Pacific Graphics 2022
N. Umetani, E. Vouga, and C. Wojtan
(Guest Editors)

Volume 41 (2022), Number 7

Color-mapped noise vector fields for generating procedural micro-patterns

C. Grenier, B. Sauvage, J.-M. Dischler and S. Thery

ICube, Université de Strasbourg, CNRS, France

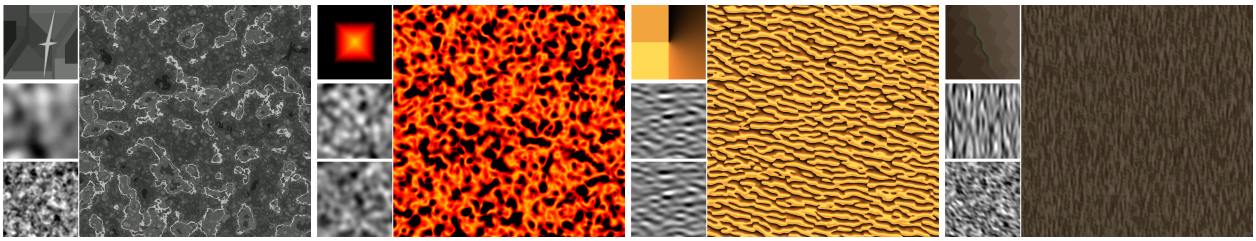


Figure 1: *Bi-variate noise vector fields generalize scalar noise fields and enable to generate procedural micro-patterns. Using two input noises (bottom left) and a bi-dimensional color-map (top left) provides a much larger variety of patterns (large images), including Gaussian patterns, profiled waves, concentric and non-concentric patterns. Discontinuities in the color-map yield structured micro-patterns. Both generation and filtering are real-time.*

Abstract

Stochastic micro-patterns successfully enhance the realism of virtual scenes. Procedural models using noise combined with transfer functions are extremely efficient. However, most patterns produced today employ 1D transfer functions, which assign color, transparency, or other material attributes, based solely on the single scalar quantity of noise. Multi-dimensional transfer functions have received widespread attention in other fields, such as scientific volume rendering. But their potential has not yet been well explored for modeling micro-patterns in the field of procedural texturing. We propose a new procedural model for stochastic patterns, defined as the composition of a bi-dimensional transfer function (a.k.a. color-map) with a stochastic vector field. Our model is versatile, as it encompasses several existing procedural noises, including Gaussian noise and phasor noise. It also generates a much larger gamut of patterns, including locally structured patterns which are notoriously difficult to reproduce. We leverage the Gaussian assumption and a tiling and blending algorithm to provide real-time generation and filtering. A key contribution is a real-time approximation of the second order statistics over an arbitrary pixel footprint, which enables, in addition, the filtering of procedural normal maps. We exhibit a wide variety of results, including Gaussian patterns, profiled waves, concentric and non-concentric patterns.

CCS Concepts
• ***Computing methodologies*** → ***Rendering; Antialiasing; Texturing;***

1. Introduction

Rendering images that look as realistic as possible is an important goal in Computer Graphics. Textures are an efficient way to improve objects' appearance. Procedural modeling of textures has many advantages compared to the use of classical texture maps (discrete data arrays). By construction, they are generally resolution-independent, not bounded in space and, yet, extremely compact. They allow the generation of almost infinite variants through the use of random number generators. By calling a single procedure on every pixel of an image, it is possible to obtain a large amount of details at a low cost. Procedural texture generation can be based on procedural noises, used as random number generators. They offer many advantages in terms of real-time evaluation and compactness in memory. Last, procedural noises come with real-time filtering methods, as filtering is needed to ensure coherence and avoid flickering during movement and zoom in the rendered scene.

However, several difficulties limit the use of procedural textures, especially for real-time applications. They are generally defined by graphs as introduced by Cook in 1984. These graphs contain input nodes (noises, color maps, ...) and operation nodes (heart of the graph, that compute operations between the input to create the output). Graphs representing rich patterns are often too time-consuming to be evaluated in real-time and most of the operation nodes are difficult/impossible to filter (especially non-linear nodes). Therefore most real-time applications do not use directly procedural textures, but rather "bake" them into classical texture maps, which can be then pre-filtered and rendered using fast texture fetches. Often, only when memory is critical, as for 3D textures such as clouds, a truly procedural expression is kept in the shader. In this case, the graph is enforced to be as simple as possible: only a sum of a few noises is used and eventually combined with a very simple 1D color/transparency transfer function stored as a 1D texture. But in this case, the limited expressiveness of noises combined with simple 1D transfer functions represents an important break in the use of procedural textures in real-time applications. An industrial implementation of these kinds of graphs is used in *Substance Designer*, among others. Our pattern model can be easily implemented as a new atomic node for texture graphs.

In this paper, we propose a novel approach to the use of noise in combination with transfer functions. Our key contribution is to introduce a 2D vector noise combined with a subsequent more complex, two-dimensional transfer function. Our bi-variate approach significantly increases the range of patterns that can be modeled using only noises yet keeping the shader program very simple (thus adapted to real-time applications). Furthermore, it also comes with a known filtering method.

After a review of the literature (section 2), this paper presents the following contributions:

- a new stochastic pattern model, based on the composition of a 2D transfer function by a vector noise (section 3);
- a real-time method to estimate the footprint statistics (section 4), which allows for accurate real-time filtering;
- a practical GPU implementation using WebGL (section 5);
- a series of results (section 6) exhibiting a variety of new procedural patterns, as well as improvements for known techniques (LEAN mapping, phasor noise).

2. Previous work

Procedural noises are popular tools in computer graphics since long [Per85]. They provide generation of texture over an arbitrary large surface and do not require storage. Several families of methods exist [LLC*10]. Sparse convolution techniques are based on the use of kernels convoluted by a random distribution [LLDD09; GSV*14]. Other methods are based on the blending of discrete kernels [GLM17; HN18]. These methods are limited in the range of structured patterns they can produce.

The extreme compactness of noise makes it an ideal tool for volumetric texturing. Many natural phenomena modeling systems use noise to enhance models with high-frequency geometric details, which is often called amplification. Historically, this has been introduced by hypertextures [PH89]. Hypertextures also introduced 1D transfer functions, like the gain and bias functions. Beyond adding color to noise (as previously stated), it turns out to be also a successful way of modifying the appearance of noise, and in the case of hypertextures, the shape of 3D details. Noise-based amplification addresses various phenomena such as cloudscapes [Sch17], running water flows [PDG*19], or moss and soils [GD09]. Compactness is generally the core reason why an explicit procedural model is kept in the renderer, even for applications requiring high performance like games, as in [Sch17].

Transfer functions are used for long to map material properties to scalar fields. Especially scientific volume visualization applies this technique. In this area, it has been early understood that 1D transfer functions have a too limited scope. Early on, higher dimensioned transfer functions have been introduced [KKH02]. It was shown that 2D (or higher) transfer functions significantly improve the segmentation of features. In addition to the scalar value, these functions are indexed by gradient or curvature, or any other multi-variate information. A survey of multidimensional transfer functions in scientific volume visualization is out of our scope but can be found in [LKG*16].

Inspired by scientific volume visualization, our goal is to push forward the use of transfer functions and explore novel possibilities to design fast and compact patterns by using more complex multi-dimensional functions composed with noise vector fields. We are not aware of any other work that explicitly uses higher dimensioned transfer functions for procedural modeling and rendering of micro-patterns, which is our core motivation.

