and highly relevant to the subsequent discussions. In Perlin noise, the domain of the noise function ($\mathbb{R}^n$) is implicitly tiled by the lattice of points with integer coordinates. A hash function $H : \mathbb{Z}^n \to \mathbb{Z}$ is defined which maps each lattice vertex to an index of an array G containing a set of predefined gradients (random unit vectors) in $\mathbb{R}^n$. At each point $\boldsymbol{w}$, the noise value $N(\boldsymbol{w})$ is computed by interpolating the dot products $G[H(c_{ij})] \cdot (\boldsymbol{w} - c_{ij})$ for each corner c_{ij} of the lattice cell containing $\boldsymbol{w}$. Gradients and the point vectors of a lattice cell in 2D as well as 3D are illustrated in Fig. 2. The interpolation process uses a weighting function to smooth the transition between lattice cells. The smoothstep function $f(t) = 3t^2 - 2t^3$ was used in the original version of the algorithm [2], and later replaced with the smootherstep function $f(t) = 6t^5 - 15t^4 + 10t^3$ [6]. A closely related noise



(a) Square cell (b) Cubic cell

Fig. 2 Gradient and point vectors of a lattice cell.

is simplex noise [22], which is defined on a triangular or tetrahedral lattice and uses a hash function to convert lattice points to gradients and interpolates dot products.

Perlin noise ultimately provides a function to procedurally generate noise at a given frequency and amplitude. *Frequency* controls the sampling period while *amplitude* controls the range in which values lie that are generated by the function. Utilizing Perlin noise in conjunction with fBm [8] allows multiple varied frequencies and amplitudes of Perlin noise to be conflated. Such a process generates interesting textures, often used to simulate clouds or height maps, as shown in Fig. 3.

3.2 Prime gradients

In this subsection, we discuss the generation of unit vectors in 2D and 3D from prime numbers. We use the notation $\theta(r)$ and $\{r\}$ to denote the floating-point remainder of r by 2π and 1, respectively (the latter is just the fractional part of r).

Weyl proved [47] the following theorem:

Theorem 3.1 (Weyl's Criterion). *Let $(x_n)_n$ be a sequence of real numbers. Then the sequence $(\{x_n\})_n$ is uniformly distributed (u.d.) in $[0, 1)$ iff for all $h \in \mathbb{Z}$:*

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{\infty} \mathrm{e}^{2\pi \mathrm{i} h x_n} = 0 \tag{2}$$

Weyl proved the special case that $(\{\alpha n\})_n$ is u.d. in $[0, 1)$ for irrational α. A number-theoretic result of Vinogradov [47] implies that $(\{\alpha p\})_{p\in\mathbb{P}}$ is also u.d. in $[0, 1)$. (The citations are taken from a text on

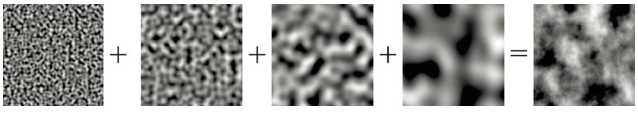


Fig. 3 fBm composition of Perlin noise.

uniform distribution of sequences, since Weyl and Vinogradov did not publish in English.)

Pick $\alpha = 1/2\pi$; since $(\{p/2\pi\})_{p\in\mathbb{P}}$ is u.d. in $[0,1)$, $(\theta(p))_{p\in\mathbb{P}}$ is u.d. in $[0,2\pi)$. This suggests a technique for pseudorandom generation of angles. While the sequence $(\theta(n))_n$ is also u.d. in $[0,2\pi)$, the irregular gaps between the prime numbers are better suited to a pseudo-random number generator (PRNG). There are many other integer sequences $(a_n)_n$ such that $(\theta(a_n))_n$ is u.d.; the primes are merely a toy example, and future work includes investigating easier-to-compute $(a_n)_n$ suitable for our PRNG.

Plots of the first few terms of the point sequences $((p\cos(\theta(p)),\ p\sin(\theta(p))))_{p\in\mathbb{P}}$ and $((\cos(\theta(p)),\ \sin(\theta(p))))_{p\in\mathbb{P}}$ in Figs. 4 and 5 respectively illustrate this uniform distribution.

The pre-process which generates the gradient table

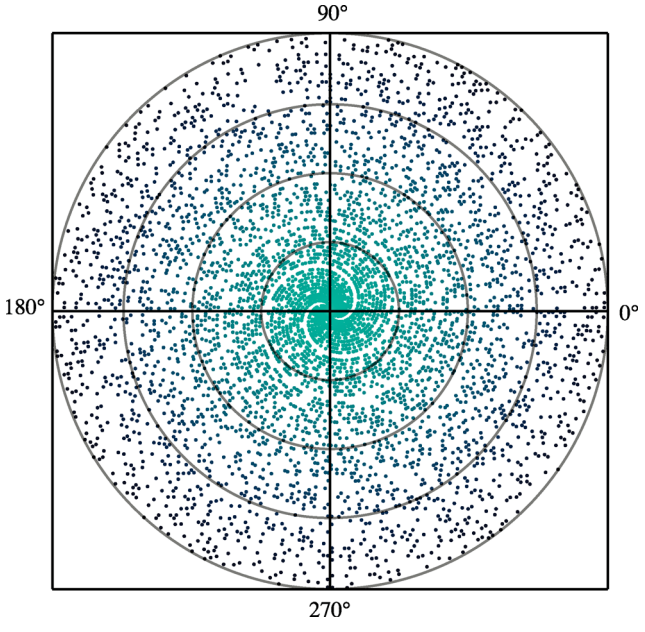
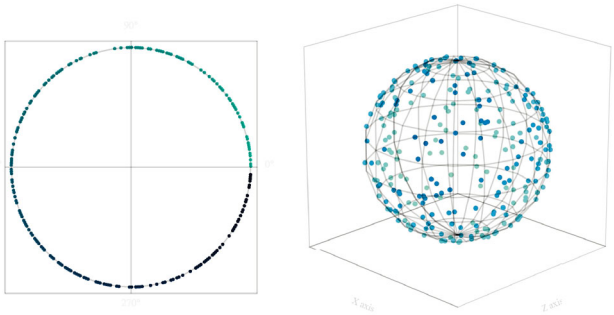


Fig. 4 Polar plot of primes.



