DOI: 10.1111/cgf.14061
Eurographics Symposium on Rendering 2020
C. Dachsbacher and M. Pharr
(Guest Editors)

Volume 39 (2020), Number 4

Semi-Procedural Textures Using Point Process Texture Basis Functions

P. Guehl[1], R. Allègre[1], J.-M. Dischler[1], B. Benes[2], and E. Galin[3]

[1]ICube, Université de Strasbourg, CNRS, France [2]Purdue University, USA [3]LIRIS, Université de Lyon, CNRS, France

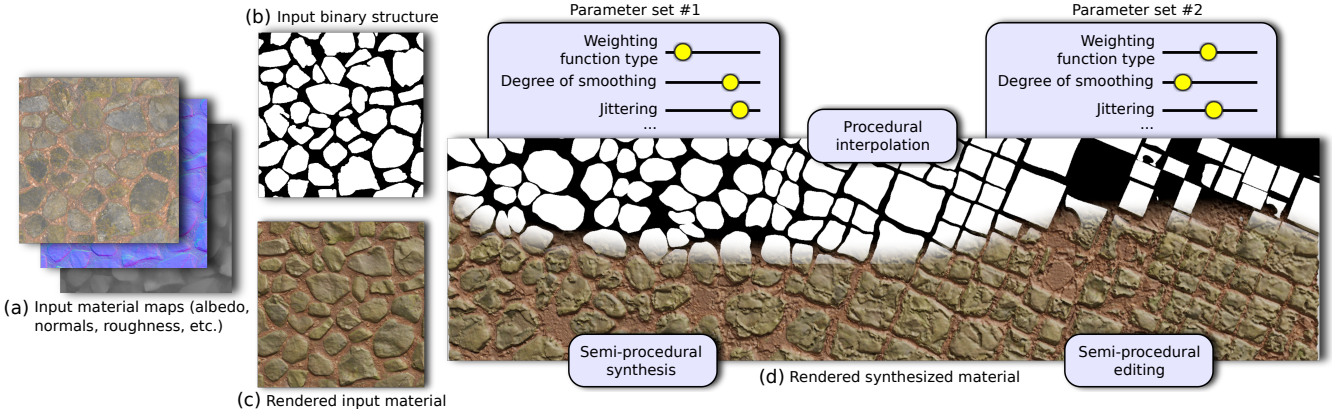

Figure 1: *Input data, a single texture or multiple texture maps (a), and a binary structure (b) are used to generate a semi-procedural output (d). It is a novel texture representation where structure is procedural (d, top) and details are data-driven (d, bottom). Generated textures have procedural properties: infinity, no repetition, self-consistency, and genericity. The structure can be edited by using parameters (only three of them are shown here). Morphing is implicitly obtained by interpolating these parameters. The data-driven details guarantee a good visual match with the exemplar. We call* Semi-procedural synthesis *the synthesis from a structure that matches the input exemplar, and* Semi-procedural editing *the synthesis from a user edited structure. A rendered view of the input material is shown for comparison (c).*

Abstract

We introduce a novel semi-procedural *approach that avoids drawbacks of procedural textures and leverages advantages of data-driven texture synthesis. We split synthesis in two parts: 1) structure synthesis, based on a procedural parametric model and 2) color details synthesis, being data-driven. The procedural model consists of a generic Point Process Texture Basis Function (PPTBF), which extends sparse convolution noises by defining rich convolution kernels. They consist of a window function multiplied with a correlated statistical mixture of Gabor functions, both designed to encapsulate a large span of common spatial stochastic structures, including cells, cracks, grains, scratches, spots, stains, and waves. Parameters can be prescribed automatically by supplying binary structure exemplars. As for noise-based Gaussian textures, the PPTBF is used as stand-alone function, avoiding classification tasks that occur when handling multiple procedural assets. Because the PPTBF is based on a single set of parameters it allows for continuous transitions between different visual structures and an easy control over its visual characteristics. Color is consistently synthesized from the exemplar using a multiscale parallel texture synthesis by numbers, constrained by the PPTBF. The generated textures are parametric, infinite and avoid repetition. The data-driven part is automatic and guarantees strong visual resemblance with inputs.*

1. Introduction

Texture authoring is an important part of creating virtual 3D worlds. The constant increase of complexity requires to set up automatic, scalable, controllable, and efficient texture synthesis techniques. Different approaches have been proposed, with the two prominent approaches being *procedural modeling* and *data-driven*.

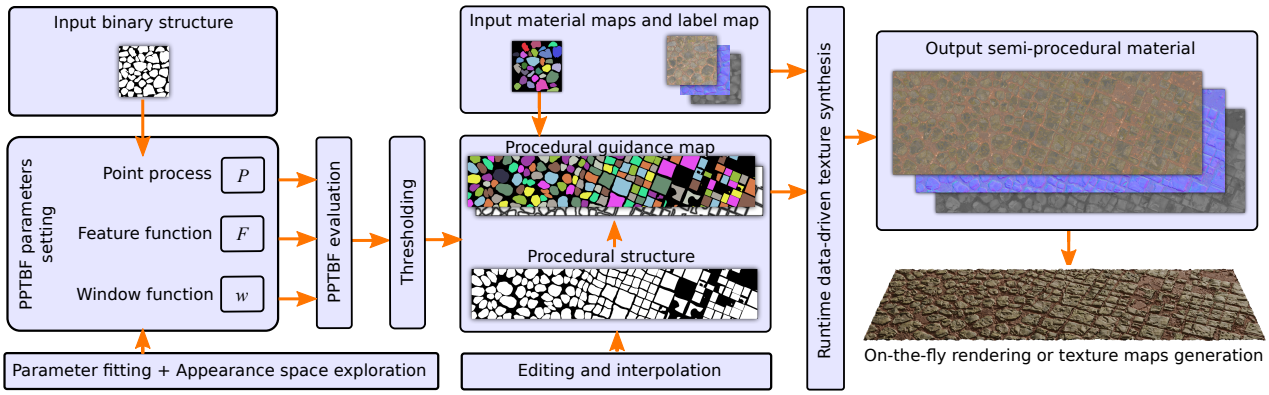

Figure 2: *The pipeline of our method. PPTBF is controlled by parameters that can be estimated from an input, edited by hand, or interactively explored via database browsing and/or latent appearance space inspection. Structure is obtained by thresholding. It can be further edited and spatially interpolated with other procedural structures. Then, data-driven synthesis is applied to generate semi-procedural textures.*

Procedural textures provide data compression and genericity. They are defined by sets of parameters and by generative rules [EMP*02] and many procedural texture assets, based on handcrafted node graphs, are available [Ado19]. However, the creation of procedural textures is a tedious task requiring expert knowledge and trial and error authoring. So called inverse procedural texture modeling methods attempt to encode textures procedurally, but they fail to generate *novel* assets. Modeling all color details to reach a realistic appearance is difficult because prohibitively complex graphs are generally required. Subsequently, in addition to classification issues, a perfect match is almost never found even in very large databases. HU, DORSEY, and RUSHMEIER [HDR19] circumvent this limitation by augmenting textures with style transfer [GEB16].Unfortunately, style transfer cannot recover all visual properties.

Data-driven texture synthesis supplies an input exemplar that is automatically extended, while preserving visual properties. Most methods assume the exemplar is the realization of a statistical process and synthesis consists in creating new realizations by preserving the statistics. However, this inherently raises two issues: (1) the ability of the statistical estimators to match visual criteria and, as for any statistical estimation, (2) the quality of the input. If the input is small, which is the most common practical case, a bias is introduced in estimators, bringing visual artifacts such as repetition, or structural inconsistencies. In addition, control and editing of visual variants is difficult, because, unlike procedural models, there are no parameters controlling specific visual properties.

We introduce a novel complementary approach that merges procedural and data-driven texture synthesis. It uses a generative procedural model for the global structure of the texture, thus guaranteeing structural consistency, genericity and a high degree of user control, in combination with an example-based technique, thus improving visual resemblance.

The *procedural part* is addressed by defining a new generic **Procedural Point Process Texture Basis Function** (PPTBF), inspired by sparse convolution noise [LLDD09; LLD11] and Worley's cellular texture basis function [Wor96]. We avoid the use of existing asset databases, basically designed for color textures, and prefer drawing our approach on noise-by-example [GLLD12]. Using collections of assets has several limitations: it requires 1) to classify texture structures to find out which asset to use, 2) to modify manually all assets to synthesize only structures instead of full colors and materials, 3) to empirically set up specific transitions between all possible combinations of assets to be able to smoothly, and spatially morph structures. Noise-by-example is much simpler as 1) it requires no database at all, 2) structure can be obtained by thresholding and 3) transitions by parameter interpolation. However, noise is limited to unstructured, so called Gaussian textures, and even phase fixing [GSV*14] provides only limited structures. Driven by the structure database we built, we designed our PPTBF to cover a large span of most common stochastic structures, including cells, stripes, stains, dots, grains, etc.

The *data-driven part* provides visual match with input exemplars. This step inherently avoids the aforementioned inverse procedural modeling issues: a perfect procedural match is not required, since details can be transferred from the exemplar. We procedurally generate a guidance function over $\mathbb{R}^2$, based on the PPTBF. The key challenges are to devise a one-to-one mapping that associates resolution independent procedural PPTBF values directly to patches taken from the exemplar. The user controls with a single scalar value how much structure is brought by the PPTBF and how much details by the exemplar.

Our contributions are: 1) a novel generic texture basis function, defined by sparse convolution with a kernel function, that can be easily controlled by intuitive parameters and is GPU compliant, 1) a database of frequently observed 2D structures representing various natural texture structures obtained by segmenting images from other texture databases, and 3) a novel texture generation that decouples structure and details.