Some previous methods were proposed to generate structured micro-patterns. Local random-phase noise [GSV*14] partially controls the phases in a Gaussian process. The tiling and blending algorithm [HN18] uses histogram transfer functions to reproduce non Gaussian histograms. Phasor noise [TEZ*19] extracts the instantaneous phase of a complex Gabor noise, allowing for direct control of the profile of waves. Until now, phasor noise is not suited to real-time rendering because no filtering technique is available. Our model generates micro-patterns with more variety and control than local random-phase noise and tiling and blending. It can also reproduce and filter the phasor profiled noise in real-time.

Filtering is necessary to avoid aliasing problems such as flickering. It ensures consistency during movement and zoom. Several methods exist for textures [BN11]. Procedural noises come with real-time filtering methods. Noises defined by their power spectrum such as Gabor noise [LLDD09] can be filtered using frequency clamping. Noise algorithms using discrete kernels such as Texton noise [GLM17] or Tiling and blending [HN18] are filtered using MIPmapping [Wil83].
Several methods were also proposed for filtering the composition by a transfer function. Bergner et al. [BMWM06] present a method to estimate the Nyquist frequency of the composition using spectral analysis. Heitz et al. [HNPN13; HNPN14] propose a technique to pre-filter non-linear composition functions stored in a color map.
Filtering methods need statistical information about the noise in the pixel footprint, such as mean and variance. Some procedural noise algorithms [LLDD09] own a closed form of the variance, but only over an infinite extent, not over an arbitrary footprint. Deliot et al. [DH19] approximate the variance as a constant for each level

of detail. However, this erases many local variations and the results are over-smoothed. The noise model presented in this paper can be filtered using Heitz et al. method [HNPN14] with two-dimensional function and we present a real-time estimation of the required statistical information in the pixel footprint. This real-time estimation allows us to obtain a real-time implementation of the LEAN-mapping algorithm for the filtering of normal-maps [OB10].

3. Color-mapped noise

In this section we present our noise model and the filtering method used to make it usable in real-time rendering. This filtering method was proposed by Heitz et al. [HNPN14]. In Section 3.2 we present the formalization of their process to our 2D model.

3.1. Model

Our noise is defined as a composition

$$S(\mathbf{u}) = H \circ \mathbf{N}(\mathbf{u}) \tag{1}$$

where

$$\mathbf{N}(\mathbf{u}) = \begin{bmatrix} N_x(\mathbf{u}) \\ N_y(\mathbf{u}) \end{bmatrix} \tag{2}$$

associates a noise vector field $\mathbf{N}$ with any position $\mathbf{u} \in \mathbb{R}^2$. The transfer function H is defined on an interval of $\mathbb{R}^2$. It maps a 2D vector onto an intensity in $\mathbb{R}^c$, where c is the number of channels. In this paper we show examples of greyscale ($c = 1$) and color ($c = 3$) patterns. Figure 2 illustrates the model in the context of texture synthesis. $\mathbf{u}$ are the texture coordinates, and H is encoded as a look-up table, a.k.a. color map.

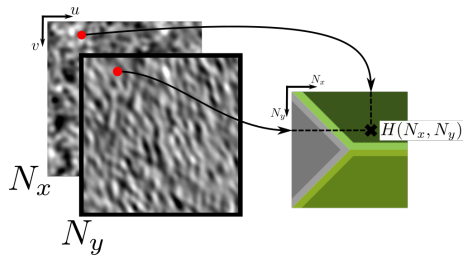


Figure 2: *The noise vector field* $\mathbf{N}$ *(left) is evaluated at position* $\mathbf{u}$ *and the intensities are used as coordinates in the color map* H *(right).*

This model encompasses several known procedural noises. A Gaussian noise G can be reproduced trivially by setting $N_x = G$ and $H(x,y) = x$ (Figure 3a). Note that the frequently used ridged noise is also trivially obtained by setting $N_x = G$ and $H(x,y) = abs(x)$. The phasor noise [TEZ*19] is reproduced by setting $\mathbf{N}$ equal to the phasor field and $H = P \circ antan2(x,y)$ with P a periodic profile function (Figure 3b). Classical color mapped textures are reproduced by fixing a 1D color map along the x-axis in H (Figure 3c). These results are marginal applications of our model, which can address many new patterns by leveraging the richness of 2D transfer functions (Figure 3d-3f). Thus, adding a second dimension to the color-map provides a much wider range of possible layouts for the colors. For example, uni-dimensional color-maps are restricted to concentric patterns (Figure 3c). It is not possible to draw three distinct areas, all in contact with each other that produce mixed blots, as in Figure 3d.

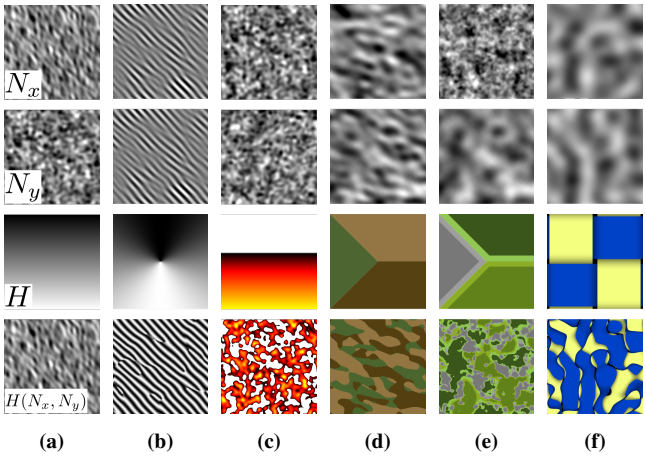


Figure 3: *Our model encompasses several procedural stochastic patterns, including Gaussian noise (3a), phasor noise (3b), and color mapped textures (3c). It can also generate new patterns by leveraging the 2D transfer function (3d-3f).*

3.2. Filtering

In real-time applications, procedural patterns are generated and filtered on-demand at rendering time. To this end, it is mandatory to have a real-time down-sampling process. In this section, we present such a process for our patterns generator. The goal is to compute the average

$$\overline{S}(\mathcal{P}) = \int H \circ \mathbf{N}(\mathbf{u}) \omega_{\mathcal{P}}(\mathbf{u}) \mathrm{d}\mathbf{u} \tag{3}$$

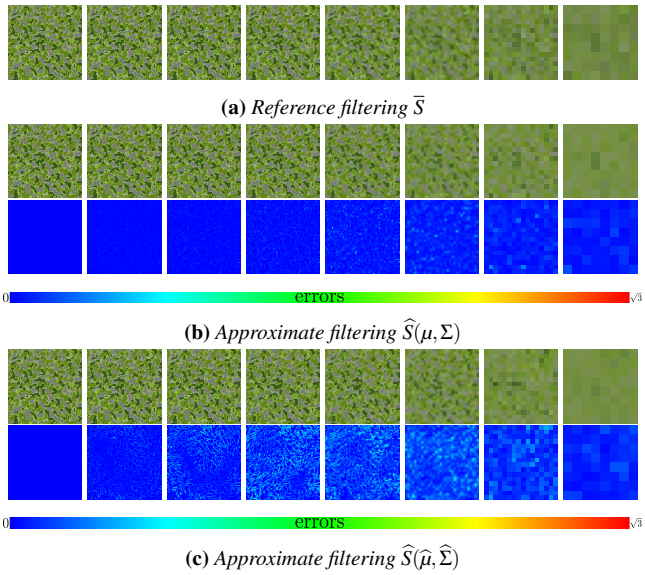
over an arbitrary pixel footprint $\mathcal{P}$, which is represented by a non-negative and compactly supported function $\omega_{\mathcal{P}}$ that integrates to 1 –usually a box or a clamped Gaussian. The challenge is to compute Equation (3) in constant time. This is difficult because H is non-linear (it cannot move out of the integral) and $\mathbf{N}$ is procedural ($H \circ \mathbf{N}$ cannot be stored and pre-filtered). To achieve this, we apply the filtering method developed by Heitz et al. [HNPN14]. They have shown that Equation (3) is equal to the combination of the colors in the color map H weighted by the filter and their presence in $\mathbf{N}$ over the pixel footprint $\mathcal{P}$. This way, Equation (3) is equal to

$$\overline{S}(\mathcal{P}) = \int H(\mathbf{n}) \mathcal{D}_{\mathbf{N},\mathcal{P}}(\mathbf{n}) \mathrm{d}\mathbf{n}, \tag{4}$$

where the weighting function $\mathcal{D}_{\mathbf{N},\mathcal{P}}$ is the probability distribution of $\mathbf{N}$ over $\mathcal{P}$.

