Texton Noise

Bruno Galerne, Arthur Leclaire, Lionel Moisan

Volume xx (200y), Number z, pp. 1–12

Texton Noise

B. Galerne[1], A. Leclaire[2] and L. Moisan[1]

[1]Laboratoire MAP5, Université Paris Descartes and CNRS, Sorbonne Paris Cité
[2]CMLA, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France

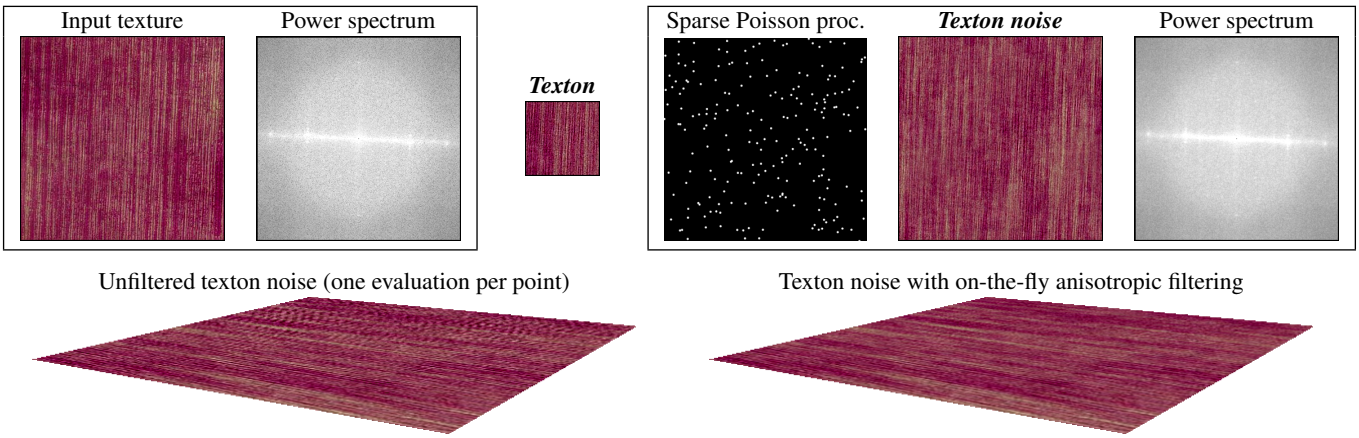

Figure 1: *We introduce an algorithm to compute a small bilinear texture called* **texton** *that summarizes the frequency content of an input texture image.* **Texton noise** *is a simple sparse convolution noise using the texton texture, producing output visually close to any input Gaussian texture even for a very sparse Poisson point process. This results in an unprecedented evaluation speed for noise by example while allowing for high-quality on-the-fly anisotropic filtering by simply invoking existing GPU hardware solutions for texture fetches.*

Abstract

Designing realistic noise patterns from scratch is hard. To solve this problem, recent contributions have proposed involved spectral analysis algorithms that enable procedural noise models to faithfully reproduce some class of textures. The aim of this paper is to propose the simplest and most efficient noise model that allows for the reproduction of any Gaussian texture. Texton noise *is a simple sparse convolution noise that sums randomly scattered copies of a small bilinear texture called* texton*. We introduce an automatic algorithm to compute the texton associated with an input texture image that concentrates the input frequency content into the desired texton support. One of the main features of texton noise is that its evaluation only consists to sum thirty texture fetches on average. Consequently texton noise generates Gaussian textures with an unprecedented evaluation speed for noise by example. A second main feature of texton noise is that it allows for high quality on-the-fly anisotropic filtering by simply invoking existing GPU hardware solutions for texture fetches. In addition, we demonstrate that texton noise can be applied on any surface using parameterization-free surface noise and that it allows for noise mixing.*

Categories and Subject Descriptors (according to ACM CCS): [Computer Graphics]: Image manipulation—Texturing

1. Introduction

Procedural noise is now a three-decades old technique introduced by Perlin [Per85] and widely used today for the generation of textures, the main focus of this paper, but also for the generation of other random contents involving spatial correlation such as terrains, surface bumps, and so on. The classical approach for generating a procedural texture is to combine several procedural noises with tweaked non linear functions (e.g. colormap and coordinate perturbation of a sinewave to create marble-like textures [Per85, LLC*10]).