Figure 1 illustrates our method on an input binary structure and material maps. Note how the corresponding estimated procedural structure smoothly varies when it is interpolated with another structure. We performed a comparative study with noise by example, state-of-the-art texture synthesis and inverse procedural modeling,

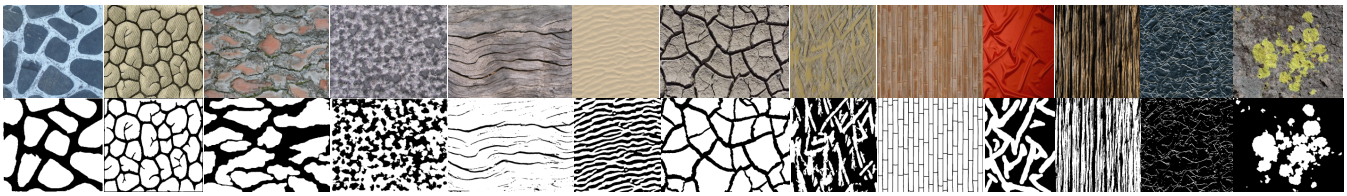

Figure 3: *Many natural textures (top) embed spatial stochastic structures (bottom) such as cells, cracks, grains, scratches, spots, stains or waves. While intricate color patterns are complex to model through procedural texture graphs, structures are simpler to manage. Our PPTBF extends sparse convolution noise and is designed to procedurally produce similar spatial structures. This figure is a subset of our database of 2D visual structures, that we make publicly available.*

which is developed in the supplemental materials. It demonstrates that our method offers a novel complementary alternative to these works by unifying their advantages.

2. Related work

Sparse convolution noises. [Lew89]uses Poisson point distributions to randomly sum weighted clamped Gaussian kernels. Gabor noise [LLDD09] distributes phase-augmented kernels to improve spectral control. Phasor Noise [TEZ*19] replaces the single cosine by a a more complex function to maintain an optimal contrast along wave fronts. For these models, structure is missing, because no correlations among the kernels are considered.

LRP noise [GSV*14] uses a Fourier series to define local patterns. By fixing some phases, the approach is able to represent some weak structures. But strong edges, as for cellular structures, cannot be processed because of blending. *Local spot-noise* [CGG19] uses anisotropic Gaussians. Spatial control is easier but spectral control is lost. *Cellular texture basis function* [Wor96] is a linear combination of n-th closest distances, strictly speaking they are not noises, but noise-similar functions. It creates discontinuities at equidistant locations from the points, *i.e.,* Voronoi cells. However, the shapes of cells are limited because only isotropic distances are considered.

All these models are limited in the range of structures they can represent. Our PPTBF can be seen as a mathematical extension offering more versatility by relying on more advanced convolution kernel functions.

Procedural textures by example. *Noise by example* [GLLD12] unifies advantages of procedural and by-example texture synthesis. However, *noise* by example is not *texture* by example, noise contains no structure. GILET, SAUVAGE, VANHOEY, et al. [GSV*14] fix phases in spectral domain. Fixing a unique set of phases enforces structure to be repeated. Faster noise models have been proposed in [GLM17; HN18]. They directly use discrete example images, that may contain some structures. But blending degrades visual quality when structure is dominant. GUINGO, SAUVAGE, DISCHLER, and CANI [GSDC17] sees textures as a combination of two layers: structure and noise. All of these discrete image-based approaches lack control. Our approach models random structures in a fully procedural way.

Inverse procedural texture modeling seeks to find a procedural model and its parameters using a procedural texture database [BD04; GKHF14; LGD*18; ZWW18]. Recently, HU, DORSEY, and RUSHMEIER [HDR19] perform a supervised classification of a collection of procedural assets by training a CNN for parameter regression. Results strongly depend on the database and the quality of classification. An appropriate asset might not exist. Our approach does neither require a pre-defined database of assets, nor classification as it directly fits the parameters of a unique generic model.

Data-driven texture synthesis has been extensively studied in literature and we refer to surveys [WLKT09; RDDM17]. We may distinguish region matching and parametric methods. *Region matching methods* search best matching regions between the input and output textures, *e.g.,* trying to maximize spatial coherence [LH05; HRRG08], or to minimize a global energy [KEBK05; KNL*15]. If the input exemplar contains structures, these methods require *texton maps*, which are provided as contours extracted from the input enriched with a distance transform. When the input contains subpatterns, the "texture-by-numbers" algorithms [HJO*01; PELS10] can synthesize textures with control over spatial distributions of patterns. This is achieved by providing input and output label maps as extra data to guide synthesis. As an alternative, synthesis can be guided by texture exemplars, e.g. to perform reflectance transport [AWL15]. We adopt a parallel method inspired by [LH06]. The core issue is to device an one-to-one mapping between PPTBF values and local neighborhoods for synthesis.

Parametric methods generate textures that match global image statistics by iteratively refining an initial noise image, so that it progressively matches a set of statistical constraints. These constraints are computed from the responses of multi-scale and multi-orientation filters [PS00]. Texture synthesis using CNNs [GEB15; AAL16; SC17] rely on a statistical description learned from training datasets. Recent approaches generate textures at realtime rates after a long and costly training process [ZZB*18; FAW19]. As for region matching methods, texton maps are required to preserve structures, as well as label maps to control feature distributions. They are not suitable for our approach because they operate on bound domains.

Point sampling and point distributions have been intensively explored. Some methods generate point distributions from exemplars [ÖG12; RÖG17], others control spectral properties [ZHWW12]. They are not suitable for our case 1) because of performance issues and 2) because they use global statistics, such as the Fourier transform, thus operating on bound domains only.

3. Point Process Texture Basis Function

Observation of texture databases, *e.g.,* the well-known Brodatz album [Bro66], reveals that for many textures, their *structure* can be characterized by a binary map, also called texton map (see Fig.3). These structures often embed individual stochastic components like cells, cracks, grains, scratches, spots, stains or waves, characterized by only three components: 1) their distribution (which can be modeled as a *spatial point process*), 2) their local visual shapes (modeled as a *feature function*) and 3) their mutual interaction, *i.e.,* how they blend with each others or stay localized in isolated regions (modeled as a *window function*).

The previous observation has already been considered to develop procedural noise models. Moving further along this line, we introduce a Point Process Texture Basis Function (*PPTBF*), a *generic* procedural texture basis function that increases the amount of stochastic structures that can be reached compared to existing noise models. Our PPTBF encompasses several of them like Gabor noise [LLDD09], Worley's cellular texture basis function [Wor96] and local spot noise [CGG19]. The larger span of reachable texture features allows us to get rid of construction graphs and to use the PPTBF as a stand-alone function to define visual structures by thresholding. Using this structure, we synthesize corresponding infinite textures by transferring colors from input exemplars.

We designed our PPTBF according to database observations: it consists in using 1) arbitrary tessellations to generate clustered point distributions, which are more frequent in nature than the uniform Poisson point distributions used to define sparse convolution noise [LLDD09], 2) anisotropic distance functions instead of isotropic distances, which improves the reachable range of cellular structures compared to [Wor96], and 3) a feature function, being itself a random field described by a statistical mixture of Gabor functions. Unlike local spot noise [CGG19], it permits to unify spatial and spectral control, and allows a direct modeling of features like grains, scratches, stains and flakes. Although our model increases the number of parameters compared to basic sparse convolution noises and Worley's texture basis functions, this number remains sufficiently low to keep automatic parameter estimations tractable.

3.1. PPTBF definition

Noise with arbitrary power spectral density (PSD) can be defined by filtering white noise with a filter kernel function. In spatial domain, filtering is obtained by convolution. Sparse convolution consists in approximating white noise by sets of discrete points (Dirac impulses) with associated random weights. Let us consider an arbitrary distribution P of elements, expressed by a sum of Dirac impulses located at *feature points*. We define our PPTBF by the convolution of P with a kernel being the product between a visual feature f and a window w. P is a sum of Dirac impulses. Convolution can be computed as a discrete sum, limited to k closest regions for computational reasons:

$$PPTBF_k(\mathbf{x}) = \sum_{i|\mathbf{x_i}\in\mathcal{N}_k(\mathbf{x})} f(\mathbf{x}-\mathbf{x_i})\ w(\mathbf{x}-\mathbf{x_i}), \qquad (1)$$

where $\mathcal{N}_k(\mathbf{x}) = \{\mathbf{x_1},...,\mathbf{x_k}\}$ is the set of k closest points $\mathbf{x}_i$ to $\mathbf{x}$, $\mathbf{x}_i$ resulting from a spatial point process. The value of k defines the trade-off between speed and accuracy (we use $k = 18$ in our implementation). The value of w decreases to zero so as to guarantee finite spatial extend. It models the interaction among features. The term f defines the features, *i.e.,* some local patterns.

We now describe our practical choices for $\mathbf{x}_i$, w and f. We present continuous, analytic, computationally efficient functions, whose evaluation can be performed independently for every point of $\mathbb{R}^2$, leading to implicit parallelism and easy implementation on graphics hardware. We exclude heavy computations based on numerical solvers requiring bound domains.

3.1.1. Non-Uniform procedural point distributions

We are interested in generating point processes X whose realizations, denoted as $\mathbf{x}_i$, are locally finite subsets of $\mathbb{R}^2$. They are measurable mappings defined on some probability space inducing a distribution probability P_X. In practice, the measurability of X is the number $\mathcal{N}(\mathbb{S})$ of points $\mathbf{x}_i$ in a sub-part $\mathbb{S}$ of $\mathbb{R}^2$, $\mathcal{N}$ being a random variable. The intensity of a point process is defined as a mean over S. We distinguish four types of point processes, described below.