(a) Unit circle (2D) (b) Unit sphere (3D)

Fig. 5 Plots of the first 256 normalized prime numbers.

must select a finite set of prime numbers, one per gradient. Our algorithm fixes an interval $[m,n]$ (the choice of m and n may be considered the "seed" for the PRNG) and runs an Eratosthenes sieve [48] to extract the prime numbers from that range. An alternative implementation might implicitly choose m and n and hardcode the prime table $[m,n]$ to avoid the sieving step. The time complexity of the sieve is discussed in Section 4.7.

In our 2D algorithm, each gradient is computed on the fly from each prime p as $(\cos(\theta(p)),\sin(\theta(p)))$. In our 3D algorithm, a table of pairs of primes (p,q) is constructed instead. To construct each gradient from a pair (p,q), an intermediate pair is constructed by the following formula (the set of such pairs is u.d. in $[-1,1)\times[-1,1)$):

$$(\cos(\theta),u) = \left(\cos(\theta(p)),\frac{\theta(q)}{\pi}-1\right) \tag{3}$$

The gradient is then constructed from this pair by the formula:

$$(x,y,z) = (\sqrt{1-u^2}\cos(\theta),\sqrt{1-u^2}\sin(\theta),u) \tag{4}$$

which gives a uniform distribution on the unit sphere [49]. In higher dimensions, PGN could implement techniques such as uniformly sampling points in the unit n-cube and discarding ones which lie outside the unit n-sphere.

3.3 Szudzik hashing

The hash functions $H_n : \mathbb{Z}^n \to \mathbb{Z}$ for 2D and 3D Perlin noise are implemented as

$$H_2(i,j) = \sigma[\sigma[i]+j] \tag{5}$$

$$H_3(i,j,k) = \sigma[\sigma[\sigma[i]+j]+k] \tag{6}$$

where σ is a pseudorandom permutation table and indices are taken modulo $|\sigma|$. Kensler et al. [7] demonstrate that this method contributes to axial correlation. They suggest the hash functions:

$$H_2(i,j) = \sigma[i]\oplus\tau[j] \tag{7}$$

$$H_3(i,j,k) = \sigma[i]\oplus\tau[j]\oplus\phi(k) \tag{8}$$

where σ,τ,ϕ are distinct permutation tables for the x,y,z axes respectively. The drawback to this method is the extra storage required.

We present alternative hash functions based on pairing functions which uses only one permutation table while avoiding axial correlation. A pairing is a bijection $F : \mathbb{N}^2 \to \mathbb{N}$, and can be plotted on $\mathbb{N}^2$ as a path consisting of discrete steps from each $F^{-1}(n)$ to $F^{-1}(n+1)$ (see Fig. 7).

Our proposed hash functions are implemented as

$$H_2(i,j) = \sigma[F(i,j)] \tag{9}$$

$$H_3(i,j,k) = \sigma[F(F(i,j),k)] \tag{10}$$

Note that since $F(i,j)$ is a bijection $\mathbb{N}^2 \to \mathbb{N}$, $F(F(i,j),k)$ is a bijection $\mathbb{N}^3 \to \mathbb{N}$. This composition of pairing functions allows us to extend our hashing technique to any number of dimensions. In practice, it is usually possible to force all sampled lattice points to have non-negative indices. If lattice points with negative indices are required, $F(i,j)$ can be replaced by $F(B(i),B(j))$ where $B : \mathbb{Z} \to \mathbb{N}$ is the bijection:

$$B(i) = \begin{cases} -2i, & i < 0 \\ 0, & i = 0 \\ 2i-1, & i > 0 \end{cases} \tag{11}$$

We compare three well-known pairing functions for use in the hash algorithm: the Cantor pairing [10]:

$$\text{cantor}(i,j) = i + \frac{(i+j)(i+j+1)}{2} \tag{12}$$

the Rosenberg-Strong pairing [50]:

$$\text{rosenbergstrong}(i,j) = \begin{cases} j^2 + i, & i < j \\ i^2 + 2i - j, & i \geqslant j \end{cases} \tag{13}$$

and the Szudzik pairing [10]:

$$\text{szudzik}(i,j) = \begin{cases} j^2 + i, & i < j \\ i^2 + i + j, & i \geqslant j \end{cases} \tag{14}$$

It is convenient to visualize pairing functions as lattice paths, where each step is from $F^{-1}(n)$ to $F^{-1}(n+1)$. The diagrams in Fig. 7 show that the Rosenberg-Strong and Szudzik pairings map lattice points in a given rectangular region to a smaller interval in $\mathbb{N}$ than the Cantor function. This is desirable in low-memory environments since larger numbers use more bits.

The zoomed version of the periodograms in Fig. 6 show that the spectra of Perlin noise with the Cantor and Rosenberg-Strong functions display diagonal streaks, while the spectrum of PGN and Perlin noise with the Szudzik pairing is free of such anomalies. The spectrum of the original Perlin noise also shows some more subtle horizontal streaks compared to PGN.

3.4 The algorithm

The PGN algorithm is parameterized by an "offset" parameter m determining which prime numbers are used to generate the gradients. We refer to the

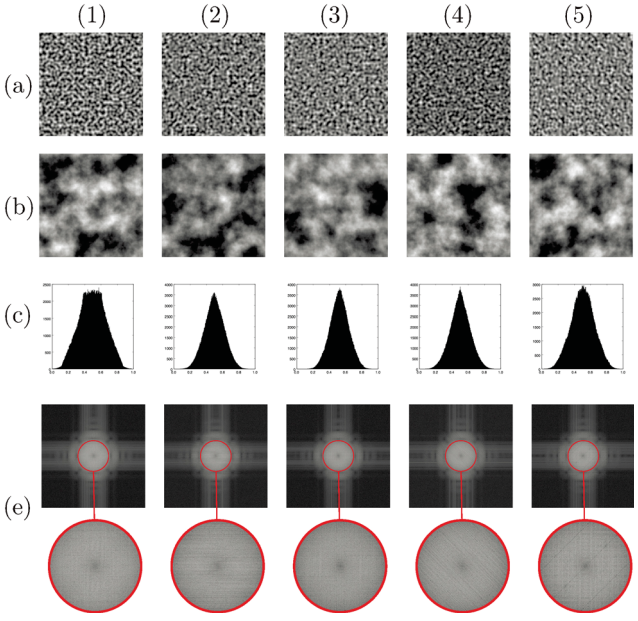


Fig. 6 Analysis of two dimensional lattice-based procedural functions. (a) Noise. (b) fBm noise with 8 octaves. (c) Amplitude distribution. (d) Periodogram with highlighted center region. (1) Prime gradient noise. (2) Perlin noise with Perlin's pairing function. (3) Perlin noise with Szudzik's pairing function. (4) Perlin noise with Cantor's pairing function. (5) Perlin noise with Rosenberg-Strong's pairing function.