To determine $\mathcal{D}_{\mathbf{N},\mathcal{P}}$, Heitz et al. hypothesized that the input noise $\mathbf{N}$ is gaussian. Thus, they approximate the distribution over $\mathcal{P}$ with a gaussian distribution $\mathcal{G}_{\mu_{\mathcal{P}},\Sigma_{\mathcal{P}}}$, where μ_P and $\Sigma_{\mathcal{P}}$ are the mean and covariance matrix of $\mathbf{N}$ over $\mathcal{P}$. This is only an approximation, crude for small footprints, good for large ones. Thus Equation (4) is approximated by a convolution

$$\overline{S}(\mathcal{P}) \approx \left[H * \mathcal{G}_{\mathbf{0},\Sigma_{\mathcal{P}}}\right](\mu_{\mathcal{P}}) \stackrel{\text{def}}{=} \widehat{S}(\mu_{\mathcal{P}}, \Sigma_{\mathcal{P}}). \tag{5}$$



(a) *Reference filtering* $\overline{S}$

(b) *Approximate filtering* $\widehat{S}(\mu,\Sigma)$

(c) *Approximate filtering* $\widehat{S}(\widehat{\mu},\widehat{\Sigma})$

Figure 4: *Comparison of the filtering approximations* $\widehat{S}$ *with the reference* $\overline{S}$*. From left to right: increasing footprint from* 1 *up to* 128^2 *(image size* 1024^2*). The color-coded error maps use the* 2*-norm* $||\widehat{S}-\overline{S}||_2$*. The reference (4a) is computed with Equation* (3)*. The approximation (4b) is defined by Equation* (5) *and exact* μ *and* Σ*. The approximation (4c) is defined by Equations* (5)*,* (8)*, and* (12)*.*

Figure 4b shows that the filtering approximation $\widehat{S}$ closely matches the reference $\overline{S}$. For small footprints (left) the error is very low because S is locally almost linear. As the footprint size approaches the noise wavelength (fourth and fifth column), the error grows because the Gaussian approximation over $\mathcal{P}$ loses accuracy. For large footprints (right) the error drops because the distribution over $\mathcal{P}$ approaches the theoretical Gaussian distribution.

Computing Equation (5) still requires a real-time evaluation of the convolution. For that, Heitz et al. proposed to pre-compute the convolution for a discrete set of mean and variance values, and store it as

$$\overline{H}(\mu_{\mathcal{P}},\Sigma_{\mathcal{P}}). \tag{6}$$

Our practical implementation of this idea is detailed in section 5, and illustrated in Figure 5 right. Note that there is only two sampling dimensions σ_x and σ_y for the covariance matrix, because the independence of N_x and N_y make the covariance vanish so $\Sigma_{\mathcal{P}} = \begin{bmatrix}\sigma_x & 0\\ 0 & \sigma_y\end{bmatrix}$ is diagonal.

4. Real-time estimators of the footprint statistics

The last brick we need for computing Equation (5) in real-time is an estimate of the mean μ and the covariance matrix Σ for an arbitrary footprint $\mathcal{P}$, which depends on the noise synthesis algorithm. Exact values of μ and σ can be computed, but not in real-time, because the complexity grows linearly with regard to the size of the footprint. For real-time applications, the challenge is to provide an estimation

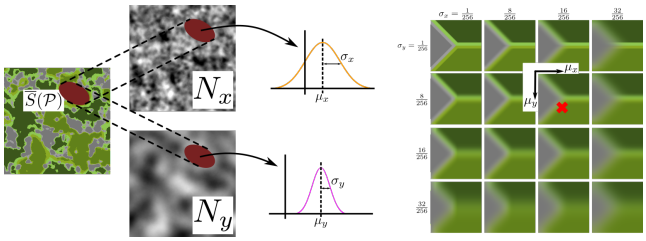


Figure 5: *Estimation of* $\overline{S}$ *over* $\mathcal{P}$ *(left; dark red region) by* $\widehat{S}$*. Using independent noises* N_x *and* N_y *allows to neglect the covariance and to consider only mean and variance of the noises for the filtering. The means and standards deviation (middle;* μ_x*,* μ_y*,* σ_x *and* σ_y*) are used to fetch and interpolate pre-computed values* $\overline{H}$ *(right; red cross).*

in constant complexity whatever the size of the footprint. Most of the algorithms that implement a Gaussian noise come along with an estimate formula for the mean. Algorithms based on band-limited kernels, such as Gabor noise [LLDD09] or LRP-noise [GSV*14], use frequency clamping. Algorithms based on discrete kernels or tiles, such as texton noise [GLM17] or tiling and blending [HN18], use a MIPmap of the discrete example. Conversely, the variance is usually known only for *an infinite extent* (i.e. the theoretical variance of the stochastic process). Only Deliot et al. [DH19] propose to pre-compute a constant variance per level-of-detail, but it does not vary across the texture. To the best of our knowledge, real-time estimation of the variance *over an arbitrary footprint* is still an open problem. Here, we propose a first real-time variance estimator for the tiling and blending algorithm [HN18]. We explain only the main formulas, useful for understanding and implementation. We refer the reader to the supplemental document for a detailed derivation of these formulas.

4.1. Tiling and blending background

We recall here the background about the tiling and blending algorithm [HN18; DH19; Bur19]. This algorithm synthesizes a noise N from a discrete input example E by tiling the infinite plane with overlapping hexagonal tiles trimmed in E. Without loss of generality, it is assumed E has zero mean (otherwise both E and N are shifted). Assuming that E is the realization of a Gaussian process, N is evaluated at any location $\mathbf{u}$ as a weighted average of three overlapping tiles :

$$N(\mathbf{u}) = \sum_{i=1}^{3} w_i(\mathbf{u})E_i(\mathbf{u}) \tag{7}$$

The tiles E_i are trimmed at random locations in E. The weighting functions w_i decrease from 1 at the center of the tile to 0 at the boundary, with $\sum_i w_i^2(\mathbf{u}) = 1$ for all $\mathbf{u}$.

4.2. Mean estimation

To filter N in real-time, a MIPmap of E is pre-computed, as suggested in [HN18]. We denote $\{E_0, E_1, E_2, \ldots\}$ the MIP hierarchy, with $E_0 = E$ the finest level. Then, for a texel at level l, having a

square footprint $\mathbb{P}$ with center $\dot{\mathbb{P}}$ the mean over $\mathbb{P}$ is approximated by

$$\overline{N}(\mathbb{P}) \approx \sum_{i=1}^{3} w_i(\dot{\mathbb{P}}) E_{l,i}(\mathbb{P}) \stackrel{\text{def}}{=} \widehat{N}_l(\mathbb{P}) \tag{8}$$

The tile $E_{l,i}$ has the same shape as E_i but is trimmed in E_l. The approximation is due to w_i which is considered constant over the footprint and evaluated at its center. At this stage, we can estimate the mean over a square footprint $\mathbb{P}$.