Poisson point processes define complete spatial randomness, *i.e.,* the intensity is constant over $\mathbb{R}^2$. The points do not interact with each others. Procedural approximations on infinite domains can be computed efficiently using an integer lattice consisting of square cells with constant area 1. By generating κ random independent positions inside each cell, using pseudo-random number generation (PRNG), the intensity over $\mathbb{R}^2$ is constant.

Cox point processes, also called *doubly stochastic Poisson processes*, consider the intensity function of a Poisson point process as a realization of a random field, *i.e.,* the intensity varies and it is itself a noise. Subsequently, points are non-uniformly distributed and form clusters. Various techniques have been proposed to generate different types of Cox processes. The Neyman-Scott process is a straight approach that uses the Poisson point process to define cluster centers. Around these centers, points are distributed according to some PDF. A special case, the Matern process, uniformly draws the points inside disks. Procedural point clusters can be generated over an infinite domain using random tessellations of $\mathbb{R}^2$, where cells consist of rectangles R_i of varying sizes. By drawing κ points inside the R_i, the intensity of points varies w.r.t to the cell sizes.

Gibbs point processes consider interactions among points. An example is the Lennard-Jones process yielding attraction at long scales and repulsion at short scales. In computer graphics, Poisson disk point distributions (blue noise), *e.g.,* distributions where points are never too close nor too far from each other, have given rise to extensive research. The Lloyd algorithm allows to create procedural approximations by iteratively generating centroidal Voronoi tessellations. It allows for fast parallel implementations on unbound domains.

Arbitrary point processes require complex iterative schemes and are not suitable in our case (see Section 2).

By applying various generative rules, an almost endless number of unbound tessellations of $\mathbb{R}^2$ can be imagined in order to generate the most diverse point distributions. Procedural modeling tools implement many different techniques, often depending on a large number of parameters. Their inherent complexity makes automatic

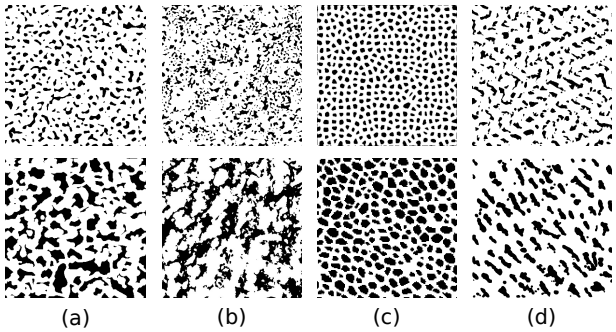

Figure 4: *We procedurally generate following point processes on unbound domains: uniform Poisson distributions (a), clustered distributions (b) and approximations of Poisson-disk (c) point distributions. Jittering amplitude controls randomness: a low value is able to model some near-regular distributions (d). Building on texture databases observations, we experienced these types of distributions to be the most relevant. The first row is procedurally generated with our approach. The second row shows real-world segmented structures depicting similar distributions.*

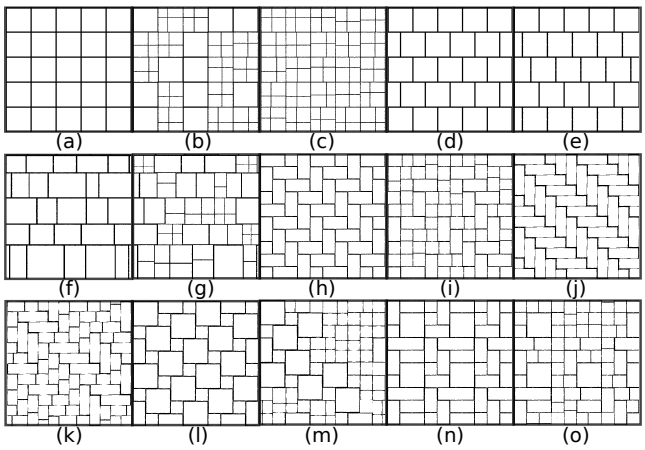

Figure 5: *Practical tilings used to generate point distributions (uniform and clustered). Tilings (b), (c), (g), (i), (k), (m) and (o) use a simple random recursive subdivision (at a single level). Tilings (a) and (b) have additional Lloyd relaxation variants.*

generation from exemplars hardly tractable. We experienced that mainly artificial structures, such as building facades and city layouts require complex schemes, for example grammar-based. Conversely, by looking at texture databases, we observed that uniform, clustered and Poisson-disk point distributions already well cover a large span of natural stochastic patterns. Figure 4 illustrates this. Since these are our target structures, we only implemented a low number of tessellations, in practice 15 (see Fig. 5). A unique parameter for our point distribution is given by a distribution label called ν_T. The tessellations are based on few fixed and simple probabilistic subdivision rules, in addition to a parallel Lloyd algorithm (only applied to tessellations (a) and (b)). Thus, the integer parameter ν_T varies from 1 to 17.

Each of our tessellations defines an infinite set of rectangular cells R_i over $\mathbb{R}^2$. Points (x_i) are randomly drawn inside the R_i using the jittering technique, according to a jitter amplitude *jit*. The latter defines the maximal accepted difference between the point and the center of R_i. When *jit* is low and the tessellation periodic, we also get a near-periodic point distribution, which allows our PPTBF to marginally cover some near-regular structures (see Figure 4(d)). The use of only two parameters makes automatic estimations highly efficient.

3.1.2. Window Function

The function w defines how neighboring features interact. In signal processing, many window functions have been proposed, like the truncated Gaussian, Kaiser-Bessel or tapered cosine windows. We use the PPTBF to define the spatial structure of stochastic patterns. In this context, the window profile does not much influence structural characteristics. Conversely, its spatial footprint (2D shape) turns out to be of primary importance. We distinguish two main families of window shapes:

Non-overlapping windows can be generated from point processes using Voronoi cells or directly from procedural tessellations, since both define a partitioning of $\mathbb{R}^2$ (*i.e.,* without overlaps). They represent visually appealing cellular structures which appear for example in some crack patterns. The distance measurement used to define the Voronoi cells determines their shapes.

Overlapping windows. Overlapping regions can be handled in different ways: added, averaged, stacked or blended. The distance measurement also defines their shape.

We unify these different characteristics by defining a generic window function that is a weighted sum of collections of non-overlapping and overlapping *basis* windows with Gaussian profile:

$$w(\mathbf{x}) = \sum_{j=1}^{n_w} \frac{\omega_j}{\mathrm{N}_j} w_j(\mathbf{x}),\ w_j = \lfloor e^{-\sigma_j D_j(\mathbf{x}-\mathbf{x_i})} \rfloor_{d_j}, \qquad (2)$$

where n_w is the number of *basis* windows, ω_j their weights, and N_j a normalization term. The value D_j is the distance measure accounting for the shape of the window, $\lfloor\ \rfloor_{d_j}$ is the clamping operation, s.t. if $D(\mathbf{x}-\mathbf{x_i}) > d_j$ then $w_j = 0$. d_j thus defines the spatial extent. $\mathbf{x_i}$ is the centroid of the window (Equation 1). It results from the previous procedural point process (see Section 3.1.1).

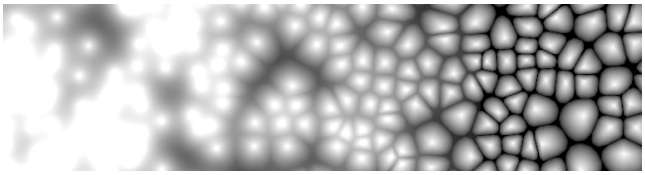

Figure 6: *Our window is defined by a linear combination of basis windows. For practical reasons, we use only two basis windows, but more of them can be used. From left to right, we show the influence of* $\omega \in [0,1]$ *on the appearance of w.*

The functions w_j are understood as basis functions for a window function space, in which a variety of different windows are

expressed by linear combinations. For practical and computational reasons, we implemented only a limited number of basis windows n_w and corresponding distance measures: a single non-overlapping window, w_1 (we call it *cellular window*) and a single overlapping window, *i.e.,* $n_w = 2$ (see Figure 6). The motivation is to keep parameter estimation tractable. We have selected two basis functions, as described next, because they match common texture structures. In order to further reduce parameters, we normalize their weights, $\sum_j \omega_j = 1$ so that, in practice, there remains a single parameter ω.

Non-overlapping window w_1. An isotropic cellular window is defined by computing a distance $D(\mathbf{x}-\mathbf{x_i}) = \|\mathbf{x}-\mathbf{x_i}\|_c$ and by applying a min operator to define corresponding Voronoi cells. Unlike Worley's cellular function, we enforce our $w_1(\mathbf{x})$ to have a constant iso-value exactly on the border of the Voronoi cells, because it allows a better control of visual characteristics when thresholds are applied (recall that we apply thresholds to characterize structure). We thus define $|\mathbf{x}-\mathbf{x}_i|_c = ||\mathbf{x}-\mathbf{x}_i||_p / ||\mathbf{x}-\mathbf{r}_i||_p$, where $\mathbf{r}_i$ is the intersection of the line $(\mathbf{x}\mathbf{x}_i)$ with the Voronoi cell and $||\ ||_p$ represents a p-norm. In practice, $\mathbf{r}_i$ is quickly computed using a binary search. By construction, D takes the value 1 exactly on the Voronoi cell border. Corresponding cellular patterns are generated by setting $d_1 = 1$, which avoids window overlaps.