However, recent contributions addressing *noise by example* [LVLD10, GLLD12, GSV*14] demonstrated that a fair amount of textures can be produced directly using a single procedural noise. The main advantage of this new noise by example approach is to propose algorithms that automatically estimate the noise parameters instead of relying on creative non linear tweaks. However, to achieve their goal, these papers use quite complex procedural noise models with many parameters, accompanied with involved spectral analysis for parameter setting and non trivial evaluation algorithm involving probabilistic sampling of precomputed frequency tables. As acknowledged in [LLC*10], all standard procedural noise functions produce approximately Gaussian outputs since they all mix linearly independent random inputs. Hence, a natural class of textures for procedural noise is the class of Gaussian textures, which are textures solely defined by their power spectrum (see Section 3 for a discussion on Gaussian textures).

The main motivation of this work is to propose the simplest possible (and thus most efficient) noise model that allows for the reproduction of any Gaussian texture. We present *texton noise* which is a *sparse convolution noise* [Lew84] that uses a single kernel, called *texton*. This means that our noise simply consists in summing randomly positioned copies of the texton. This texton kernel determines alone the power spectrum of the noise. It can be seen as a compact summary of the input texture, and thus provides an "inverse texture synthesis" [WHZ*08, SCSI08] solution for Gaussian textures. To ensure a fast evaluation and a compact representation of the noise, the texton is modeled as a generic bilinearly-interpolated function having a small support, that is, a small GPU texture. Our main contribution is to show that this simple noise model is sufficient to reproduce any Gaussian texture, with a texture analysis and a procedural evaluation that are several orders of magnitude faster than recent competing approaches [GLLD12, GSV*14] since they do not rely on the usual spectral decomposition/frequency sampling approach. The main features of *texton noise* are the following:

1. **Arbitrary spectrum:** The noise can approximate any general spectrum and the evaluation cost does not increase with the complexity of the spectrum that is summarized within the texton. It is naturally designed for noise by example and is able to reproduce any Gaussian texture.
2. **Very fast evaluation:** The cost for evaluating texton noise at a point is the one of 30 texture fetches on average, without any GPU evaluation of trigonometric function.
3. **On-the-fly antialiasing:** Texton noise allows for simple and very efficient antialiasing filtering by simply invoking existing GPU hardware solutions for texton texture fetches.
4. **Automatic analysis:** The texton computation is fully automatic and fast.
5. **Simple noise storage:** Texton noise is represented by two parameters, the noise mean color and the texton stored as a GPU texture. The noise hence have a controlled compact memory cost (size of the texton) and does not involve any specific data structure.

In addition, texton noise generates textures directly in RGB space without using any colorspace transformation and allows for surface noise, that is, noise on surface without surface parameterization. An additional contribution of the paper is to demonstrate that texton noise allows for spatial noise mixing relying on recent contributions in Gaussian texture mixing [XFPA14].

2. Related Work

2.1. Gaussian textures

The richness of the Gaussian texture model was demonstrated in [vW91]. The author introduces the spot noise model that consists in the addition of randomly shifted copies of a geometric kernel, and demonstrates that this model can produce a large class of microtextures by varying the kernel. The study of Gaussian textures has recently seen several developments in the field of image processing. In [GGM11], the asymptotic discrete spot noise model was clearly expressed as a stationary Gaussian random field. The authors also proposed an automatic analysis/synthesis algorithm based on the discrete Fourier transform (DFT) for the synthesis by example of Gaussian texture images. Gaussian texture images correspond to microtexture images whose perception is not affected by random shuffling of the Fourier phase. A Gaussian texture is thus characterized by its Fourier modulus. Desolneux et al. [DMR12] introduced a compact representation of a Gaussian texture by considering the image with same Fourier modulus and zero Fourier phase, called "canonical texton". Galerne et al. [GLM14] showed that this compact "canonical texton" is not suited for direct spot noise synthesis, and introduced an algorithm for the computation of a "synthesis-oriented texton" that can be used for low-intensity spot noise synthesis of Gaussian texture images, which is faster than Fourier simulation and allows for parallel local evaluations. Furthermore, as demonstrated in [XFPA14], it is possible to rigorously solve the problem of texture mixing for the Gaussian model using optimal transport barycenters between probability distributions. We demonstrate in this paper that texton noise also allows for noise mixing. To finish, let us also mention that the Gaussian model is also convenient for dynamic texture modeling [DCWS03, XFPA14].