More advanced filtering, including anisotropic footprints and weighting functions, requires an estimation over an arbitrary footprint $\mathcal{P}$. Let $\mathcal{P} = \bigcup \mathbb{P}$ be the coverage by several texels $\mathbb{P}$ at level l, weighted by $w_{\mathcal{P}}(\mathbb{P}) > 0$ that sum up to 1. The mean is estimated by

$$\widehat{N}_l(\mathcal{P}) = \sum_{\mathbb{P} \in \mathcal{P}} w_{\mathcal{P}}(\mathbb{P}) \widehat{N}_l(\mathbb{P}). \tag{9}$$

Equations (8) and (9) formalize the algorithm of Deliot et al. [DH19]. We now move on to our contribution, which is the estimation of the second order statistics, i.e. the covariance matrix.

4.3. Variance estimation

The purpose of this section is to derive a variance estimator similar to the mean estimator in section 4.2. However, conversely to the mean, the variance is not linear so its estimation is challenging. Specifically, the derivations imply the estimation for mixes of random variables (due to tile blending) and union of samples (due to union of footprints). In the following, we provide the main results. We refer the reader to the supplemental document for a detailed derivation of these formulas.

First, a MIPmap of variances $\{V_0, V_1, V_2, \ldots\}$ is pre-computed. It is defined for any texel footprint $\mathbb{P}$ at level l, by

$$V_l(\mathbb{P}) \stackrel{\text{def}}{=} \frac{1}{\#\mathbb{P}} \sum_{\mathbf{u} \in \mathbb{P}} \left(E(\mathbf{u}) - E_l(\mathbb{P})\right)^2. \tag{10}$$

The variance over a texel

$$\sigma^2(\mathbb{P}) = \frac{1}{\#\mathbb{P}} \sum_{\mathbf{u} \in \mathbb{P}} \left(N(\mathbf{u}) - \overline{N}(\mathbb{P})\right)^2 \tag{11}$$

is then estimated by

$$\widehat{\sigma^2}_l(\mathbb{P}) \stackrel{\text{def}}{=} \sum_{i=1}^{3} w_i^2(\dot{\mathbb{P}}) V_{l,i}(\mathbb{P}). \tag{12}$$

This implies approximating w_i over $\mathbb{P}$ by a constant and neglecting the covariance between tiles. Note how w_i is squared compared to Equation (8).

Let $\mathcal{P}$ be an arbitrary footprint, discretized as $\mathcal{P} = \bigcup \mathbb{P}$ with weights $w_{\mathcal{P}}(\mathbb{P})$. The variance

$$\sigma^2(\mathcal{P}) = \overline{N^2}(\mathcal{P}) - \left(\overline{N}(\mathcal{P})\right)^2 \tag{13}$$

is approximated by

$$\widehat{\sigma^2}_l(\mathcal{P}) \stackrel{\text{def}}{=} \widehat{N^2}_l(\mathcal{P}) - \left(\widehat{N}_l(\mathcal{P})\right)^2, \tag{14}$$

where $\widehat{N}_l(\mathcal{P})$ is defined by Equation (9), and

$$\widehat{N^2}_l(\mathcal{P}) \stackrel{\text{def}}{=} \sum_{\mathbb{P} \in \mathcal{P}} w_{\mathcal{P}}(\mathbb{P}) \left(\widehat{N}_l(\mathbb{P})^2 + \widehat{\sigma^2}_l(\mathbb{P})\right). \tag{15}$$

4.4. Covariance estimation

Let's have a second noise N' synthesis using tiling and blending method from a discrete input example E'. We denote $\{E'_0, E'_1, E'_2, \ldots\}$ the MIP hierarchy, with $E'_0 = E'$ the finest level. The estimation of covariance between N and N' is similar to the variance estimation.

We define a MIPmap of covariance $\{C_0, C_1, C_2, \ldots\}$ as

$$C_l(\mathbb{P}) \stackrel{\text{def}}{=} \frac{1}{\#\mathbb{P}} \sum_{\mathbf{u} \in \mathbb{P}} \left(E(\mathbf{u}) - E_l(\mathbb{P})\right) \left(E'(\mathbf{u}) - E'_l(\mathbb{P})\right). \tag{16}$$

The covariance over a texel is estimated by

$$\widehat{cov}_l(\mathbb{P}) \stackrel{\text{def}}{=} \sum_{i=1}^{3} w_i^2(\dot{\mathbb{P}}) C_{l,i}(\mathbb{P}) \tag{17}$$

The covariance over $\mathcal{P} = \bigcup \mathbb{P}$ is estimated by

$$\widehat{cov}_l(\mathcal{P}) \stackrel{\text{def}}{=} \widehat{NN'}_l(\mathcal{P}) - \widehat{N}_l(\mathcal{P}) \widehat{N'}_l(\mathcal{P}) \tag{18}$$

with

$$\widehat{NN'}_l(\mathcal{P}) \stackrel{\text{def}}{=} \sum_{\mathbb{P} \in \mathcal{P}} w_{\mathcal{P}}(\mathbb{P}) \left(\widehat{N}_l(\mathbb{P}) \widehat{N'}_l(\mathbb{P}) + \widehat{cov}_l(\mathbb{P})\right) \tag{19}$$

5. Implementation and performance

To allow easy experimentation of our method for all readers, we choose to do the implementation using WebGL2.0. (based on the specification of OpenGL ES 3.0). This allows us also to ensure that it could really be implemented on almost any graphics hardware. The tiling and blending part has been inspired by the original code from Deliot et al. [DH19]. The input textures (noises, variances, and color-maps) can be stored on 1-byte fixed precision channel as usage of 32 bits floating-point storage does not improve visual quality. As the variance values are always below 0.05 we multiply it by 16 to optimize the fixed precision of the eight bits channel. All intermediate computations are done and temporarily stored with floating-point precision to avoid approximation. The pre-computations of variances MIPmap and Gaussian filtering of the color-map are done on GPU by multiple passes of drawing in Frame Buffer Objects. A single clipped triangle is drawn to traverse the texture. Results are then copied from floating-point textures into the final 1-byte storage.

In our implementation, the noise values N_x and N_y lie in $[0;1]$. μ_x and μ_y are uniformly sampled with 256 values in $[0;1]$. As the changes are more sensitive for small variance values than large ones, we sample σ_x and σ_y exponentially, at $2^k/256$ for $0 \leq k \leq 5$. It results in 6^2 pre-filtered color maps $\overline{H}(\cdot,\cdot,\sigma_x,\sigma_y)$ of size 256^2, as illustrated in Figure 5 right. The total storage for three 8-bit color channels amounts to $7MB$. Then, for an arbitrary set of parameters, $\widehat{S}$ is fetched in $\overline{H}$.

Rendering is straightforward. The actual bottleneck of the method is memory bandwidth usage. As the noises and their variances are 1-channel textures, they are compacted in one 4-channel

8-bits texture. This does not reduce memory usage but reduces the spreading of data in memory, and allows this part of the algorithm to be equivalent to the implementation of Deliot et al. [DH19]. Theoretically, we do four more texel fetches than in the original method, but in practice, we just use four channels instead of three. The two variances of generated noise textures can be computed with only one matrix vector product. Once we get mean and variance values for both the two input noises, we can just fetch the value in the color-map filtered images. They are stored in a Texture2DArray, which is an array of textures internally managed by OpenGL. We use bilinear interpolation with respect to (μ_x, μ_y), while nearest level selection proved to be sufficient with respect to (σ_x, σ_y). It results in 7 fetches: 3 in the MIP-mapped noises and 4 in the pre-filtered color-maps.