In spite of using various point distributions and p-norms, we experimentally found that an isotropic distance still has a too limited expressiveness w.r.t. to observed real-world cellular patterns. To better fit observations of texture databases, we increase the range of cell shapes. Many specialized generative rules exist for the most various cellular patterns, like mosaics and stone walls. Their inherent complexity makes them unsuitable for our context. A more appropriate way for extending Voronoi cells shapes, consists in making D anisotropic. The challenge is to keep the amount of parameters as low as possible. Exploiting the rectangular cells R_i of the tessellations (that inherently hold an anisotropy information) offers an good opportunity. We thus introduce a single anisotropy coefficient λ to weight the previous isotropic p-norm against an anisotropic ρ–norm linked to R_i: $\|\mathbf{x}-\mathbf{x_i}\|_c = \lambda\|\mathbf{x}-\mathbf{x_i}\|_c + (1-\lambda)\|\mathbf{x}-\mathbf{x_i}\|_\rho$, with $\|(x,y)\|_\rho = \|(x/dx, y/dy)\|_p$, and where (dx,dy) is the closest distance to the border of R_i. From a visual point of view, λ permits a smooth morphing between the tessellation cell shape and the polygonal Voronoi cell shape.

Natural cellular patterns often depict curved cell shapes, as opposed to straight polygonal shapes (see Figure 3 left. Bézier curves permit the modeling of arbitrary curved shapes, while the amount of parameters remains low. To smooth our window shapes, we put inside each Voronoi cell a polygon composed of n_v vertices used as control points. The vertices are distributed on the Voronoi cell border in an equiangular way, *i.e.,* by fixing a constant angular step $2\pi/n_v$. A coefficient $l_c \in [0,1]$ then allows to control the degree of smoothness (0 means straight lines, 1 means curved). This simple approach for smoothing Voronoi cells was motivated by two practical advantages: 1) it requires only two parameters and 2) computations are highly efficient, because it avoids the costly computation of complete Voronoi cell shapes.

Our w_1 is eventually controlled by following parameters: p-norm, σ_1, λ, l_c and the number n_v of control points. Its normalization term N is always set to 1.

Worley uses n-th closest distances, based on an isotropic p-norm to define collections of F_n functions, that are linearly combined. Our window extends this principle by using anisotropic distance measures. Experimentally, we found that using the first-closest distance for w_1 in association with different tessellations, anisotropy and smoothing already allows to cover well the variety of cellular structures present in our structure database: from round stone walls to crack patterns. We therefore did not consider additional second (or more) closest distance functions.

Overlapping window allows to merge neighboring features. We define our single overlapping basis window w_2 by an isotropic distance $D_2 = ||\mathbf{x}-\mathbf{x}_i||_p$. Its normalization term is given by $N_2 = \varsigma + (1-\varsigma) \times \sum w_2(\mathbf{x})$. ς takes values between 0 and 1, so that the normalization results in adding features when $\varsigma = 1$, and in averaging features when $\varsigma = 0$. For overlapping windows, isotropic distances are sufficient, because the shape of the window does not have a significant visual impact when merging features.

A procedural vector field would be a more general way to control anisotropy (orientations and directional scales), but it would also add more parameters to the model and make the search of closest points less easy and efficient. Conversely, a grid made of rectangles, which also inherently hold anisotropy information according to their length and their width, is simple, permits to avoid adding parameters and guarantees fast processing.

3.1.3. Feature Function

The feature term f models visual features, *i.e.,* local patterns, like stripes, grains, dots or stains. For texture synthesis, this term is generally of primary importance, because it strongly influences the visual characteristics of the textures. Here, the objective is to produce the overall structure of a texture. Precise modeling of high frequency components or specific shapes, such as daisy flowers, maple leaves, etc., is not required.

Different representations are possible: 1) An *image*, as used in [GLM17; HN18] is fast, but not suitable in our case, because "extraction" from an exemplar is not easy (trivially taking a crop is not adequate, as a crop may contain a cellular structure, that we do not want to be part of the feature function) and it is not generic (editable). 2) *Local Fourier series* as in [GSV*14] permit spectral control. Phases can be fixed for given frequencies to model some types of weak structures. However, phase fixing is problematic and freezing a single set of phases generates strong repetitions. 3) *Sums of shifted anisotropic Gaussians* as in [CGG19] avoid repetitions by making parameters random (positions, orientations, etc.). Designing structures is easier in spatial domain than using phase fixing in spectral domain. But spectral control is lost.

We argue that a sum of multiple anisotropic Gabor functions is more versatile. It is inherently able to unify both, spectral and spatial control. We randomly generate Gabor kernel positions, orientations, etc. according to some given PDFs. A strength of our model is that our PDF can include correlations with the previous window function. We also extend Gabor functions to be able to bend the cosine stripes and to make them either thinner or thicker. This has proven useful for modeling common stringy structures like grains and scratches, that would otherwise require complicated phase cor-

relations in spectral domain. We call this extended anisotropic Gabor function, stringed Gabor function, denoted as $\tilde{G}$. Our feature function is defined by :

$$f(\mathbf{x}) = \sum_{j=1}^{J} \omega_j(\mathbf{x})\tilde{G}_j(\mathbf{x}), \quad (3)$$

$$\tilde{G}_j(\mathbf{x}) = A_j\left(e^{-\sigma_{j|i}\left\|\Xi(\mathbf{x}-\mu_{\mathbf{j}|\mathbf{i}})\right\|_f}\right)\left(.5+.5\,cos\left(\phi\left\|\Xi(\mathbf{x}-\mu_{\mathbf{j}|\mathbf{i}})\right\|_f\right)\right)^{\tau}.$$

The index notation $j|i$ means that the corresponding value is correlated to the tessellation cell R_i (see section 3.1.1) and corresponding window centroid $\mathbf{x}_i$ (Equation 1). The term ω_j determines how $\tilde{G}_j$ contributes to the sum on position $\mathbf{x} \in \mathbb{R}^2$. Because of the latter, we call our feature function a *mixture*, rather than a sum. $\sigma_{j|i}$ is randomly drawn proportionally to the size of the corresponding cell R_i. J is bound by $[J_{min}, J_{max}]$, these two parameters controlling the amount of stringed Gabor functions in the mixture. A_j represent weights. $\mu_{\mathbf{j}|\mathbf{i}}$ are positions selected around the window centroids $\mathbf{x}_i$. Ξ is a matrix that allows to orient the cosine stripes according to an angle $\theta_j \in [0, \theta]$. The matrix includes a scaling factor η accounting for anisotropy:

$$\Xi = \begin{pmatrix} \cos(\theta_j) & -\eta\sin(\theta_j) \\ \sin(\theta_j) & \eta\cos(\theta_j) \end{pmatrix} \quad (4)$$

The distance $\|\cdot\|_f = \sqrt{(x')^2\kappa^2 + (y')^2}$ with $\mathbf{x}' = \Xi(\mathbf{x} - \mu_{\mathbf{j}|\mathbf{i}})$ includes a curvature parameter κ that controls curliness of stripes: when $\kappa = 0$ the stripes are rectilinear, when $\kappa = 1$ the stripes form concentric circles around $\mu_{j|i}$. ϕ represents the frequency of the cosine and τ controls the thickness of the stripes.

The previous formulation (Equation 3) is versatile: by defining different mixtures and PDFs for generating the J, $\mu_{j|i}$, θ_j, A_j, etc., it theoretically permits the creation of wide varieties of spatial and spectral structures. However, automatic estimation from exemplars becomes difficult when non-uniform distribution functions and mixture models are not known beforehand. Since we only want to produce rough stochastic structures, we implemented a limited number of mixtures (functions ω_j) with few fixed PDFs. A label parameter $\nu_{\tilde{G}}$ for f defines this (see supplemental materials for implementation details). The PDFs use a correlation coefficient $correl \in [0,1]$, similar to the Matern point process disk radius for $\mu_{j|i}$, such that $\mu_{\mathbf{j}|\mathbf{i}} = correl \times \mathbf{x}_i + (1 - correl) \times \xi_{\mathbf{j}|\mathbf{i}}$, $\xi_{\mathbf{j}|\mathbf{i}}$ being a uniformly drawn random position inside R_i. $\sigma_{j|i}$ is also correlated to R_i, by setting it proportional to the area of R_i. This generates clusters around window centroids. We experienced that correlating positions $\mu_{j|i}$ and variance $\sigma_{j|i}$ with the window function has a significant visual impact. Conversely, correlations have less visual impact for the remaining parameters, which we draw therefore independently. Figure 7 shows examples of obtained feature functions.

4. By-example PPTBF parameters estimation

PPTBF generates continuous scalar fields over $\mathbb{R}^2$. We do not use it to build procedural textures, but only procedural *structures* (by applying a threshold, converting floating points into binary values). A random subset of structures that can be reached with PPTBF is shown in Figure 8. As opposed to structures obtained by segmenting texture images, procedurally generated structures are resolution independent, defined over an infinite domain and editable.