Except for [vW91], all these image processing papers only considered Gaussian textures on discrete pixel grid. By using a similar probabilistic approach, we demonstrate with texton noise that it is possible to extend the results of [GLM14] and [XFPA14] to the continuous framework of procedural noise functions while fulfilling specific technical requirements such as fast evaluation and on-the-fly filtering.

2.2. Noise by example

As recalled in the introduction, procedural noise is a three-decades old technique introduced in [Per85]. Several procedural noise functions have been proposed and we refer to the survey [LLC*10] for a complete discussion. Let us just mention again that texton noise is a sparse convolution noise [Lew84] which is also called spot noise [vW91]. This paper tackles the noise by example problem, that is, producing a noise that is visually similar to an input texture image. This subject has seen several developments in these past five years, and we refer to the papers [GLLD12] and [GSV*14] for description of related works previous to 2010. The main idea of noise by example is to find noise parameters that match the power spectrum of the input image. The limitations of existing methods are

threefold: non fully automatic analysis requiring manual intervention, model limitation, slow procedural evaluation.

Gilet et al. [GDS10] proposed a segmentation approach of the power spectrum using ellipses to determine the parameters of a Gabor noise model [LLDD09]. However, their method requires significant manual intervention and trial for each exemplar.

Lagae et al [LVLD10] estimated frequency bandwidth energies to set the parameters of a wavelet noise [CD05]. Their analysis is fully automatic but the approach is limited to isotropic textures.

With Gabor noise by example (GNBE), Galerne et al. [GLLD12] demonstrated that any Gaussian texture could be obtained with a Gabor noise [LLDD09] using several bandwidths. The resulting noise is visually indistinguishable from the targeted Gaussian texture, but the proposed automatic analysis procedure takes several minutes and the procedural evaluation is relatively slow.

Lately, Gilet et al [GSV*14] introduced *local random phase* noise (LRP noise). This new noise uses a regular spatial grid to sum localized cosines with random or deterministic phases. The notable contribution of this work is to generate noise with structured patterns by preserving some local phases of the input texture, making it the state of the art method. However, when applied to Gaussian textures, the output noise is slightly different from the targeted Gaussian texture due to a slightly approximate spectral analysis and the use of the *principal variation color space* [VSLD13] based on color clustering.

In comparison, texton noise is theoretically guaranteed to produce Gaussian textures, the parameter setting is fast, simple and automatic, and the evaluation speed is unprecedented for noise by example. Texton noise allows for the faithful reproduction of any Gaussian texture, but, similarly to GNBE [GLLD12] it is strictly limited to Gaussian textures, and thus cannot produce more structured textures contrary to LRP noise [GSV*14]. In short, texton noise improves significantly the state of the art for noise by example applied to Gaussian textures, while LRP noise remains the unchallenged state of the art for noise by example for structured textures.

2.3. Texture Tiling and Texture Bombing

Let us recall that many procedural methods consist in assembling texture pieces on the synthesis domain. The chaos mosaic technique [GSX00] shuffles the blocks of the tiled input texture and next solves efficiently the mismatched features on the block boundaries. Similarly, the algorithm of [CSHD03] assembles precomputed Wang tiles (with matching boundaries) in order to progressively fill the synthesis domain. An improvement was later suggested in [Wei04] allowing for hardware implementation by packing the Wang tiles in a texture map. We refer to [LKF*08] for further details about tiling applications in graphics. The method of [LN03] consists in throwing predefined patterns in the cells of a mosaic in a randomized manner. Texture bombing [Gla04] samples textures in the dead leaves model [Mat68] which means that several predefined patterns are thrown on the synthesis domain with an occlusion principle. Lefebvre et al. [LHN05] proposed an optimized method allowing for interactive animated texture design: the user can drop (and edit) texture sprites (i.e. small patterns) which can be assembled with a customized blending principle. Vanhoey et al. [VSLD13] suggest to break the repetitions of a periodic tiling by random replacements of well-chosen interchangeable patches.

In all these methods, each pixel or each cell is covered by one dominant texture pattern, and thus, they are more adapted to macro-textures. In contrast, with texton noise each point is covered by several randomly shifted copies of the texton which are simply summed. We will see on the results that when covering a point with 30 textons, individual textons are not distinguishable and that the noise patterns only result from the noise spectrum. In short, the texton is not a precomputed piece of texture but a small texture summarizing the frequency content of a Gaussian texture.