The filtering of variance cannot be done automatically by OpenGL. We have implemented the anisotropic filtering algorithm as explained in the OpenGL specification. This allows us to compute the variance following Equation (14). Results on mean values are visually equivalent to the use of OpenGL filtering. In the case of anisotropic filtering ($\times K$), we must do a maximum of $3K$ fetches in the MIPmap. All tiling and blending computations are also done K times. The access and computation done on the color-map is not impacted.

We benchmarked on three different GPUs: NVidia 2080Ti, AMD Radeon Pro 5500M (Apple MacBook pro) and Intel UHD 630. Figure 6 reports the performances in Giga-texels per second.

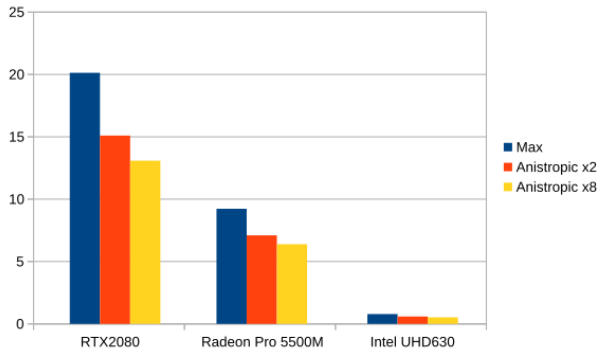


Figure 6: *Performance in Giga-texel per second on different GPUs.*

Our implementation allows us to obtain 3D rendering of patterns (Figure 15). The WebGL2.0 implementation can be found on the project web page.

6. Results

In this section, we first evaluate the performance of our model. Then we investigate the impact of the color map and the noise spectra while exploring a variety of stochastic patterns. We also reproduce the phasor noise and provide unprecedented real-time filtering. We show a valuable combination of our variance estimators with LEAN mapping. Eventually, we discuss limitations and avenues for future works.

6.1. Error measurements

Table 1 shows approximations errors for the first order moment $\widehat{\mu}$ and the standard deviation $\widehat{\sigma}$. Note that we measure standard deviation (not variance) for commensurability with the mean. Note also that we measure mean errors (not RMSE, which would be much lower) because they are known to be more meaningful visually. For each level of filtering (rows), we measure the absolute error and the relative error (in parenthesis). Absolute errors are quite low for both mean and standard deviation. Due to low reference values for σ, the relative error may increase up to 30%. However, once composed with the color map, the results are much less sensitive to the errors in σ than in μ. Figure 4c shows that the visual difference remains low. The errors concentrate on the edges of micro-patterns, which correspond to the discontinuities in the color map that increase the approximation error of Equation (5). A crucial observation is that the approximations are spatially coherent, which implies that the rendering is temporally coherent as the camera moves. Extremely contrasted color-maps provoke some few residual shimmers. As expected, anisotropic filtering improves the image sharpness. This can be observed in the video and in the WebGL demo that accompanies the paper.

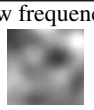
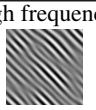
	low frequency 		high frequency 	
	$\widehat{\mu}$	$\widehat{\sigma}$	$\widehat{\mu}$	$\widehat{\sigma}$
$\mathbb{P} = 2 \times 2$	0.0108 (0.0230)	0.0014 (0.3269)	0.0455 (0.0910)	0.0111 (0.3973)
$\mathbb{P} = 4 \times 4$	0.0181 (0.0383)	0.0027 (0.2902)	0.1131 (0.2261)	0.0283 (0.4534)
$\mathbb{P} = 8 \times 8$	0.0330 (0.0698)	0.0056 (0.3085)	0.1018 (0.2035)	0.0394 (0.3345)
$\mathbb{P} = 16 \times 16$	0.0453 (0.0959)	0.0117 (0.3265)	0.0322 (0.0643)	0.0459 (0.2820)
$\mathbb{P} = 32 \times 32$	0.0818 (0.1732)	0.0248 (0.3660)	0.0063 (0.0127)	0.0477 (0.2793)
$\mathbb{P} = 64 \times 64$	0.0851 (0.1801)	0.0355 (0.3194)	0.0017 (0.0033)	0.0441 (0.2493)
$\mathbb{P} = 128 \times 128$	0.0619 (0.1310)	0.0344 (0.2444)	0.0005 (0.0009)	0.0342 (0.1882)

Table 1: *Approximation error for mean $\widehat{\mu}$ and standard deviation $\widehat{\sigma}$ estimators. Each cell contains both the absolute and relative (in parentheses) mean errors. Left / right: a low frequency isotropic noise and a high frequency anisotropic noise are tested. Top to bottom: increasing footprint size. Mean error values of the approximation of first order moment and standard deviation.*

6.2. Choice of the color map

The results are highly influenced by the layout (Figure 7) and the palette (Figure 8) of the color map.

Each region in the color-map corresponds to a specific couple of noise intensities. The borders of the map correspond to the highest and lowest values of N_x and N_y, while the center of the map corresponds to values close to the average intensity. Coloring the borders

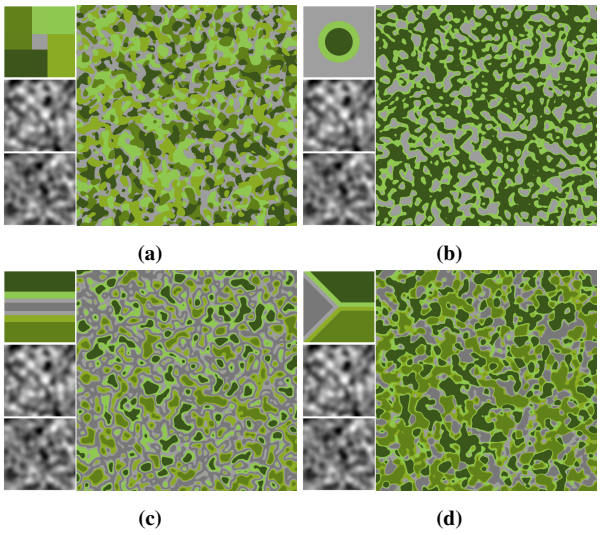


Figure 7: *Exploration of different color map layouts (top left). The same input noises (bottom left) and the same colors are used. Note how the layout modifies arrangements of blots.*

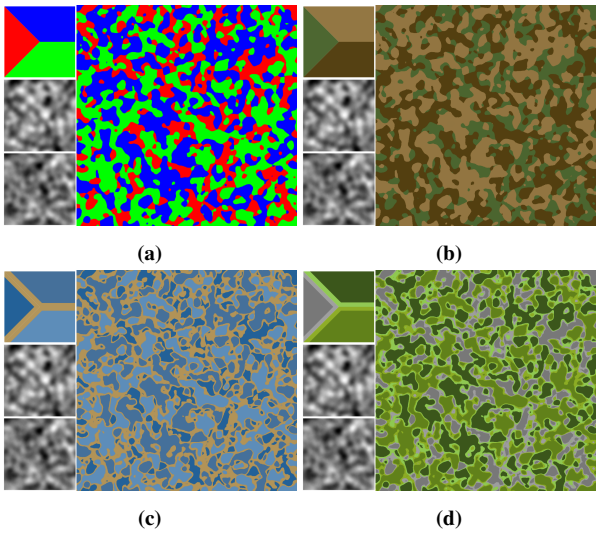


Figure 8: *Exploration of various color palettes for similar layouts and same input noises (left of each sub-figure).*

of the color-map (Figure 7a) produces scattered blots that can overlap. With concentric regions (Figure 7b), the central color (dark green) draws a kind of a network partially connected depending on the size of the central region. The ring-shaped region (light green) wrapping the central region, creates an edging around the previous patterns. Drawing a completely horizontal (or vertical) (Figure 7c) color-map will reproduce a uni-dimensional color-map along x-axis (respectively y). It brings out completely concentric patterns due to the utilization of only one of the two input noises and the fact that one colored band can only be in contact with a maximum of two other colors. On the contrary, with oblique boundaries (Figure 7d) all the blots seem intertwined in the same layer.