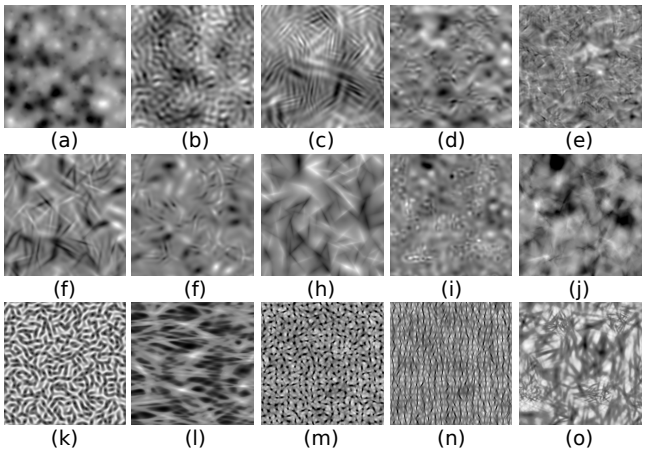

Figure 7: *Comparison of noise patterns generated by sparse convolution with isotropic Gaussians (a) and Gabor functions (b). (c)–(o) illustrate our mixture of stringed Gabor functions, introducing additional parameters: anisotropy η, thickness τ and curliness κ. It allows a more efficient modeling of anisotropic features like grains, stripes and folds.*

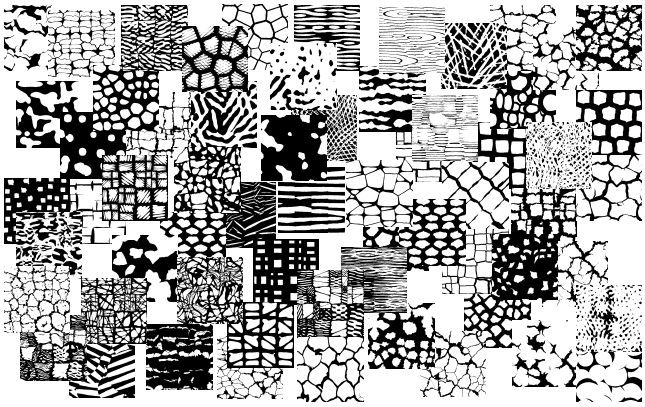

Figure 8: *A subset of structures generated with PPTBF. Structures are defined by thresholding. In spite of a low number of parameters, by construction, PPTBFs cover a large variety of different structural appearances, that are similar to the natural structures shown in Figure 3.*

PPTBF parameters can be edited to tune and control structural characteristics. Here, we only briefly describe automatic example-based parameter recovery, since this is not a contribution. Given a structure exemplar I, encoded as a binary image, we determine the parameters $p_{PPTBF} = \{p_0, p_1, ...p_{k-1}\}$ of the PPTBF so that, when a threshold is applied, we obtain a similar look. We use an iterative scheme to retrieve optimal parameters, based on a similarity measure. A database of pre-computed, sampled and thresholded, PPTBFs is used for initializing a Monte Carlo random search.

To generate the database, we sample the parameter space of the PPTBF and compute a texture feature vector for each image, as presented *e.g.,* in [GKHF14; LGD*18]. We interactively explored the parameter space to devise an empirical sampling scheme, avoiding redundancy as well as large gaps between visual structures. Based on this non-regular sampling scheme, we generated a database of 450k images by applying three different thresholds to the PPTBF fields (20%, 50%, and 80%). In practice, these are sufficient, because input exemplars can be normalized to one of these thresholds by applying morphological erosion and dilatation. The visual exploration and subsequent generation of the database required several days. We note that this is not an unusual approach: most computer graphics and vision techniques that apply deep learning, likewise require a pre-process based on a "manual" generation of training datasets. This is often the most painstaking task.

We apply geometrical transforms to normalize PPTBF generated images. We considered following transforms: scaling, stretching along the X axis, rotation and stochastic Brownian distortions. The latter is motivated by the observation that many natural structures are resulting from complex physical processes well described by fractal Brownian motion. Such a deformation can be easily simulated by applying spatial distortions using fractal noise characterized by a $1/f^n$ spectrum. Our spatial Brownian distortion depends on three parameters: amplitude, frequency and the exponent of the spectrum. We excluded more complex deformations, for example resulting from surface curvatures and perspective distortions. We also assume the structure exemplar is stationary.

Once the database has been augmented with deformations, we use a descriptor to evaluate similarities by computing distances on feature vectors. Defining novel texture descriptors is out of our scope. Many different descriptors have been proposed during the past decades [LCF*18]: we implemented and tested several common ones, including multiscale LPB, Gabor Binary Patterns, histograms of Gabor filters and CNN-based (VGG19). For each descriptor, we precomputed the corresponding set of feature vectors that are then used to query the database by applying a closest distance using the L^2 norm and the FLANN library [ML09]. The query provides instant initial results. We generate more results by Monte Carlo sampling, that are further iteratively refined using either automatic Metropolis search or a visual selection by the user.

5. Semi-Procedural Textures

5.1. Data-driven details

The PPTBF may be directly converted into textures by using color tables or specific color spaces as in [GLLD12; GSV*14]. The problem is that results look unrealistic because they are too "simple" (rich visual details are missing) and because color processing is inherently problematic. We therefore use a data-driven approach. The PPTBF is used to define only the global structure. The latter is obtained by applying a threshold *thresh*. Using a binary image has two advantages: 1) the same PPTBF scalar field can generate different structures with different topologies depending on *thresh* and 2) it avoids over–constraining data-driven texture synthesis.

We aim to consistently transfer color details from an input exemplar to the structure generated by the PPTBF. It consists in applying constrained example-based texture synthesis (textures by numbers or texture transfer [HJO*01; EF01]). Optimization-based texture synthesis consists in minimizing an energy made of two terms: the standard mean square difference energy and a regularization term, so as to balance local details with the given constraint:

$$\operatorname{argmin}\left\{\sum_{\pi_i}\|I(\pi_i)-S(\pi_i)\|^2+\chi\left\|\bar{I}(\pi_i)-\bar{S}(\pi_i)\right\|^2\right\}, \qquad (5)$$

where I is the exemplar, S the synthesized texture and π_i a set of overlapping patches. χ is a constant that controls the constraint. $\bar{\cdot}$ represents the external information, *i.e.,* the guidance map.

In our semi-procedural context, such an approach raises several issues: 1) by sampling the PPTBF, its inherent infinite procedural nature gets lost. 2) A binary structure may not be sufficient to capture some subtle structural details that a texture image can contain: multiple labels are generally used in label maps, not binary images. The labels allow to classify and identify sub-patterns (Fig. 10). 3) The mixing of two energies with a constant constraint might generate visual inconsistencies, especially at the transition regions between sub-textures. The reason is that PPTBF generate novel, non-repetitive variants that may not be present in exemplars.

Our solution is based on an appropriate operator $\bar{\cdot}$, to be applied to both the input exemplar and our PPTBF. In practice, this operator converts the PPTBF into a guidance map. Our choice is as follows:

- the user may not want to make the synthesized texture exactly match the PPTBF structure *everywhere*. Therefore, we define a range $[thresh-\varepsilon, thresh+\varepsilon]$, *thresh* being the threshold used to define the binary structure, such that χ falls down to zero when the PPTBF has a value within this range. Since *thresh* defines the contours of PPTBF structures, it allows to release the PPTBF constraint around features during synthesis. In an extreme, case χ can be zero everywhere: in this case, only standard data-driven texture synthesis is applied. Figure 9 illustrates an example: ε is progressively increased from left to right.
- We generate a distance field from the binary structure images to guarantee a consistent synthesis on feature borders: it creates smooth transitions on borders. But using a border smoothing may not be sufficient to control synthesis. Label maps are more appropriate, because they allow one to distinguish different types of features. This is obtained by extending our PPTBF, so that it also generates a random label directly at the positions $\mathbf{x}_i$. Figure 2 illustrates an example of guidance map generation from the PPTBF: it shows the smoothed feature borders in the binary structure image overlapped by labels shown as false colors.

For optimization (Equation 5), we use a parallel approach, with a block-based initialization. We also apply padding, *i.e.,* block overlapping, to guarantee continuity over $\mathbb{R}^2$. We start from a coarse resolution. Then we refine using upscaling and correction passes, as in [LH05]. Our correction pass uses a nearest neighbor search algorithm based on a random walk. It applies a L2 norm on texels to compute errors by mean square differences over patches of size 5×5. To address material synthesis, we use multiple maps (albedo, normal, roughness, height, and ambient occlusion). These maps are superimposed so that each texel is composed of a higher dimensioned vector instead of only RGB vectors. Upscaling and correction passes remain unchanged: we apply the L2 norm on 9-D vectors instead of 3-D vectors.

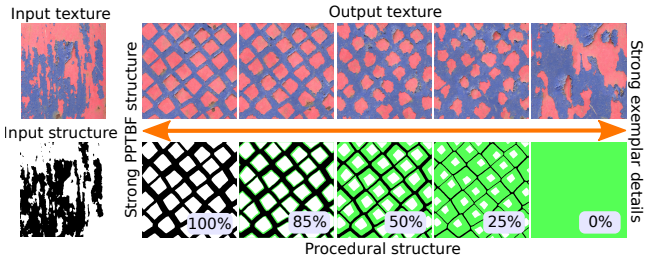

Figure 9: *Semi-Procedural texture synthesis combines the use of PPTBF with texture transfer. The PPTBF is used to constrain synthesis. The user controls the synthesis by defining a function χ. The green regions around features correspond to zero values of χ. It relaxes the PPTBF constraint to preserve the original structural characteristics of the input in these regions.*

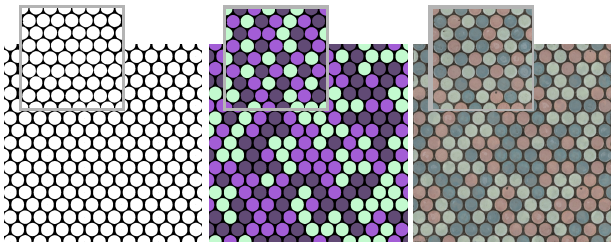

Figure 10: *Textures with non homogeneous features are processed using label maps, e.g. using the unsupervised texture classification method by* LOCKERMAN, SAUVAGE, ALLÈGRE, *et al. [LSA*16]. Our method automatically assigns labels to feature components in a random fashion during semi-procedural texture synthesis (small squares show exemplar).*

5.2. Smooth transitions

A key advantage of using a single PPTBF model is that texture structure morphing is straightforward. In contrast, when using different assets, new specific "transition" nodes have to be build manually, which is an extremely painstaking task because trivial blending generally does not work.