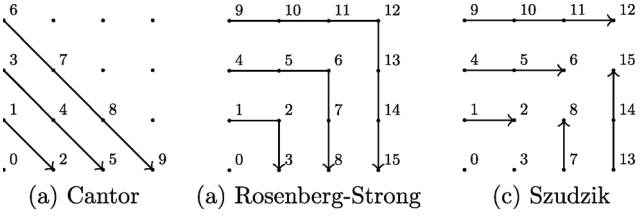


Fig. 7 Lattice paths corresponding to different pairing functions.

floating point remainder of p by 2π as $\theta(p)$ as in Section 3.2. We describe the gradient generation process here; the interpolation step is exactly the same as improved Perlin noise [6].

3.4.1 2D algorithm

A pre-process computes and shuffles a table of prime numbers for later gradient generation. First the Eratosthenes sieve [48] is run on $[2^{15}, 2^{16}]$ to mark all primes in that range (the list can also be explicitly provided, but the sieve is very fast for such small numbers). Then the primes at the 256 indices (allowing wraparound) starting with m are copied to an array P. Finally, P is shuffled using the Fisher-Yates algorithm [51], which generates all permutations of P with equal probability. Given a lattice vertex (i,j), the hash function computes $h = \text{szudzik}(i,j)$ and sets $p = P[h\%256]$. The gradient to be returned is computed as $(\cos(\theta(p)), \sin(\theta(p)))$.

3.4.2 3D algorithm

A pre-process extracts 256 primes, makes two copies of them in arrays P, Q, and shuffles P and Q into separate permutations (separating the permutations slightly increases the randomness). Given a lattice vertex (i, j, k), the hash function computes $h = \sigma[\text{szudzik}(\text{szudzik}(i, j), k)]$ and sets $p = P[h\%256]$, $q = Q[h\%256]$, $u = \theta(q)/\pi - 1$. The gradient to be returned is computed as $(\sqrt{1-u^2}\cos(\theta(p)), \sqrt{1-u^2}\sin(\theta(p)), u)$.

3.5 Heterogeneous noise via parameterized PGN

Up until this point, any use of prime gradient noise with fBm has been exclusively homogeneous and isotropic. Homogeneous fBm can be described as each point being treated the same in terms of how its value is generated, resulting in a consistent roughness throughout. However, it is evident that nature produces more heterogeneous patterns, as exemplified in Fig. 8 where higher elevations have a greater degree of roughness (e.g., tops of mountains) and lower elevations a lesser degree of roughness (e.g., ground beneath water). Moreover, the resulting fBm composition, detailed in Fig. 9, exemplifies the amplification of roughness at higher elevations and the smoothing at lower elevations. This can be generalized such that the noise value at a given point is ultimately influenced by nearby points, in turn resulting in gradual yet correlated transitions with respect to roughness [8].

Heterogeneous fBm is commonly achieved through the process of scaling octaves values such that lower noise values are scaled down and higher noise values are scaled up. Values can be scaled such that the current octave is multiplied by the current summation of noise values, in turn resulting in a greater degree

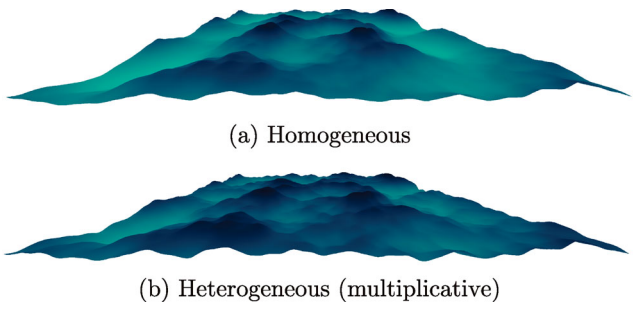


Fig. 8 Colour coded homogeneous and heterogeneous fBm noise, where darker blues represent higher elevations and lighter blues represent lower elevations.

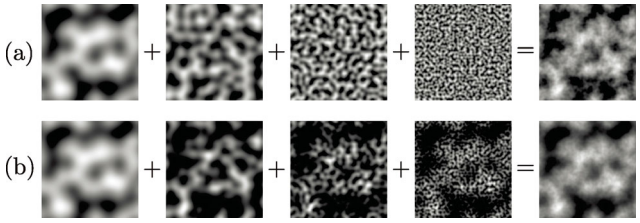


Fig. 9 Homogeneous and heterogeneous fBm compositions of prime gradient noise corresponding to the height maps in Fig. 8. (a) Homogeneous. (b) Heterogeneous (multiplicative).

of roughness at higher noise values and a lesser degree of roughness at lower noise values, resulting in multiplicative heterogeneous fBm [8]. Alternatively, a hybrid between homogeneous and heterogeneous fBm can be achieved using an additive cascade of parameterized PGN by allowing for different offsets at each octave as in Eq. (15):

$$fBm(p) = \sum_{i}^{N} PGN_i(p * 2^i) * 2^{-H*i} \tag{15}$$

where p is the point at which the function is evaluated, and N and H are the number of octaves and fractal increment parameter [8] respectively. This results in a sliding window approach to compute $PGN_i(.)$, whereby the range used in the gradient table is offset, dependent upon the current octave. The offset is generated as $2^i - 1$, such that $0 < i < N$ where i is the current octave and N is the total number of octaves. Consequently, this leads to the fact that an expanded prime table (in 2D), or tables (in 3D), are required to accommodate for the said offset. The size of the prime tables can be calculated to be $256 + 2^{N-1} - 1$, where N denotes the total number of octaves. The combination of these two aspects enables the usable range of the prime table or tables to shift, dependent upon on the current octave, as exemplified in Fig. 10.

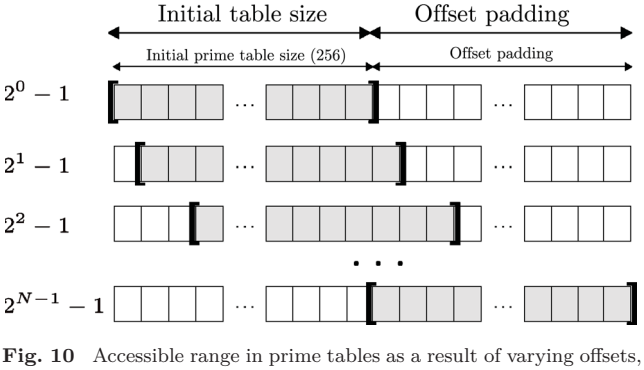


Fig. 10 Accessible range in prime tables as a result of varying offsets, where N denotes the total number of octaves and each grey section represents the accessible range of the prime table at a given octave. Prime table size is $256 + 2^{N-1} - 1$.