3. Background on Gaussian Textures and Shot Noise

3.1. Gaussian Texture Synthesis

Given any image u defined on the pixel domain $\Omega = \{0,\dots,M-1\} \times \{0,\dots,N-1\}$ containing $|\Omega| = MN$ pixels and having mean color $\text{mean}(u)$, one samples the Gaussian version of u simply by convolving the normalized image $\frac{1}{\sqrt{MN}}(u - \text{mean}(u))$ with a standard Gaussian white noise (the pixel values are independent and have Gaussian distribution with mean 0 and variance 1), and then add $\text{mean}(u)$ to each pixel [GGM11]. By definition, Gaussian textures are the texture images that are visually well-reproduced by this simple convolution-based procedure, and they are characterized by their mean color and their covariance or power spectrum. Several pairs of textures and their Gaussian counterparts are shown in the two first rows of Figure 7 and demonstrate that the Gaussian model is realistic for various fabrics, wood, rocks,... Still Gaussian textures form a limited class of textures since they do not contain sharp contours. However this limitation is counterbalanced by the fact that their precise probabilistic definition enables to derive algorithms with theoretical guarantee of success for texture synthesis [GGM11] as well as for texture mixing [XFPA14]. The goal of this paper is to achieve both these tasks with an efficient procedural noise function. This is challenging since one cannot use grid-based FFT convolutions in a continuous framework. This is the reason why we now turn to approximate Gaussian simulations using high intensity shot noise.

3.2. Discrete Shot Noise Associated with an Image

The previous section describes the Gaussian texture synthesis on a finite pixel grid Ω while we would like to achieve Gaussian texture synthesis on the domain $\mathbb{R}^2$. The difference is twofold: 1) $\mathbb{R}^2$ is infinite; 2) $\mathbb{R}^2$ is continuous. The second point deserves more attention since defining a Gaussian texture model on the infinite discrete domain $\mathbb{Z}^2$ is straightforward. Given an image $u \in \mathbb{R}^{M\times N}$ as in the previous section, one defines the normalized version of u extended to $\mathbb{Z}^2$ by $t_u(k) = \frac{1}{\sqrt{|\Omega|}}(u(k) - \text{mean}(u))$ if $k \in \Omega$, and 0 otherwise, so that the auto-correlation of u can be written

$$\tilde{t}_u \star t_u(k) = \frac{1}{|\Omega|} \sum_{l\in\Omega} (u(l) - \text{mean}(u))(u(l+k) - \text{mean}(u))$$

with the convention that $u(l+k) - \text{mean}(u) = 0$ if $l+k \notin \Omega$, where $\star$ denotes convolution between $\mathbb{Z}^2$-indexed sequences, and where

$\tilde{t}_u(k) = t_u(-k)$. Then the discrete shot noise (DSN) of intensity $\lambda > 0$ associated with u is defined on $\mathbb{Z}^2$ [GLM14] by

$$F_u(k) = \sum_j t_u(k - k_j), \quad k \in \mathbb{Z}^2,$$

where $\{k_j\} \subset \mathbb{Z}^2$ is a Poisson process of intensity λ over $\mathbb{Z}^2$. This DSN has expectation zero and its covariance is $\lambda \tilde{t}_u \star t_u$, and when the Poisson intensity λ tends to $+\infty$ the variance-normalized DSN $\frac{1}{\sqrt{\lambda}} F_u$ tends towards a Gaussian random field over $\mathbb{Z}^2$ with mean 0 and covariance $\tilde{t}_u \star t_u$ which is called Asymptotic DSN (ADSN). Experiments show that if u is a Gaussian texture, then both ADSN and DSN with high λ generate textures that are visually similar to the Gaussian input, while defined on the infinite domain $\mathbb{Z}^2$. Our goal in what follows is to propose a procedural texture model, defined on the continuous domain $\mathbb{R}^2$, that produces the same visual quality.

3.3. Single Kernel Shot Noise

Strictly speaking, the texton noise model is not new since it is a sparse convolution noise with a single kernel. The novelty of our approach consists in the computation of a specific kernel whose corresponding noise can approximately reproduce any Gaussian texture or any desired noise spectrum in a very efficient way.