All floating-point parameters are linearly interpolated. There are however, three integer parameters, ν_T, f_t and J that need a special processing. ν_T corresponds to the underlying tiling: it is used to define the $\mathbf{x_i} \in \mathcal{N}_k(\mathbf{x})$. These are used to compute distances D_j. Their number is fixed by constant k. The points being ordered according to increasing Euclidian distance, we propose to interpolate the values of D_j of first closest, second closest, ... k-closest distances D_j. In practice, this works very well as distance fields represent a common solution for shape morphing. f_t is the mixture model. It determines how the A_j and the function $\omega_j(\mathbf{x})$ are computed for $\tilde{G}_j$. But these are all floating-point values that can be interpolated. Finally, J is the number of $\tilde{G}_j$. To interpolate between J_a and J_b, we take the maximum $J_{max} = \max\{J_a, J_b\}$ and linearly fade out the amplitudes A_j, $j > J_{max}$ of the kernels that are too much. This makes the surplus of kernels simply vanish during interpolation.

6. Results, Comparisons, and Discussion

The pseudo-code of the PPTBF is available in the supplemental materials. A full GPU implementation of the PPTBF is available on our website †, enabling the reproduction of our full database for parameter estimation. We also provide a software implementation of our semi-procedural texture synthesis method.

Structure modeling. We set up a database of 150 structures (see supplemental material) obtained by segmenting other texture databases to evaluate the ability of our PPTBF to reproduce these structures. Figure 11 shows a subset of this database. We experienced that the PPTBF is able to cover a large span of different structures (PPTBF is on the right). It succeeds as long as structural components do not have too complex spatial organization and shapes. We show examples of cells, dots, stains, cracks, waves, grains and scratches. We also compared our results to noise by example (Gabor noise [GLLD12] and LRP noise [GSV*14]). Not surprisingly these methods, designed for Gaussian and weakly structured textures, often fail to correctly reproduce structures. The entire database with these comparisons can be found in the supplemental material.

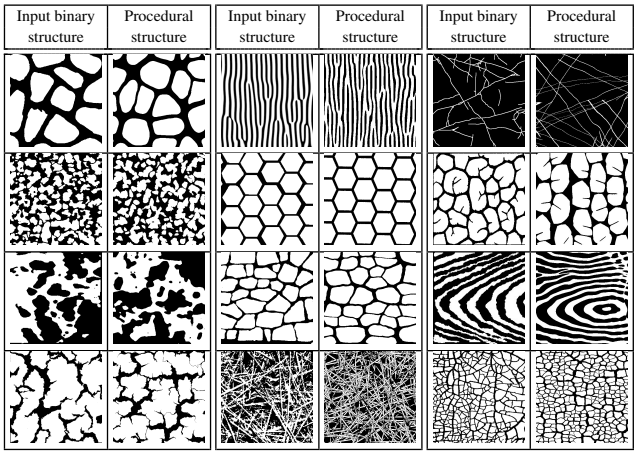

Figure 11: *Evaluation of the capability of PPTBF to produce natural structures (we use segmented images on left): parameters were estimated by querying a collection of 450k samples and by applying refinement.*

Material synthesis. Our approach synthesizes textures that match input exemplars. PPTBF provides a consistent structure, and data-driven synthesis provides visual details. Extensions to layered materials are shown in Figure 14. The top shows the exemplar, the bottom our semi-procedural texture.

Comparisons with example-based synthesis. Our semi-procedural synthesis can be compared with by–example texture synthesis, provided input structures match the input exemplars (typically when exemplars have been segmented). Figure 13 illustrates a comparison with state-of-the-art methods. These methods focus on quality but are not generative procedural approaches: textures cannot be modified and edited in realtime. Our semi-procedural approach transfers the structure of the PPTBF to the resulting texture. It generates varied, *i.e.,* novel and random

† https://github.com/ASTex-ICube/semiproctex

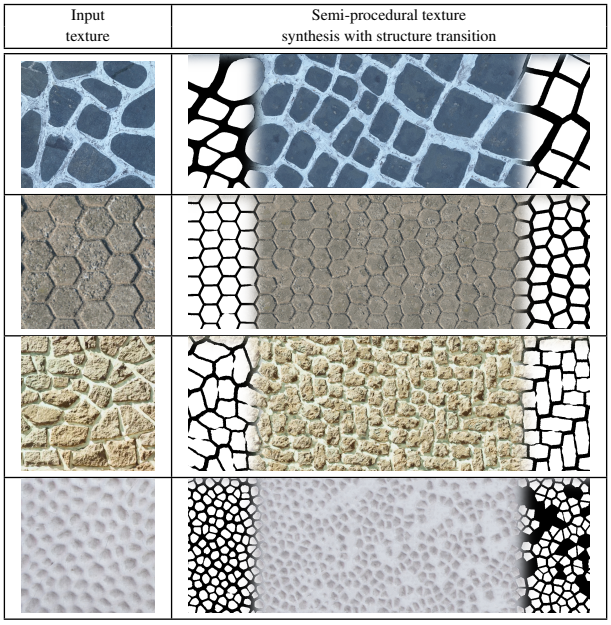

Figure 12: *Some semi-procedural texture synthesis results with varying PPTBF parameters.*

content, yet maintaining an overall consistent aspect with the input. Note that we are able to address some types of regular structures, which is related to the jitter parameter: when it is set to 0, randomness of point positions disappears and the underlying regularity of the tiling is revealed. However, our method does not address tilings or regular textures in general. Its main focus is on stochastic textures.

Comparisons with inverse procedural texture modeling. Inverse procedural texture modeling [HDR19] uses databases of procedural texture assets, combined with classification and parameter estimation. In spite of rich databases, there is no guarantee, as opposed to data-driven techniques, that a good visual match is obtained. Post-processing is necessary. Even the latter is not able to recover all details. Conversely, our approach uses a data-driven technique for synthesis: details are transferred from the exemplar, which guarantees a visual resemblance, even if the structure does not perfectly match. Moreover, the user can balance how much details are taken from the exemplar and how much structure is kept from the PPTBF. Figure 16 illustrates this: (a) is the input, (b) is result of [HDR19] (top: predicted, bottom: style transfer), (c) is our result. Though randomness is added, local leaf structures are better preserved.

Genericity. Our approach generates procedural textures, meaning that its parameters can be edited to control the creation of textures, and to easily create visual variants. Genericity is a key advantage of procedural modeling. The teaser illustrates a manual editing of some parameters. Figure 12 illustrates that users may alternatively provide two sets of parameters as input leading to two different structures (shown on the left and right side). The two distinct sets of PPTBF parameters can be then interpolated to provide smooth transitions. This is also possible for materials, as shown in Figure 15. Note that unlike inverse procedural modeling with style transfer, where the quality degrades when the procedural models are modified away from the original predicted results (as stated by [HDR19]), our approach conversely keeps a good visual match, even when the structure is strongly modified.

Performance. As shown in the accompanying video, the PPTBF can be edited interactively, including the use of details synthesis. In particular, we measured 9 ms for a 256^2 image, 37 ms for 512^2, 153 ms for 1024^2, and 606 ms for 2048^2 on a GeForce GTX 1060 with 6 GB of RAM.

Discussion. Our approach preserves some procedural generative properties, yet maintaining a visual match with an input exemplar. However, several natural structures are not covered by our PPTBF. Figure 17 illustrates a branching (tree) structure that cannot be represented by the point processes and tilings we used. Such structures are generally modeled using L-systems and grammars. Multiple labels (shown on the right) with specific spatial organizations also cannot be modeled with our PPTBF. Finally, while our semi-procedural approach inherits advantages of data-driven techniques, it also inherits their drawbacks: in some cases, because of the random search of best matching neighbors, results can become noisy. Smooth (low gradient) details can also not be well reproduced.

7. Conclusions

We introduced semi-procedural texture generation, a novel method that can be seen as a complementary approach between pure inverse procedural texture modeling and pure example-based texture synthesis. The procedural part is addressed with a point process texture basis function, that avoids the use of databases of procedural assets as well as classification. Yet, it permits to reach a wide range of structures.

We believe semi-procedural texture generation can be a starting point for many extensions to further improve controllable texture synthesis and content creation. First, the model lets itself well extend to more advanced generative algorithms, such as grammars for defining more complex tilings, point distributions and structures. Using feature functions with more elaborate PDFs can be imagined to further extend the range of reachable textures. Texture basis functions inherently let themselves well extend to higher dimensions, so that solid and volumetric texture synthesis from 2D exemplars could be addressed in future. Finally, the approach could be combined with procedural 3D object modeling augmented with scanned data, object shape being procedural and surface details being data-driven.

Devising a robust method for estimating PPTBF parameters from exemplars is another interesting topic for further research. Currently we use a technique based on a pre-computed database. It has limitations as it depends on the sampling of the PPTBF and the descriptors that are used. A learning-based approach could lead to more robust matching, e.g. via Gaussian Process Regression [ZWW18] or perceptual feature regression as proposed in [LGD*18].

Acknowledgments This work is supported by the *HDWorlds* project funded by the French National Research Agency (project ID: ANR-16-CE33-0001). Image courtesy: Textures.com, Mayang's free texture library (formerly available online at http://www.mayang.com/textures, now spread out on the web).