3.6 Comparison with Perlin noise

The overall algorithm of prime gradient noise generally resembles that of Perlin noise, preserving features such as aperiodicity and random accessibility. However, prime gradient noise differs from Perlin noise in two ways. These differences pertain to the method by which values in a lattice are generated, along with the method associated with how these values are used in noise generation. Rather than generating pseudo-random values for each lattice point and dimension, prime gradient noise utilizes a sequence of prime numbers that is randomly permuted such that a single table is needed in 2D and two tables are needed in 3D, which are referred to as prime tables. The parametrization element of this function relates to an offset that in turn specifies the range of prime numbers used, as exemplified in Section 3.5. Prime gradient noise reduces the number of times in which lattice values must be accessed. Where Perlin noise accesses two lattice values in 2D and three lattice values in 3D, prime gradient noise only requires a single lattice value to be accessed in 2D and two lattice values to be accessed in 3D. The resulting value or values in 2D and 3D, respectively, is subsequently used in the conversion of polar or spherical coordinates to Cartesian coordinates, allowing all influence vectors to be computed from a single lattice lookup in 2D or two lattice lookups in 3D, as described in Section 3.4. Consequently, the prime gradient method reduces the overall space requirements in relation to lattice values. Quantitatively, the number of per cell look-ups differs slightly between prime gradient noise and Perlin noise, which follows from the different number of lattice tables associated with each lattice point. Where Perlin noise results in $2^{D+1} - 2$ lookups, where D denotes the dimension, prime gradient noise reduces this number by a factor close to 2, such that the number of lookups amounts to 2^D.

4 Results and discussion

4.1 Preliminaries

We implemented the prime gradient noise algorithm in C++ using functions from OpenGL and OpenCV libraries. For comparison purposes, we also implemented related algorithms including Perlin [2, 6, 18], better gradient [7], Worley [23], wavelet [37], Gabor [31], and phasor [36] noise functions. An open source repository containing all these noise functions along with necessary analysis tools such as *amplitude distribution* and *periodogram* functions will be made publicly available soon.

We conducted different experiments to exemplify the impact of prime gradient noise on various graphical entities. All experiments were performed on a machine having a 2.3 GHz processor with 8 GB of RAM. Throughout each experiment, prime gradient noise was contrasted both in its parameterized and non-parameterized forms. Through its parameterized fBm form, prime gradient noise uses varying offsets at each octave, where the offset for an octave $i \leqslant N$, is calculated using $2^i - 1$, where N denotes the number of octaves.

4.2 Comparative analysis

In this section, we analyse and compare prime gradient noise with other lattice noise functions (Perlin [2, 6], and better gradient [7], an explicit noise (wavelet [37]), and Gabor [31] noise). We have also included Perlin with Szudzik hashing for comparison. The qualitative analysis is performed by estimating the power spectrum through *periodograms* and *amplitude distributions*. Amplitude distribution is estimated through the use of a histogram. Periodograms are generated by way of the Fourier transform, such that a given pixel value is represented by the squared magnitude of the Fourier transform [3].

Figure 11 shows the analysis of various procedural noise functions. The top row displays the 2D noise generated by the various noise functions. Evidently, the outputs yield similar results for prime gradient noise, Perlin noise, Perlin noise with the Szudzik pairing function, and the wavelet noise. Continuing, better gradient noise and Gabor noise yield similar results to one another. Both of these noises demonstrate higher aliasing effects than the previous noise textures.

The second row in Fig. 11 displays the amplitude distributions. It is evident that Perlin noise, Perlin noise with the Szudzik pairing function, better gradient noise, Gabor noise, and wavelet noise display similar Gaussian distributions of noise values. These distributions differ slightly from that of prime gradient noise. Prime gradient noise prooduces a flat-shaped appearance at an amplitude of 0.5. This simply results in the even distribution of

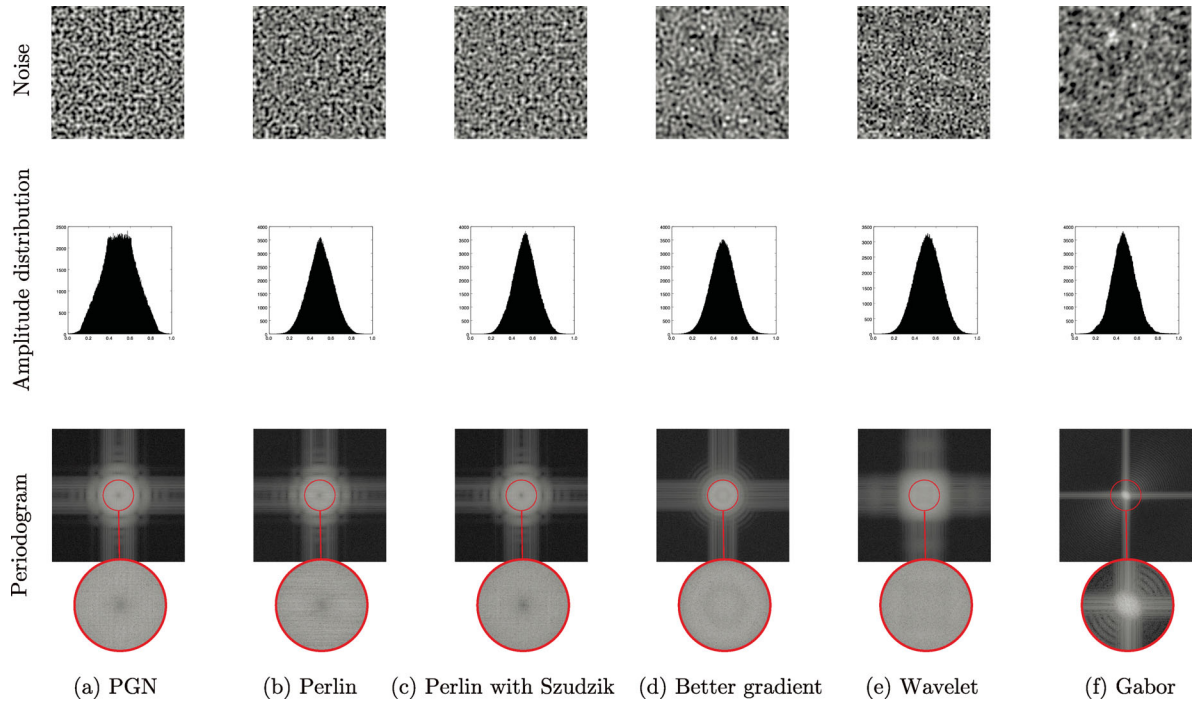


Fig. 11 Comparison of various procedural noise functions. Subtle streaks in the zoomed version of Perlin noise periodogram are absent in the prime gradient noise implying a reduction in the axial correlations.

moderate amplitude values between 0.4 and 0.6, thereby resulting in a lower variance than Gaussian distributions. While a lower variance implies a lesser extent of dispersion within the set of amplitude values, this set still generates desirable effects comparable to that of lattice noises with Gaussian distributions.