The sparse convolution noise [Lew84,LLDD09], also called spot noise in graphics [vW91], is what is denominated a shot noise process in applied probability [Pap71, Ric77]. It simply consists in the sum of several copies of a kernel h that are randomly shifted all over the plane $\mathbb{R}^2$ according to a Poisson process of intensity $\lambda > 0$ denoted by Π_λ. The resulting shot noise is thus the random field

$$f_\lambda(x) = \sum_{x_j \in \Pi_\lambda} h(x - x_j), \quad x \in \mathbb{R}^2, \tag{1}$$

where the kernel $h : \mathbb{R}^2 \to \mathbb{R}$ is such that both integrals $\int |h(x)|dx$ and $\int h^2(x)dx$ are finite. We will speak of single kernel shot noise since the function h is fixed and deterministic. Let us clarify some notation. For a function $g : \mathbb{R}^2 \to \mathbb{R}$, $\tilde{g}$ denotes the symmetric function $\tilde{g}(y) = g(-y)$, $\hat{g}$ denotes its Fourier transform $\hat{g}(\xi) = \int_{\mathbb{R}^2} h(y) e^{-2i\pi\langle \xi, y\rangle} dy$, and $*$ denotes the convolution product between functions defined on $\mathbb{R}^2$.

Proposition 1 (Shot noise mean, covariance and power spectrum) The random field f_λ defined by (1) is well-defined, stationary and has a finite variance. Its expectation is given by $\mathbb{E}(f_\lambda(x)) = \lambda \int_{\mathbb{R}^2} h(y)dy$. Its covariance function is equal to $\tau \mapsto \lambda C(\tau)$ where $C(\tau) = \int_{\mathbb{R}^2} h(y+\tau)h(y)dy = \tilde{h} * h(\tau)$ is the autocorrelation of h, its power spectrum is thus $S(\xi) = \lambda \hat{C}(\xi) = \lambda \left|\hat{h}(\xi)\right|^2$.

Single kernel shot noise are well suited for the generation of Gaussian textures thanks to the following Gaussian convergence theorem (proved *e.g.* in [Pap71]).

Theorem 2 (Shot noise Gaussian convergence) As the Poisson point process intensity λ tends to $+\infty$, the normalized random field

$$g_\lambda(x) = \frac{f_\lambda(x) - \mathbb{E}(f_\lambda(x))}{\sqrt{\lambda}} \tag{2}$$

converges in distribution towards a Gaussian random field with mean 0 and covariance function $C = \tilde{h} * h$.

This key theorem furnishes the recipe for *noise by example* with a single kernel shot noise: given a Gaussian texture image u as input,

1. **Analysis step:** Estimate the input discrete covariance $C_u = \tilde{t}_u \star t_u$ and determine a kernel h such that $C_u \simeq \tilde{h} * h$,
2. **Evaluation step:** Generate the normalized shot noise $g_\lambda(x)$ with a sufficiently high λ.

Shot noise evaluation cost A critical feature of a procedural noise is the computational cost of its evaluation. Let us stress that the relevant quantity regarding the shot noise evaluation cost is not the Poisson process intensity λ but the mean number of kernels h covering a point (that is the mean number of non null terms in the sum (1)) that we call mean number of impacts (MNI). For a kernel with a square support S of size $r \times r$, the MNI is λr^2. On the opposite, if the support of the kernel does have infinite area, the MNI is infinite and the evaluation is not feasible.

4. Texton Noise

As said in the introduction, we model the kernel function h as a generic bilinearly-interpolated function so that it can be stored in GPU texture memory and evaluated using standard GPU fetch algorithm. Hence,

$$h(y) = \sum_{k \in \mathbb{Z}^2} \alpha(k)\psi(y - k), \quad y \in \mathbb{R}^2, \tag{3}$$

where the interpolation coefficients α should be 0 outside a small finite domain and ψ is the usual 2D bilinear interpolation kernel, *i.e.* $\psi(y_1, y_2) = (1 - |y_1|)^+(1 - |y_2|)^+$ where $x^+ = x$ if $x \geq 0$ and 0 otherwise. Remark that $\alpha(k) = h(k)$ for all $k \in \mathbb{Z}^2$ but it is preferable to differentiate the notation between the stored coefficients α and the continuous function h (also, $\alpha(k) = h(k)$ is not valid if one uses higher order $\boldsymbol{B}$-spline interpolation, see the discussion in supplementary material). From now on, the bilinearly-interpolated kernel h will be called *texton*, the interpolation coefficients α will be called *texton interpolation coefficients*, and the corresponding normalized shot noise g_λ defined by (2) will be called *texton noise*. Remark that a texton noise is solely determined by its interpolation coefficients α. The main problem is to determine the texton interpolation coefficients α that enable to reproduce an exemplar Gaussian texture image $u \in \mathbb{R}^{M \times N}$.