In Figure 8 we explore different color palettes with the same layout. The generated patterns have the same shape but evoke different textures. Three weld colors (Figure 8a) generate adjacent blots. The green-brown palette (Figure 8b) evokes camouflage patterns. Adding a uni-color band in the midst of the previous regions (Figure 8c) creates thin strands between blots, which evoke marble-like patterns. Bordering the three regions with different colors (Figure 8d) colors the edging of the blots.

6.3. Choice of the noise spectrum

The spectral characteristics of the noise field **N** impact the shape of the elements in the generated pattern. In Figure 9 we explore combinations of low/high frequencies and isotropic/anisotropic spectra. Two low-frequency isotropic noises (Figure 9a) generate isotropic patterns of the same frequency. Combining low and high frequencies (Figure 9b) creates detailed sub-patterns. Mixing isotropy and anisotropy (Figure 9c) superposes stripes. Mixing two orientations (Figure 9d) interleaves stripes like in weaved materials.

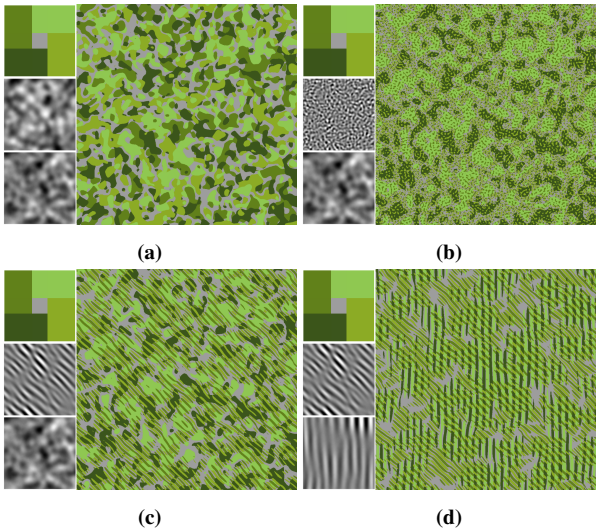
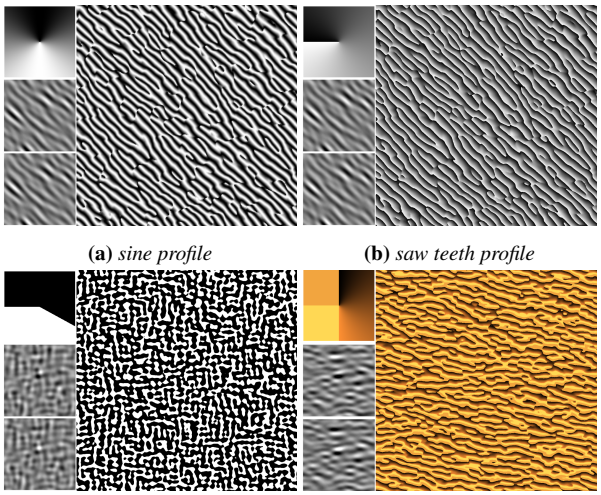


Figure 9: *Exploration of different noise spectral contents. On the left of each sub-figure, the color-map used and parts of the two input noises.*

6.4. Filtering of the phasor noise

Our model can also reproduce the phasor noise [TEZ*19], as shown in Figure 10. Recall that the phasor noise is defined as a periodic profile P composed with the argument of a phasor field. We tune our model $S = P \circ \arctan 2(N_x, N_y)$ where the transfer function $H = P \circ \arctan 2$ and **N** equals the phasor field. As a consequence, the phasor noise can be filtered in real-time with our model (Figure 11). Phasor noise is thus made suitable for real-time rendering, which is without precedent.

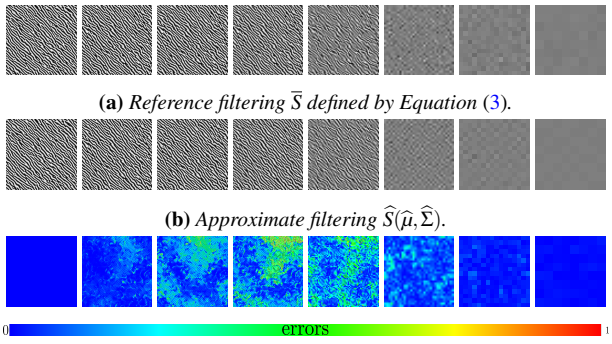
Our model allows to reproduce the phasor noise with different profiles, for example using two bi-lobe noises as input and a sine



(a) *sine profile* **(b)** *saw teeth profile*

(c) *square profile* **(d)** *saw teeth profile with colors*

Figure 10: *Reproduction of the phasor noise as a special case of our model, with different input noises and profiles. Top left is the color-map and bottom left a crop of the two input noises.*



(a) *Reference filtering* $\overline{S}$ *defined by Equation* (3).

(b) *Approximate filtering* $\widehat{S}(\widehat{\mu},\widehat{\Sigma})$.

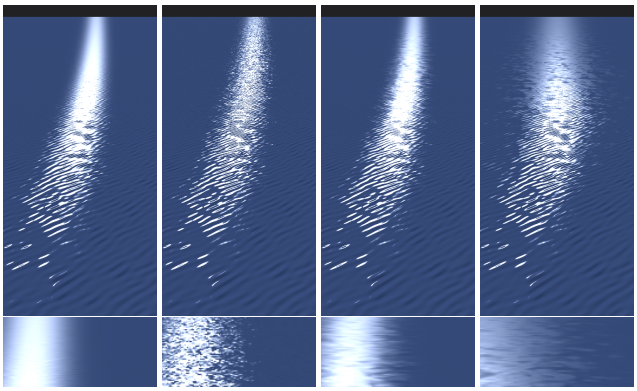
(c) *Approximation error* $||\widehat{S}(\widehat{\mu},\widehat{\Sigma}) - \overline{S}||_2$

Figure 11: *Comparison of the reference filtering* $\overline{S}$ *(11a) with the approximation* $\widehat{S}$ *(11b) using approximation of* μ *and* Σ*, for the procedural phasor noise with sinus profile. The color-coded error maps (11c) use the* l^2*-norm.*

(Figure 10a) or a saw teeth function (Figure 10b). Figure 10c shows the result using the sum of two bi-lobe noises as input and a square profile. Finally, we can use three-channel color-maps (Figure 10d), to color some parts of the pattern.

6.5. Procedural normal maps

Our variance and covariance estimators (Section 4.3 and 4.4) allows to filter procedural normal maps defined by Gaussian slope fields in real-time using the LEAN mapping algorithm [OB10]. Recall LEAN mapping consists in translating explicit macro-normals (foreground Figure 12) to micro-normals statistics (background Figure 12). These statistics (mean and covariance matrix of the slopes) define an anisotropic roughness when zooming out. Until now, the covariance matrix was approximated at best by a constant



(a) *[DH19]* **(b)** *reference* **(c)** *ours* **(d)** *ours w/o cov.*

Figure 12: *Accurate real-time filtering of procedural Gaussian normal-maps using LEAN mapping. On the top row the global view of the specular lobe and on the bottom row a zoom on the mid-distance. The previous estimator, from Deliot et al. [DH19], consider the variance as constant by level of detail (12a). Our variance and covariance estimators allow to preserve mid-distance details (12c). If we neglect the covariance (considering the input noises as independent), the shape of the lobe is modified (12d).*

for each level of detail [DH19]. As shown in Figure 12a, the details are lost and it results in a smooth specular lobe. We apply our estimators of Section 4 directly to the Gaussian slope field (N_x, N_y) computed by tiling and blending. As shown in Figure 12c it reproduces mid-distance details. Conversely to the color-mapped model, the input noises are not independent, so neglecting the covariance is not possible as it would modify the shape of the lobe (Figure 12d). Note that there is no transfer function H for this application, so our statistics estimators are used directly in the LEAN mapping. To the best of our knowledge, we reach unprecedented accuracy for filtering Gaussian normal maps that are generated procedurally in real-time.