The bottom row of Fig. 11 shows the periodogram, generated as the magnitude squared of the Fourier transform. Prime gradient noise, Perlin noise, and Perlin noise with the Szudzik pairing function produce similar outputs. Upon closer investigation, it is evident that prime gradient noise and Perlin noise with the Szudzik pairing function can be differentiated from Perlin noise through the horizontal lines emitted in the latter. This is the result of axial correlation in the Perlin noise hashing scheme, which is quite minimal in prime gradient noise and Perlin noise with the Szudzik pairing function. Similarly, better gradient noise and Gabor noise generally resemble one another. This being said, the notable differences in the size of the center circle and the lines projecting on the horizontal and vertical axes are a result of greater detail witnessed in better gradient noise. Although different in its shape, wavelet noise shows little axial correlation at its center, much like prime gradient noise and Perlin noise with the Szudzik function.

4.3 Parametrized textures

Parametrization refers to an aspect of procedural noise generation where perturbation of inputs yields distinct and differentiable outcomes [31]. Such a technique allows for broad applications of generic noise functions. Prime gradient noise can be parameterized by perturbation of noise values the function returns or through the specification of an offset in 2D noise generation (or various offsets in the case of 3D noise generation). As exemplified in Fig. 12, parameterized prime gradient noise is used to perturb noise values such that various textures are able to be generated. Row 1 in Fig. 12 demonstrate the noise images. Row 2 in Fig. 12 resembles a procedurally generated cloud texture, obtained by simply colouring the output of fBm noise such that the noise value is used to scale colours between white and blue. Row 3 in Fig. 12 shows lava or fire texture, which is generated by returning the noise value as $1 - |noise|$ and converting the corresponding noise value to a coloured output. Finally, row 4 in Fig. 12 displays a wood texture generated by returning the noise value as the fractional component (i.e., mod 1) of $noise \cdot \sqrt{(i \cdot 0.1 + 10)^2 + (j \cdot 0.1 + 10)^2}$, where (i, j) corresponds to the point passed as an argument to the 2D prime gradient noise function. From the

various textures in Fig. 12, it is quite apparent that specifying various prime number offsets results in textures that clearly differ yet provide the same general appearance.

4.4 Procedural modeling

In this section, we qualitatively and quantitatively evaluate the efficacy of prime gradient noise in procedural modeling. While the qualitative evaluation is mainly based on surface roughness, we use relative vertical vertex displacement for the quantitative evaluation. Geometrically, the roughness of a surface captures the extent to which a surface appears uneven or irregular and can be characterized through the deviation in the direction of real surface normal from its ideal form. Large normal deviations imply that the surface is rough whereas small deviations indicate a smooth surface. As an example,

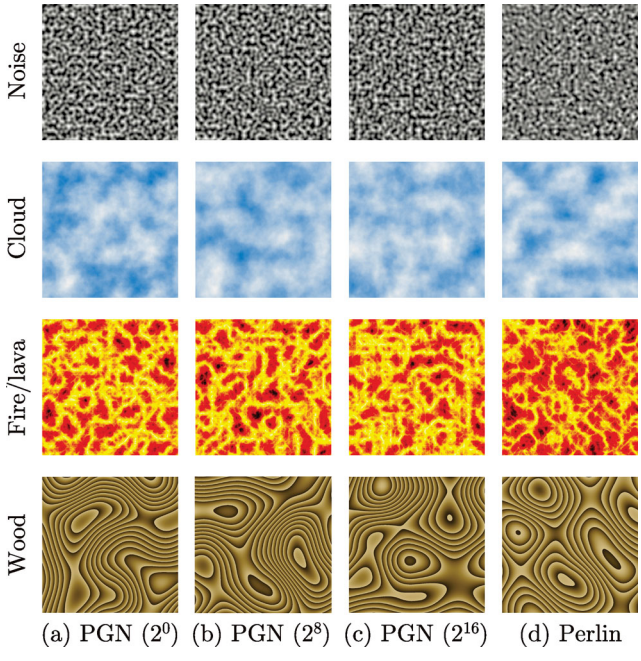


Fig. 12 Comparison of various patterns synthesized by prime gradient noises at various offsets (denoted in brackets) in contrast to the same patterns synthesized by Perlin noise.

the slopes of the mountains in Fig. 13(c) have a rougher surface than those displayed in Figs. 13(a) and 13(b). Homogeneous prime gradient noise is constructed from prime gradient noise with a fixed offset and no multiplicative cascading. The resulting output, as displayed in Fig. 13(a), displays consistent appearance in terms of roughness. Qualitatively, this differs from the results in Figs. 13(b) and 13(c), where heterogeneous prime gradient noise is used to manipulate the roughness in correlated areas. In observing the differences between Figs. 13(a)–13(c), it can be noted that the homogeneous model in Fig. 13(a) exhibits a consistent roughness throughout. Comparatively, a higher degree of roughness is displayed in both heterogeneous models, Figs. 13(b) and 13(c).