4.1. Sampled Covariance Consistency

As said above, our goal is to define a texton noise g_λ that produces the same visual quality as the ADSN associated with the input texture image u. However, these two random objects are different in nature since a shot noise is defined over the continuous domain $\mathbb{R}^2$ while an ADSN is defined over the discrete lattice $\mathbb{Z}^2$. The problem is to find a continuous covariance function $C : \mathbb{R}^2 \to \mathbb{R}$ of the form $C = \tilde{h} * h$ (see Proposition 1) that extends the discrete covariance function $C_u = \tilde{t}_u \star t_u : \mathbb{Z}^2 \to \mathbb{R}$. This is of course an ill-posed problem. We claim that the requirement that the procedural noise g_λ should be visually similar to the ADSN model over $\mathbb{Z}^2$ associated with u translates in the following constraint: **The sampling over $\mathbb{Z}^2$ of g_λ must have the same covariance as the ADSN associated with u.** We call this condition **the sampled covariance**

consistency. Note that since g_λ is stationary over $\mathbb{R}^2$, the distribution of the sampled field $(g_\lambda(k))_{k\in\mathbb{Z}^2}$, $y\in\mathbb{R}^2$, is the same as the one of any discrete field $(g_\lambda(y+k))_{k\in\mathbb{Z}^2}$ sampled on some translated grid $y+\mathbb{Z}^2$. The sampled covariance consistency condition writes $C(k)=C_u(k)$ for all $k\in\mathbb{Z}^2$, that is,

$$\tilde{h}*h(k)=\sum_{l\in\mathbb{Z}^2}(\tilde{\alpha}\star\alpha)(l)\,(\tilde{\psi}*\psi)(k-l)=\tilde{t}_u\star t_u(k),\quad k\in\mathbb{Z}^2,$$

where, again, $*$ denotes the convolution between continuous functions and $\star$ denotes the convolution between discrete functions. Let us denote by $b:\mathbb{Z}^2\to\mathbb{R}$ the sampling over $\mathbb{Z}^2$ of the function $\tilde{\psi}*\psi=\psi*\psi$, that is $b(k)=\psi*\psi(k)$ for all $k\in\mathbb{Z}^2$. b is simply the discrete sampling of the $\boldsymbol{B}$-spline kernel of order 3: its coefficients are the one of the 3×3 matrix $\begin{pmatrix}\frac{1}{6} & \frac{2}{3} & \frac{1}{6}\end{pmatrix}^T\begin{pmatrix}\frac{1}{6} & \frac{2}{3} & \frac{1}{6}\end{pmatrix}$ on $\{-1,0,1\}^2$ and zero outside $\{-1,0,1\}^2$. Taking the Fourier transform over $\mathbb{Z}^2$ of the above equation gives the translation of the sampled covariance consistency condition in terms of power spectrum,

$$|\hat{\alpha}(\xi)|^2\,\hat{b}(\xi)=|\hat{t}_u(\xi)|^2,\quad \xi\in[-\tfrac{1}{2},\tfrac{1}{2}]^2. \tag{4}$$

Hence an ideal solution would be to find interpolation coefficients $(\alpha(k))_{k\in\mathbb{Z}^2}$ such that $|\hat{\alpha}(\xi)|^2=\frac{|\hat{t}_u(\xi)|^2}{\hat{b}(\xi)}$. However the ideal story stops here. Indeed, a $\mathbb{Z}^2$-indexed sequence has finite support if and only if its Fourier transform is a trigonometric polynomial. Hence, both $|\hat{t}_u(\xi)|^2$ and $\hat{b}(\xi)$ are trigonometric polynomials, and unless the input image u has been especially designed, there is no reason for $\hat{b}(\xi)$ to be a divisor of $|\hat{t}_u(\xi)|^2$. Hence, any solution $(\alpha(k))_{k\in\mathbb{Z}^2}$ to the equation $|\hat{\alpha}(\xi)|^2=\frac{|\hat{t}_u(\xi)|^2}{\hat{b}(\xi)}$, $\xi\in[-\frac{1}{2},\frac{1}{2}]^2$ corresponds to a sequence with infinite support. This observation is rather disappointing: the ideal interpolation coefficients α for the kernel function h would result in a noise with infinite memory footprint and infinite evaluation cost! This leads us to turn to an approximation of the sampled covariance consistency condition with a controlled memory budget, that is, with a prescribed support for the coefficients α.