7. Discussion

Covariance between input noises. In Section 3.2 we formalized the filtering method using the covariance matrix Σ. Two cases can be distinguished. If the input noises are independent, we can neglect the covariance and use only the variance of the noises independently. Visually, it is characterized by an axis-aligned bi-dimensional distribution $\mathcal{D}_{\mathbf{N},\mathcal{P}}$, which can be separated into two marginal distributions. Our color patterns fall into this case, because N_x and N_y are independent. Conversely, for the case of normal map filtering using LEAN mapping, N_x and N_y are the slopes of a single Gaussian height field. So they are not independent and require the additional covariance estimator mentioned in Section 6.5. Using correlated noises as input for color patterns is an interesting idea for future work as it possibly could further increase the range of patterns.

Anisotropy. We draw the attention of the reader to three different types of anisotropy involved in this work, which are mostly independent. First, the pre-filtering of H in Section 3.2 is anisotropic be-

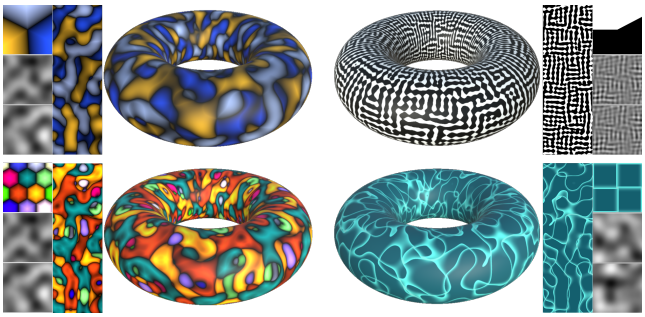


Figure 13: *3D rendering of procedural patterns implemented in Adobe Substance Designer.*

cause, even when N_x and N_y are independent they may have different variances σ_x and σ_y. Second, the noises may be isotropic or not, depending on their spectral content (see e.g. Figure 9). Third, in Section 4.2 anisotropic filtering refers to anisotropic footprints $\mathcal{P}$.

Pattern control. We have shown a variety of patterns that our model can generate. However the creation process is yet empirical (the color palette, the layout of the color-map, and the spectral content of the noises). Scientific volume visualization already pointed out some issues that would require further research. One is the design of 2D transfer functions using specific interfaces or automatic mechanisms. For example, specific inverse procedural modeling techniques could possibly help users create more easily color patterns using our approach (so-called "by-example synthesis"). This is a promising option, but instantiating the model from an input seems to be a challenge.

Artistic control over μ and σ. The filtering method needs the knowledge of first and second-order statistics (mean, variance and covariance) over the pixel footprint. To obtain a realistic rendering with spatial and temporal consistency, it is necessary to estimate values close to the exact values. Thus, we proposed real-time estimators for the second-order statistics. An artistic control over μ and σ might be an interesting idea for future work. However, it would be not trivial to design coherent values, as to enforce spatial and temporal consistency.

Synthesis algorithm. Our model is independent of the algorithm chosen to generate N_x and N_y. For offline synthesis, any algorithm for Gaussian processes works. For real-time synthesis, however, we solved the estimation of variance only for one of them (tiling and blending, Section 4). Designing real-time variance estimators for other algorithms (e.g. Gabor noise) is an open question.

Procedural texture graphs. Our model is fully compatible with industrial implementations of texture graphs. Figure 13 shows a 3D rendering of a procedural pattern implemented in *Adobe Substance Designer*. As shown in Figure 14, it is possible to cascade several operation nodes, including our bi-dimensional transfer functions to obtain more complex patterns. However, it may cause non-Gaussian intermediate noises, that our model is not yet able to filter in real-time.

8. Conclusions

In this paper, we presented a new stochastic micro-pattern model. It is defined as a composition of a noise vector field with a bi-dimensional transfer function. This model allows us to reproduce several known patterns, including procedural phasor noise, or well-known colored noises obtained using a uni-dimensional color-map. By taking advantage of the two dimensions of the color-maps, we can generate a wider range of patterns, such as non-concentric blots and interleaved stripes. Our noise model comes with a filtering technique using Heitz et al. method [HNPN14]. Finally, we presented estimators of the footprint statistics in real-time, allowing for procedural real-time generation and rendering. As a bonus, we are able to filter the phasor noise and to improve LEAN mapping for procedural Gaussian normal maps.

Acknowledgments

This work has been funded by the project ReProcTex from the Agence Nationale de la Recherche (ANR-19-CE33-0011-01). We thank Romain Fournier for the LEAN mapping implementation.

References

[BMWM06] BERGNER, STEVEN, MOLLER, TORSTEN, WEISKOPF, DANIEL, and MURAKI, DAVID J. "A spectral analysis of function composition and its implications for sampling in direct volume visualization". *IEEE transactions on visualization and computer graphics* 12.5 (2006), 1353–1360 2.

[BN11] BRUNETON, ERIC and NEYRET, FABRICE. "A survey of nonlinear prefiltering methods for efficient and accurate surface shading". *IEEE Transactions on Visualization and Computer Graphics* 18.2 (2011), 242–260 2.

[Bur19] BURLEY, BRENT. "On Histogram-preserving Blending for Randomized Texture Tiling". *Journal of Computer Graphics Techniques* 8.4 (2019) 4.

[DH19] DELIOT, THOMAS and HEITZ, ERIC. "Procedural stochastic textures by tiling and blending". *GPU Zen* 2 (2019) 2, 4–6, 8.

[GD09] GILET, G. and DISCHLER, J.M. "A Framework for Interactive Hypertexture Modelling". *Computer Graphics Forum* 28.8 (2009), 2229–2243. DOI: 10.1111/j.1467-8659.2009.01436.x 2.

[GLM17] GALERNE, B., LECLAIRE, A., and MOISAN, L. "Texton Noise". *Computer Graphics Forum* 36.8 (2017), 205–2018. ISSN: 1467-8659. DOI: 10.1111/cgf.13073 2, 4.

[GSV*14] GILET, GUILLAUME, SAUVAGE, BASILE, VANHOEY, KENNETH, et al. "Local Random-Phase Noise for Procedural Texturing". *ACM Transactions on Graphics* 33.6 (Nov. 2014). (Proceedings of Siggraph Asia'14), 195:1–195:11. DOI: 10.1145/2661229.2661249 2, 4.

[HN18] HEITZ, ERIC and NEYRET, FABRICE. "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 2, 4.

[HNPN13] HEITZ, ERIC, NOWROUZEZAHRAI, DEREK, POULIN, PIERRE, and NEYRET, FABRICE. "Filtering Color Mapped Textures and Surfaces". *I3D'13 - ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. Ed. by SPENCER, STEPHEN N. I3D '13 Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. Orlando, United States: ACM, Mar. 2013, 129–136. DOI: 10.1145/2448196.2448217 2.