Prime gradient noise with additive heterogeneity works in tandem with the frequency at a given octave, which defines the level of detail exhibited in the noise. This is conducted through the use of a multiplicative factor at each octave, multiplying the point to sample by the octave's frequency. The frequency aspect is further enhanced through the specification of an offset, which shifts the usable range of the prime table. Given that fBm is additive in nature through the summation of noise values in each octave, it results in homogeneous noise that combines an octave's frequency with point to sample. To do so, additive heterogeneous fBm also works by multiplying the point to sample by the octave's frequency, while additionally sampling different regions of the prime table depending on the current octave. This results in increased roughness at moderate noise values and decreased surface roughness at high and low noise values, as exemplified in Fig. 14. This is computationally beneficial: no multiplicative factor was used to achieve such a result. Through the use of varying offsets for each octave, prime gradient noise has the ability to produce a blend of homogeneous and heterogeneous noise. This can prove advantageous

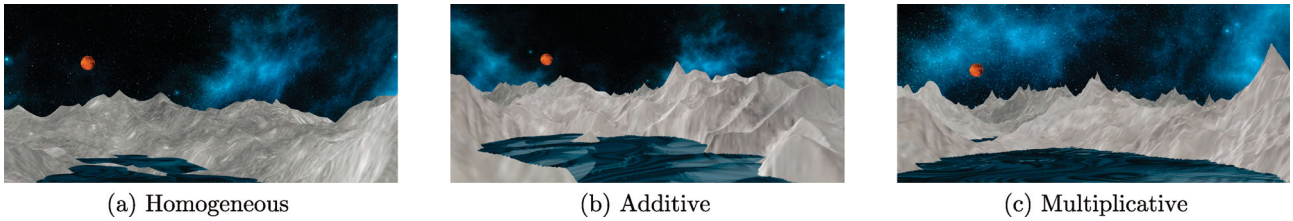


Fig. 13 Procedural landscape modelled using prime gradient noise with homogeneous fBm and additive and multiplicative heterogeneous fBm, where prime gradient noise is parametrized with 4 varying offsets and utilizes fBm with 4 octaves.

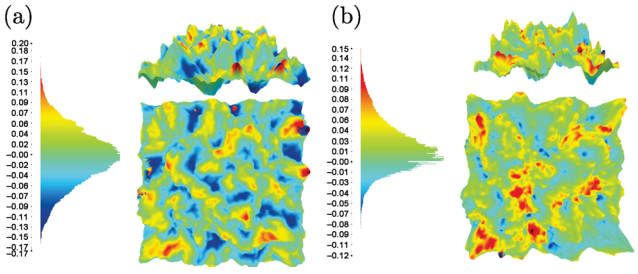


Fig. 14 Noise (amplitude) displacement between various fBm implementations. (a) Displacement between heterogeneous (additive) prime gradient noise with varying offsets at each octave and homogeneous prime gradient noise with fixed offset at each octave, each of which was generated using fBm with 4 octaves. (b) Displacement between heterogeneous (additive) prime gradient noise with varying offsets at each octave and heterogeneous (multiplicative) prime gradient noise with with a varying offsets at each octave. All height maps were generated using fBm with 4 octaves.

in uses such as procedural terrain models, where mountain peaks and valleys exhibit less roughness.

4.5 Surface texturing

Three major methods exist to generate noise on a given surface: 2D mapping, solid noise, and surface noise [3]. 2D mapping projects 2D noise onto a 3D surface. Unfortunately, this approach can cause seams and artifacts on the surface since 2D noise is intended to texture a plane and does not map onto a complex 3D shape gracefully. Solid noise [2] is 3D noise, and the observed surface texture is simply the part of the texture which lies on the object boundaries. However, solid noise is often distorted since solid noise is not designed to be sampled along a complicated surface. Surface noise uses 2D noise, but renders the texture directly onto the surface, taking the shape of the surface into account [3].

We present 2D mappings of prime gradient noise in Figs. 15 and 16. Figure 15 demonstrates basic UV mapping around a given mesh and Fig. 16 demonstrates a parallel occlusion implementation and its impact on various textures on the sphere.

Fig. 15 A gallery of 3D models textured using parameterized PGN.

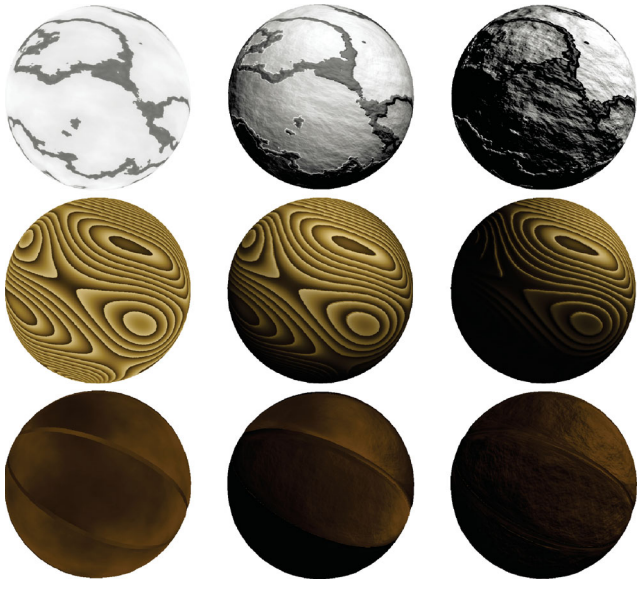


Fig. 16 2D mapping with dynamic topology and utilization of parallax occlusion mapping. Left to right: incresing topological displacement. Above: prime gradient noise parametrized with a marble perturbation, middle: prime gradient noise parametrized with a wood perturbation, and below: prime gradient noise and utilizing additive heterogeneous fBm, all utilizing additive cascades of parameterized PGN.

4.6 Solid noise - Fig. 17

The images in Fig. 17 illustrate a solid (3D) noise texture applied to a cube. The textures in these images are $256 \times 256 \times 256$ texels and span sixteen noise cells along each axis. We have also experimented with using prime gradients and Szudzik hashing in simplex noise. The result is shown in the rightmost column of the figure. Interestingly, the solid textures for prime gradient noise and prime simplex noise seem to have a more subtle pattern than their counterparts. The difference between Perlin noise and prime gradient noise is especially pronounced; the former displays prominent streaks while the latter is subdued.

4.7 Speed

Prime gradient noise is much faster in our tests than most popular noises except Perlin noise: see Fig. 18. Indeed, Perlin noise is only slightly faster than PGN

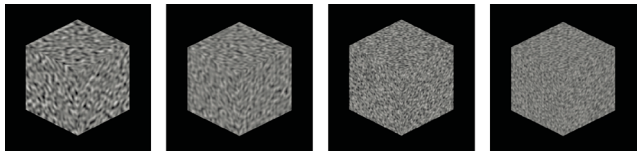


Fig. 17 Solid noise texture applied to cubes. From left to right: Perlin noise, prime gradient noise, simplex noise, and prime simplex noise.