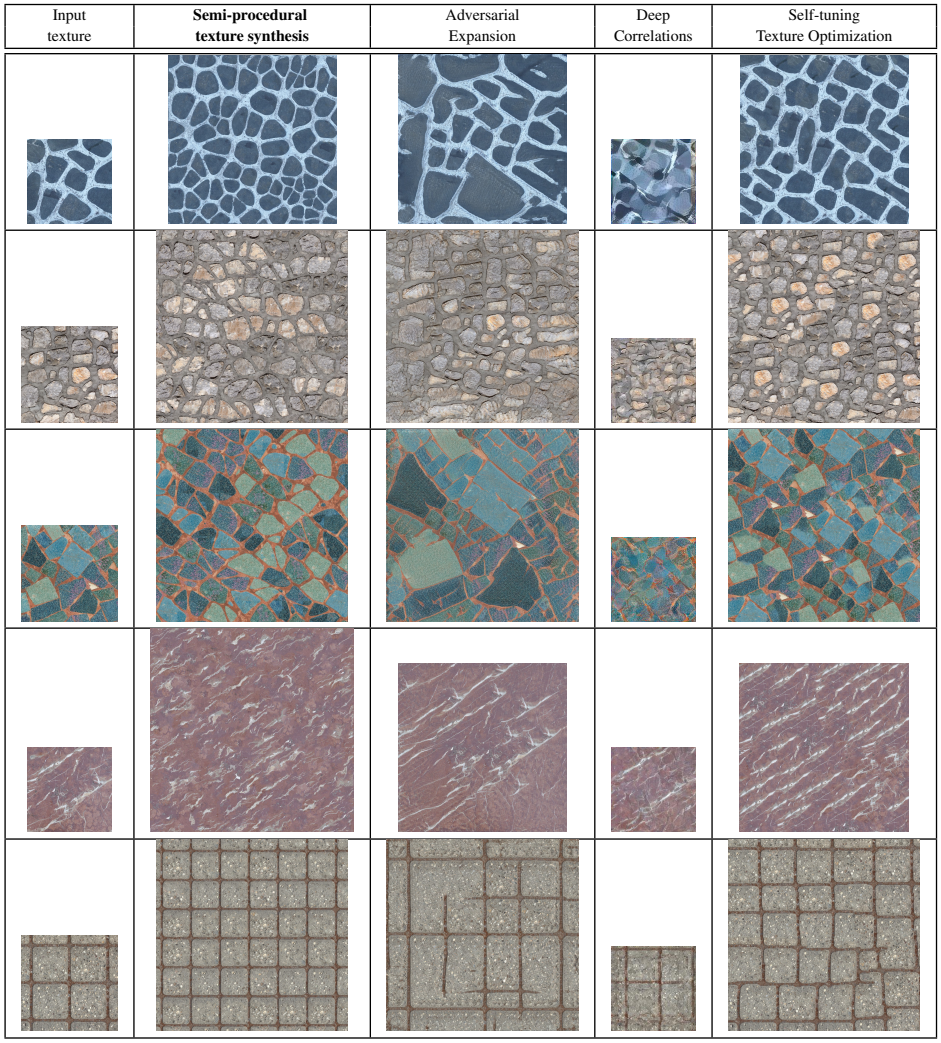

Figure 13: *Comparison of Semi-Procedural Texture Synthesis with state-of-the-art example-based texture synthesis methods. We show: our method; Adversarial Expansion [ZZB*18]; Deep Correlations [SC17]; Self-tuning Texture Optimization [KNL*15]. Note that Deep Correlations has been run on downsampled inputs (max. 256×256).*

References

[AAL16] AITTALA, M., AILA, T., and LEHTINEN, J. "Reflectance Modeling by Neural Texture Synthesis". *ACM Trans. Graph.* 35.4 (July 2016) 3.

[Ado19] ADOBE. *Substance Share*. `https://share.substance3d.com/`. 2019 2.

[AWL15] AITTALA, M., WEYRICH, T., and LEHTINEN, J. "Two-Shot SVBRDF Capture for Stationary Materials". *ACM Trans. Graph.* 34.4 (July 2015) 3.

[BD04] BOURQUE, E. and DUDEK, G. "Procedural Texture Matching and Transformation". *Comput. Graph. Forum* 23 (2004), 461–468 3.

[Bro66] BRODATZ, P. *Textures: A Photographic Album for Artists and Designers*. Dover photography collections. New York, NY, USA: Dover Publications, 1966 4.

[CGG19] CAVALIER, A., GILET, G., and GHAZANFARPOUR, D. "Local spot noise for procedural surface details synthesis". *Computers & Graphics* 85 (2019), 92–99 3, 4, 6.

[EF01] EFROS, A. A. and FREEMAN, W. T. "Image Quilting for Texture Synthesis and Transfer". *Proc. of SIGGRAPH'01*. ACM Siggraph. New York, NY, USA: ACM, 2001, 341–346 8.

[EMP*02] EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., et al. *Texturing and Modeling: A Procedural Approach*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002 2.

[FAW19] FRÜHSTÜCK, A., ALHASHIM, I., and WONKA, P. "TileGAN: Synthesis of Large-scale Non-homogeneous Textures". *ACM Trans. Graph.* 38.4 (July 2019), 58:1–58:11 3.

[GEB15] GATYS, L. A., ECKER, A. S., and BETHGE, M. "Texture Synthesis Using Convolutional Neural Networks". *Proceedings of the 28th International Conference on Neural Information Processing Systems*. NIPS'15. Montreal, Canada: MIT Press, 2015, 262–270 3.

[GEB16] GATYS, L. A., ECKER, A. S., and BETHGE, M. "Image Style Transfer Using Convolutional Neural Networks". *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, 2414–2423 2.

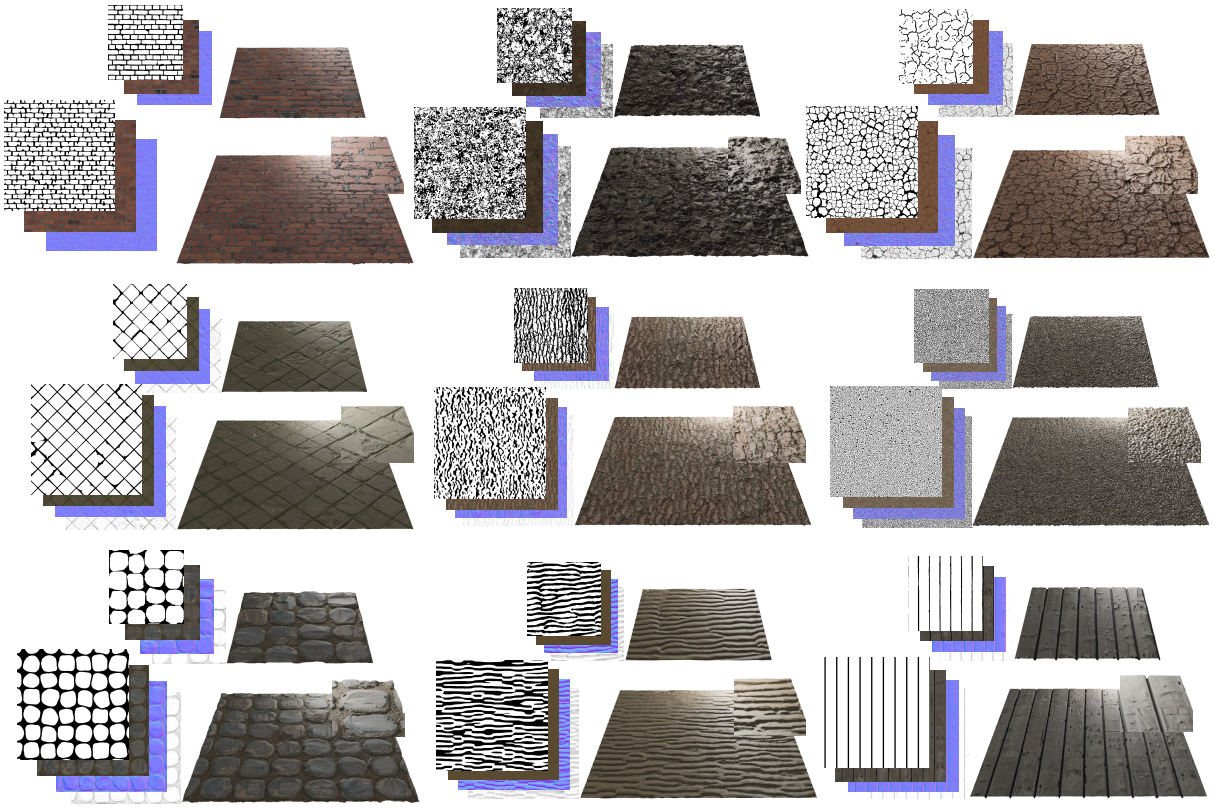

Figure 14: *Semi-procedural material synthesis for nine datasets. For each dataset, input exemplars are at the top and generated semi-procedural materials at the bottom. The right column shows corresponding renderings using albedo, roughness, and normal maps. Small insets exhibit closeup views of renderings for generated semi-procedural materials.*

[GKHF14] GIESEKE, L., KOCH, S., HAHN, J.-U., and FUCHS, M. "Interactive Parameter Retrieval for Two-tone Procedural Textures". *Proceedings of the 25th Eurographics Symposium on Rendering*. EGSR '14. Lyon, France: Eurographics Association, 2014, 71–79 3, 8.

[GLLD12] GALERNE, B., LAGAE, A., LEFEBVRE, S., and DRETTAKIS, G. "Gabor Noise by Example". *ACM Trans. Graph.* 31.4 (July 2012), 73:1–73:9 2, 3, 8, 9.

[GLM17] GALERNE, B., LECLAIRE, A., and MOISAN, L. "Texton Noise". *Computer Graphics Forum* (2017) 3, 6.