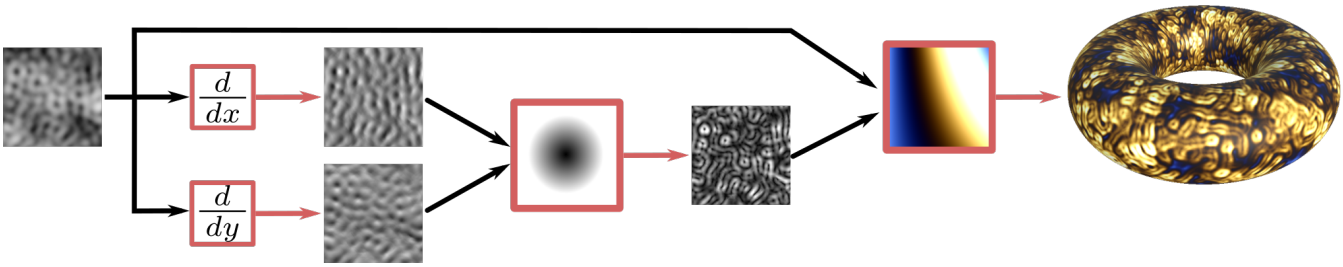


Figure 14: *A more complex texture graph cascading several operation nodes (framed in red): directional derivatives and two bi-dimensional transfer functions. It allows to generate a richer pattern (right) from an input Gaussian noise (left).*

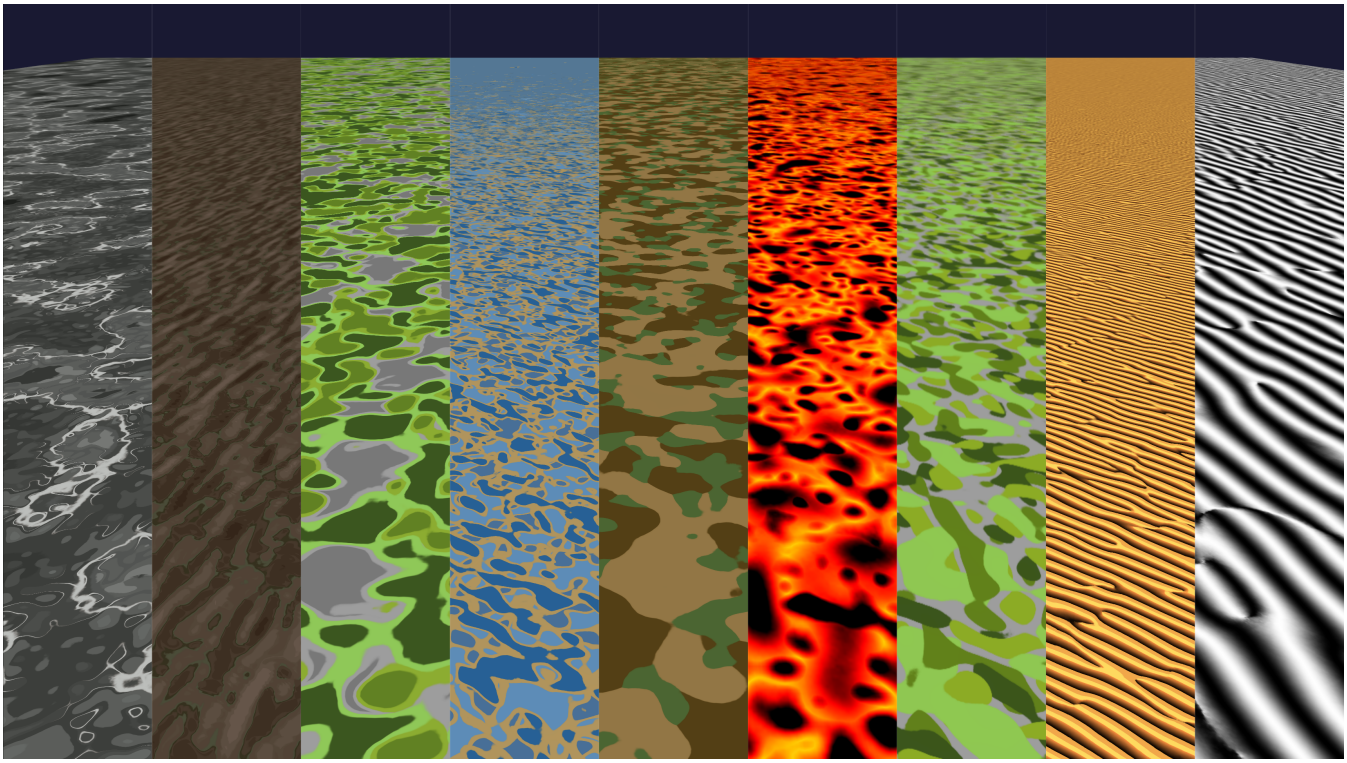


Figure 15: *Real-time generation and filtering of procedural patterns.*

[HNPN14] HEITZ, ERIC, NOWROUZEZAHRAI, DEREK, POULIN, PIERRE, and NEYRET, FABRICE. "Filtering Non-Linear Transfer Functions on Surfaces". *IEEE Transactions on Visualization and Computer Graphics* 20.7 (July 2014), 996–1008. DOI: 10.1109/TVCG.2013.102 2, 3, 9.

[KKH02] KNISS, J., KINDLMANN, G., and HANSEN, C. "Multidimensional transfer functions for interactive volume rendering". *IEEE Transactions on Visualization and Computer Graphics* 8.3 (2002), 270–285. DOI: 10.1109/TVCG.2002.1021579 2.

[LKG*16] LJUNG, PATRIC, KRÜGER, JENS, GROLLER, EDUARD, et al. "State of the Art in Transfer Functions for Direct Volume Rendering". *Computer Graphics Forum* 35.3 (2016), 669–691. DOI: https://doi.org/10.1111/cgf.12934 2.

[LLC*10] LAGAE, ARES, LEFEBVRE, SYLVAIN, COOK, ROB, et al. "A survey of procedural noise functions". *Computer Graphics Forum*. Vol. 29. 8. Wiley Online Library. 2010, 2579–2600 2.

[LLDD09] LAGAE, ARES, LEFEBVRE, SYLVAIN, DRETTAKIS, GEORGE, and DUTRÉ, PHILIP. "Procedural noise using sparse Gabor convolution". *ACM Transactions on Graphics (TOG)* 28.3 (2009), 1–10 2, 4.

[OB10] OLANO, MARC and BAKER, DAN. "LEAN Mapping". *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: Association for Computing Machinery, 2010, 181–188. ISBN: 9781605589398. DOI: 10.1145/1730804.1730834 3, 8.

[PDG*19] PEYTAVIE, A., DUPONT, T., GUÉRIN, E., et al. "Procedural Riverscapes". *Computer Graphics Forum* 38.7 (2019), 35–46. DOI: https://doi.org/10.1111/cgf.13814 2.

[Per85] PERLIN, KEN. "An image synthesizer". *ACM Siggraph Computer Graphics* 19.3 (1985), 287–296 2.

[PH89] PERLIN, K. and HOFFERT, E. M. "Hypertexture". *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '89. New York, NY, USA: Association for

Computing Machinery, 1989, 253–262. ISBN: 0897913124. DOI: 10.1145/74333.74359 2.

[Sch17] SCHNEIDER, A. "Nubis: authoring real-time volumetric cloudscapes with the Decima Engine." *SIGGRAPH Advances in Real-Time Rendering in Games Course*. ACM, 2017, 619–620 2.

[TEZ*19] TRICARD, THIBAULT, EFREMOV, SEMYON, ZANNI, CÉDRIC, et al. "Procedural Phasor Noise". *ACM Transactions on Graphics* 38.4 (July 2019), Article No. 57:1–13. DOI: 10.1145/3306346.3322990 2, 3, 7.

[Wil83] WILLIAMS, LANCE. "Pyramidal parametrics". *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*. 1983, 1–11 2.