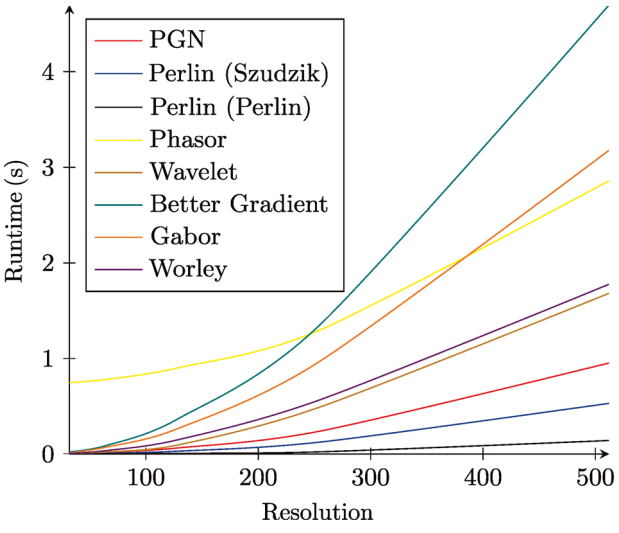


Fig. 18 Resolution versus runtime plot for various 2D noises. Runtime includes pre-processing time (gradient generation, prime sieving, etc.)

when the Perlin hash is replaced by the Szudzik hash.

Sieving the primes up to n has a time complexity of $O(n \ln \ln n)$ [48]. Assuming a bit array is used to sieve for the primes less than 2^{16}, the array takes up 4 KiB (2^{16} bits). There are 6542 primes up to 2^{16}, occupying 13 KiB at 16 bits each (stored as integers) or 26 KiB at 32 bits each (stored exactly as floating point numbers). It is feasible to hardcode this small table of primes, though the sieve runtime for a limit of 2^{16} is negligible.

Each call to the noise function takes constant time, so texture generation time is linear in the number of texels. The gradient generation from each prime is rather expensive, requiring several floating point operations. On the other hand, the Szudzik hash is hardly more expensive than nested permutation table lookup or the XOR permutation index combination of Ref. [7], and requires no space.

4.8 Limitations

Our algorithm does not scale well to large ranges of primes. The time complexity of sieving the range $[0, n]$ is $O(n \ln \ln n)$, and primes larger than 2^{16} require more storage. However, in practice, small ranges of primes suffice for synthesizing gradients. The floating point operations to convert primes to gradients are expensive and evaluated more than once per vertex to reduce storage space. Just like Perlin noise, PGN is periodic in both axes in large samples due to the implicit modulus applied to the gradient table index. This can be avoided by techniques such as Wang tiling [12]. We believe that the simplicity and performance of PGN outweigh these flaws, which we have not observed in practice.

5 Conclusions

We have introduced prime gradient noise (PGN), a lattice noise using random unit vectors constructed from polar plotted prime sequences and a Szudzik pairing function. Subsets of primes, randomly yet uniformly distributed in $[0, 2\pi]$, not only increase the randomness in the resulting noise, but also facilitate the parameterization of the noise function though offsetting. As evident in the spectral analysis, using Szudzik pairing as a hashing scheme succeeds in minimizing any discernible axial correlation of the kind found in traditional lattice noise such as Perlin noise [2]. Furthermore, we have demonstrated that fBm composition through parameterized PGN is highly useful for modeling virtual terrains with varying levels of roughness, controlled by offsetting, without using computationally intensive multiplicative cascades. Experimental results indicate that our noise function produces results that are comparable to other lattice noise functions. The proposed noise function is comparable in terms of performance and therefore, suitable for procedural modeling and texturing of various graphical entities. Further theoretical analysis on how well Szudzik pairing decorrelates the indices is indicated.

Acknowledgements

This work is supported by the National Science and Engineering Research Council of Canada (NSERC) Discovery Grant No. 2019-05092.

References

[1] Lindenmayer, A. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology* Vol. 18, No. 3, 280–299, 1968.

[2] Perlin, K. An image synthesizer. *ACM SIGGRAPH Computer Graphics* Vol. 19, No. 3, 287–296, 1985.

[3] Lagae, A.; Lefebvre, S.; Cook, R.; DeRose, T.; Drettakis, G.; Ebert, D. S.; Lewis, J. P.; Perlin, K.; Zwicker, M. A survey of procedural noise functions. *Computer Graphics Forum* Vol. 29, No. 8, 2579–2600, 2010.

[4] Planetside Software. Perlin 3D scalar. 2020. Available at https://planetside.co.uk/wiki/index.php?title=Perlin_3D_Scalar.

[5] Unity. Mathf.perlinnoise. 2020. Available at https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html.

[6] Perlin, K. Improving noise. *ACM Transactions on Graphics* Vol. 21, No. 3, 681–682, 2002.

[7] Kensler, A.; Knoll, A.; Shirley, P. Better gradient noise. SCI Institute Technical Report, UUSCI-2008-001, 2008.

[8] Ebert, D. S.; Musgrave, F. K.; Peachey, D.; Perlin, K.; Worley, S. *Texturing and Modeling: A Procedural Approach*, 3rd edn. Morgan Kaufmann, 2003.

[9] Vinogradov, I. M. *Selected Works*. Springer-Verlag Berlin Heidelberg, 1985.

[10] Szudzik, M. An elegant pairing function. In: Proceedings of the NKS Wolfram Science Conference, 2006.

[11] Wei, L. Y. Tile-based texture mapping on graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 55–63, 2004.

[12] Kirillov, A. Non-periodic tiling of procedural noise functions. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* Vol. 1, No. 2, 1–15, 2018.

[13] Olano, M. Modified noise for evaluation on graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 105–110, 2005.

[14] Hart, J.; Carr, N.; Kameya, M.; Tibbitts, S.; Coleman, T. Antialiased parameterized solid texturing simplified for consumer-level hardware implementation. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, 45–53, 1999.

[15] Hart, J. C. Perlin noise pixel shaders. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, 87–94, 2001.

[16] Rost. R. *OpenGL Shading Language*, 2nd edn. AddisonWesley Professional, 2006.

[17] Spjut, J. B.; Kensler, A. E.; Brunvand, E. L. Hardware-accelerated gradient noise for graphics. In: Proceedings of the 19th ACM Great Lakes Symposium on VLSI, 457–462, 2009.

[18] Perlin, K.; Neyret, F. Flow noise. *SIGGRAPH Technical Sketches and Applications*, 187, 2001.

[19] Bridson, R.; Houriham, J.; Nordenstam, M. Curl-noise for procedural fluid flow. In: Proceedings of the ACM SIGGRAPH 2007 Papers, 46–es, 2007.

[20] Thorimbert, Y.; Chopard, B. Polynomial methods for fast procedural terrain generation. *arXiv preprint* arXiv:1610.03525, 2016.

[21] Wyvill, G.; Novins, K. Filtered noise and the fourth dimension. In: Proceedings of the ACM SIGGRAPH 99 Conference Abstracts and Applications, 242, 1999.