[GSDC17] GUINGO, G., SAUVAGE, B., DISCHLER, J.-M., and CANI, M.-P. "Bi-Layer Textures: A Model for Synthesis and Deformation of Composite Textures". *Comput. Graph. Forum* 36.4 (July 2017), 111–122 3.

[GSV*14] GILET, G., SAUVAGE, B., VANHOEY, K., et al. "Local Random-phase Noise for Procedural Texturing". *ACM Trans. Graph.* 33.6 (Nov. 2014), 195:1–195:11 2, 3, 6, 8, 9.

[HDR19] HU, Y., DORSEY, J., and RUSHMEIER, H. "A Novel Framework For Inverse Procedural Texture Modeling". *ACM Trans. Graph.* 38.6 (Nov. 2019) 2, 3, 10, 13.

[HJO*01] HERTZMANN, A., JACOBS, C. E., OLIVER, N., et al. "Image Analogies". *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, 327–340 3, 8.

[HN18] HEITZ, E. and NEYRET, F. "High-Performance By-Example Noise Using a Histogram-Preserving Blending Operator". *Proc. ACM Comput. Graph. Interact. Tech.* 1.2 (Aug. 2018), 31:1–31:25 3, 6.

[HRRG08] HAN, C., RISSER, E., RAMAMOORTHI, R., and GRINSPUN, E. "Multiscale Texture Synthesis". *ACM Trans. Graph.* 27.3 (Aug. 2008), 51:1–51:8 3.

[KEBK05] KWATRA, V., ESSA, I., BOBICK, A., and KWATRA, N. "Texture Optimization for Example-based Synthesis". *ACM Trans. Graph.* 24.3 (July 2005), 795–802 3.

[KNL*15] KASPAR, A., NEUBERT, B., LISCHINSKI, D., et al. "Self Tuning Texture Optimization". *Comput. Graph. Forum* 34.2 (May 2015), 349–359 3, 11.

[LCF*18] LIU, L., CHEN, J., FIEGUTH, P., et al. "From BoW to CNN: Two Decades of Texture Representation for Texture Classification". *International Journal of Computer Vision* (Nov. 2018) 8.

[Lew89] LEWIS, J. P. "Algorithms for Solid Noise Synthesis". *SIGGRAPH Comput. Graph.* 23.3 (July 1989), 263–270 3.

[LGD*18] LIU, J., GAN, Y., DONG, J., et al. "Perception-driven procedural texture generation from examples". *Neurocomputing* 291 (2018), 21–34 3, 8, 10.

[LH05] LEFEBVRE, S. and HOPPE, H. "Parallel Controllable Texture Synthesis". *ACM Trans. Graph.* 24.3 (July 2005), 777–786 3, 8.

[LH06] LEFEBVRE, S. and HOPPE, H. "Appearance-Space Texture Synthesis". *ACM Trans. Graph.* 25.3 (July 2006), 541–548 3.

[LLD11] LAGAE, A., LEFEBVRE, S., and DUTRE, P. "Improving Gabor Noise". *IEEE Transactions on Visualization and Computer Graphics* 17.8 (Aug. 2011), 1096–1107 2.

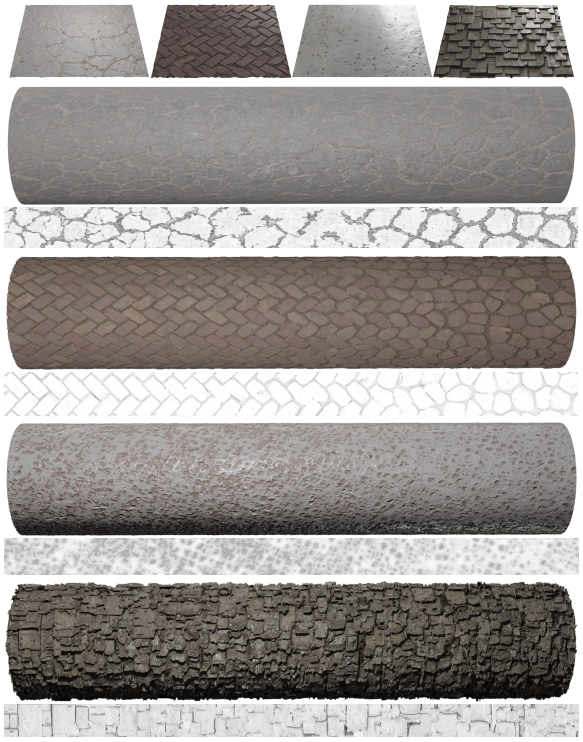

Figure 15: *Semi-procedural material transitions: input scanned material exemplars are on top. Renderings of variations with smooth transitions are shown on below (semi-procedural extrapolation).*

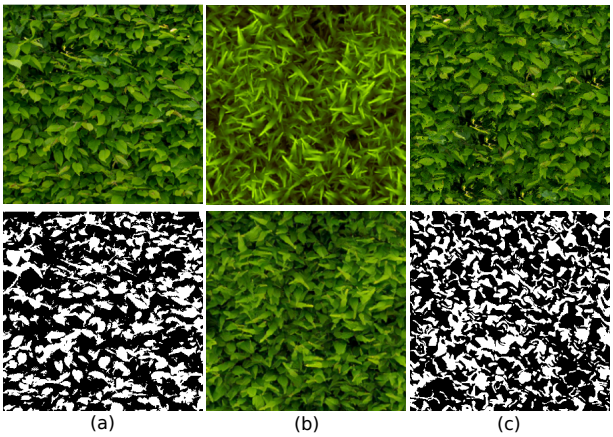

Figure 16: *Comparison with inverse procedural modeling [HDR19]: (a) is input, (b) inverse procedural modeling [HDR19] and (c) our result.*

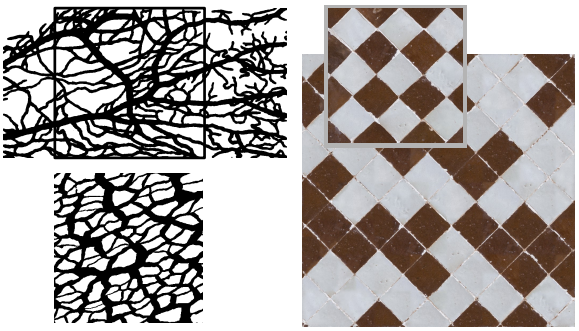

Figure 17: *Our PPTBF does not cover some types of structures, such as branching structures (top-left: input exemplar; bottom-left: estimated PPTBF after thresholding) or textures containing multiple labels with specific arrangements (top-right: input exemplar; bottom-right: semi-procedural synthesis result).*

[LLDD09] LAGAE, A., LEFEBVRE, S., DRETTAKIS, G., and DUTRÉ, P. "Procedural Noise Using Sparse Gabor Convolution". *ACM Trans. Graph.* 28.3 (July 2009), 54:1–54:10 2–4.

[LSA*16] LOCKERMAN, Y. D., SAUVAGE, B., ALLÈGRE, R., et al. "Multi-Scale Label-Map Extraction for Texture Synthesis". *ACM Trans. Graph.* 35.4 (July 2016) 9.

[ML09] MUJA, M. and LOWE, D. G. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, 331–340 8.

[ÖG12] ÖZTIRELI, A. C. and GROSS, M. "Analysis and Synthesis of Point Distributions Based on Pair Correlation". *ACM Trans. Graph.* 31.6 (Nov. 2012), 170:1–170:10 3.

[PELS10] PANAREDA BUSTO, P., EISENACHER, C., LEFEBVRE, S., and STAMMINGER, M. "Instant Texture Synthesis by Numbers". *Proc. Vision, Modeling & Visualization 2010*. 2010, 81–85 3.

[PS00] PORTILLA, J. and SIMONCELLI, E. P. "A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients". *International Journal of Computer Vision* 40.1 (2000), 49–70 3.

[RDDM17] RAAD, L., DAVY, A., DESOLNEUX, A., and MOREL, J.-M. "A survey of exemplar-based texture synthesis". *Annals of Mathematical Sciences and Applications* 3 (July 2017) 3.

[RÖG17] ROVERI, R., ÖZTIRELI, A. C., and GROSS, M. "General Point Sampling with Adaptive Density and Correlations". *Comput. Graph. Forum* 36.2 (May 2017), 107–117 3.

[SC17] SENDIK, O. and COHEN-OR, D. "Deep Correlations for Texture Synthesis". *ACM Trans. Graph.* 36.4 (July 2017) 3, 11.

[TEZ*19] TRICARD, T., EFREMOV, S., ZANNI, C., et al. "Procedural Phasor Noise". *ACM Transactions on Graphics* (2019) 3.

[WLKT09] WEI, L.-Y., LEFEBVRE, S., KWATRA, V., and TURK, G. "State of the Art in Example-based Texture Synthesis". *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009 3.

[Wor96] WORLEY, S. "A Cellular Texture Basis Function". *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, 291–294 2–4.

[ZHWW12] ZHOU, Y., HUANG, H., WEI, L.-Y., and WANG, R. "Point Sampling with General Noise Spectrum". *ACM Trans. Graph.* 31.4 (July 2012), 76:1–76:11 3.

[ZWW18] ZSOLNAI-FEHÉR, K., WONKA, P., and WIMMER, M. "Gaussian Material Synthesis". *ACM Trans. Graph.* 37.4 (July 2018), 76:1–76:14 3, 10.

[ZZB*18] ZHOU, Y., ZHU, Z., BAI, X., et al. "Non-stationary Texture Synthesis by Adversarial Expansion". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 37.4 (2018), 49:1–49:13 3, 11.