[22] Olano, M., Akeley, K., Hart, J. C., Heidrich, W., McCool, M., Mitchell, J. L.; Rost, R. Real-time shading. In: Proceedings of the SIGGRAPH 2004 Course Notes, 1–es, 2004.

[23] Worley, S. A cellular texture basis function. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, 291–294, 1996.

[24] Lewis, J. P. Texture synthesis for digital painting. In: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, 245–252, 1984.

[25] Lewis, J. P. Methods for stochastic spectral synthesis. In: Proceedings on Graphics Interface '86/Vision Interface '86, 173–179, 1986.

[26] Lewis, J. P. Algorithms for solid noise synthesis. *ACM SIGGRAPH Computer Graphics* Vol. 23, No. 3, 263–270, 1989.

[27] Van Wijk, J. J. Spot noise texture synthesis for data visualization. *ACM SIGGRAPH Computer Graphics* Vol. 25, No. 4, 309–318, 1991.

[28] Frisvad, J. R.; Wyvill, G. Fast high-quality noise. In: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia, 243–248, 2007.

[29] Gilet, G.; Sauvage, B.; Vanhoey, K.; Dischler, J. M.; Ghazanfarpour, D. Local random-phase noise for procedural texturing. *ACM Transactions on Graphics* Vol. 33, No. 6, Article No. 195, 2014.

[30] Galerne, B.; Leclaire, A.; Moisan, L. Texton noise. *Computer Graphics Forum* Vol. 36, No. 8, 205–218, 2017.

[31] Lagae, A.; Lefebvre, S.; Drettakis, G.; Dutré, P. Procedural noise using sparse Gabor convolution. *ACM Transactions on Graphics* Vol. 28, No. 3, Article No. 54, 2009.

[32] Lagae, A.; Lefebvre, S.; Dutre, P. Improving gabor noise. *IEEE Transactions on Visualization and Computer Graphics* Vol. 17, No. 8, 1096–1107, 2011.

[33] Lagae, A.; Drettakis, G. Filtering solid Gabor noise. *ACM Transactions on Graphics* Vol. 30, No. 4, Article No. 51, 2011.

[34] Galerne, B.; Lagae, A.; Lefebvre, S.; Drettakis, G. Gabor noise by example. *ACM Transactions on Graphics* Vol. 31, No. 4, Article No. 73, 2012.

[35] Bénard, P.; Lagae, A.; Vangorp, P.; Lefebvre, S.; Drettakis, G.; Thollot, J. NPR Gabor noise for coherent stylization. In: Proceedings of the ACM SIGGRAPH 2010 Talks, Article No. 40, 2010.

[36] Tricard, T.; Efremov, S.; Zanni, C.; Neyret, F.; Martínez, J.; Lefebvre, S. Procedural phasor noise. *ACM Transactions on Graphics* Vol. 38, No. 4, Article No. 57, 2019.

[37] Cook, R. L.; DeRose, T. Wavelet noise. *ACM Transactions on Graphics* Vol. 24, No. 3, 803–811, 2005.

[38] Goldberg, A.; Zwicker, M.; Durand, F. Anisotropic noise. In: Proceedings of the ACM SIGGRAPH 2008 Papers, Article No. 54, 2008.

[39] Fournier, A.; Fussell, D.; Carpenter, L. Computer rendering of stochastic models. *Communications of the ACM* Vol. 25, No. 6, 371–384, 1982.

[40] Lewis, J. P. Generalized stochastic subdivision. *ACM Transactions on Graphics* Vol. 6, No. 3, 167–190, 1987.

[41] Kwatra, V.; Essa, I.; Bobick, A.; Kwatra, N. Texture optimization for example-based synthesis. *ACM Transactions on Graphics* Vol. 24, No. 3, 795–802, 2005.

[42] Sendik, O.; Cohen-Or, D. Deep correlations for texture synthesis. *ACM Transactions on Graphics* Vol. 36, No. 5, Article No. 161, 2017.

[43] Zhou, Y.; Zhu, Z.; Bai, X.; Lischinski, D.; Cohen-Or, D.; Huang, H. Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics* Vol. 37, No. 4, Article No. 49, 2018.

[44] Liu, J.; Gan, Y. H.; Dong, J. Y.; Qi, L.; Sun, X.; Jian, M. W.; Wang, L.; Yu, H. Perception-driven procedural texture generation from examples. *Neurocomputing* Vol. 291, 21–34, 2018.

[45] Guehl, P.; Allègre, R.; Dischler, J. M.; Benes, B.; Galin, E. Semi-procedural textures using point process texture basis functions. *Computer Graphics Forum* Vol. 39, No. 4, 159–171, 2020.

[46] Zsolnai-Fehér, K.; Wonka, P.; Wimmer, M. Gaussian material synthesis. *ACM Transactions on Graphics* Vol. 37, No. 4, Article No. 76, 2018.

[47] Kuipers, L.; Niederreiter, H. *Uniform Distribution of Sequences*. John Wiley & Sons, Inc., 1974.

[48] Forbes, T. Reviewed Work: *Prime Numbers: A Computational Perspective* by Richard Crandall, Carl Pomerance. *The Mathematical Gazette*, Vol. 86, No. 507, 552–554, 2002.

[49] Weisstein, E. W. Sphere point picking. Available at https://mathworld.wolfram.com/SpherePointPicking.html.

[50] Szudzik, M. P. The Rosenberg-strong pairing function. *arXiv preprint* arXiv: 1706.04129, 2017.

[51] Fisher, R. A.; Yates, F. Statistical tables for biological, agricultural, and medical research. *Journal of the Institute of Actuaries* Vol. 90, No. 3, 370–370, 1963.

Sheldon Taylor was awarded a B.S. degree in computer science in 2020 plus a B.Comm. degree in accounting in 2016, both from Saint Mary's University, Halifax, Canada. He is currently a software engineer at Amazon where he works on S3. His research interests include procedural noise generation along with distributed and decentralized computing.

Owen Sharpe holds his B.S. degree in mathematics from Saint Mary's University. He is currently researching procedural noise and medial skeleton construction with the Graphics and Spatial Computing Lab at Saint Mary's (led by Dr. Jiju Peethambaran).

Jiju Peethambaran received his bachelor degree in information technology from the University of Calicut, Kerala, India, his master degree in computer science from the National Institute of Technology, Karnataka, Mangalore, India, and his Ph.D. degree in computational geometry from the Indian Institute of Technology Madras, Chennai, India. He is currently an assistant professor in the Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Canada. His research interests include computer graphics and computer vision, especially point cloud processing and geometric modeling.