4.2. Texton Computation Algorithm

Since there is no exact solution of the spectral equation (4) that has a finite support, we turn to an approximate solution for this equation that has a user-defined support. The smaller the support, the compacter the noise, although too small support would generally impact the noise visual quality (see discussion in Section 9). Given a small square domain $S\subset\mathbb{Z}^2$, we search for the best texton interpolation coefficients $(\alpha(k))_{k\in\mathbb{Z}^2}$ such that

1. α has support in S: for all $k\notin S$, $\alpha(k)=0$.
2. $|\hat{\alpha}(\xi)|^2\,\hat{b}(\xi)\approx|\hat{t}_u(\xi)|^2$ for all $\xi\in[-\frac{1}{2},\frac{1}{2}]^2$.

The second point leaves room for interpretation. Since both $|\hat{\alpha}(\xi)|^2\,\hat{b}(\xi)$ and $|\hat{t}_u(\xi)|^2$ are trigonometric polynomials, and thus smooth functions, it is arguably enough to enforce that all their values on a regular grid are close. This leads to a practical numerical solution, since $|\hat{t}_u(\xi)|^2$ is easily evaluated on a regular grid by Fast Fourier Transform (FFT). We propose the alternating projection algorithm described in Algorithm 1 to compute the texton interpolation coefficients α.

Within this algorithm, all images have the same size as the input u defined on the domain Ω and the hat symbol $\hat{\cdot}$ refers to the DFT on Ω (which is computed by FFT).

- **Input:** Image u defined on Ω, coefficients support $S\subset\Omega$
- **Compute spectral constraint:**
  - Compute $t_u=\frac{1}{\sqrt{|\Omega|}}(u-\mathrm{mean}(u))$ and its DFT $\hat{t}_u$
  - Compute the sampling of the $\boldsymbol{B}$-spline of order 3 b in Ω (with the DFT periodicity convention) and its DFT $\hat{b}$
- **Random initialization:** Initialize $\alpha\in\mathbb{R}^\Omega$ as a standard white Gaussian noise image
- **Iterative constraint projection:** Do $n=50$ times
  1. **Impose power spectrum** of $\frac{\hat{t}_u}{\sqrt{\hat{b}}}$:
     a. Compute the DFT $\hat{\alpha}$
     b. Set $\hat{\alpha}\leftarrow\frac{|\hat{t}_u|}{\sqrt{\hat{b}}}\frac{\hat{\alpha}}{|\hat{\alpha}|}$
     c. Compute α by backward DFT
  2. **Impose support** S: Set $\alpha\leftarrow\mathbb{1}_S\cdot\alpha$
- **Output:** Return α

Algorithm 1: Computation of the texton interpolation coefficients

The main motivation of this procedure (which is strongly inspired by the computation of the discrete synthesis-oriented texton [GLM14]) is to define coefficients α that have the targeted spectrum $\frac{\hat{t}_u}{\sqrt{\hat{b}}}$, with support S and with a DFT phase as random as possible (obtained with the Gaussian white noise initialization). As a preprocess for Algorithm 1, if the input image u is not strictly periodic it is replaced by its periodic component [Moi11] as in previous work on Gaussian texture synthesis. Figure 2 displays a texton along with texton noise simulation with varying MNI. As can be observed from this figure, texton noise is visually close to the targeted Gaussian texture for a MNI of 30, which enables for a very fast noise evaluation (see Section 9 for performance discussion). By construction texton noise is a sum of bilinearly-interpolated textons that are translated by the continuous vectors x_j of (1). However texton noise does not suffer from the usual bilinear interpolation "blocky" artefacts since these translation vectors live in the continuous domain $\mathbb{R}^2$ (see discussion and illustration in the supplementary material).

4.3. Color Texton Noise

Texton noise is naturally defined for Gaussian color textures simply by allowing the kernel h of (1) to be a bilinear RGB texture image $\boldsymbol{h}$ with coefficients $\boldsymbol{\alpha}=(\alpha_R,\alpha_G,\alpha_B)^T$. Hence color texton noise is generated directly in RGB space with a single Poisson point process, the computation cost of color texton noise is thus the same as the one of its gray-level counterpart. In addition, since we generate directly the texture in RGB space without using non-linear

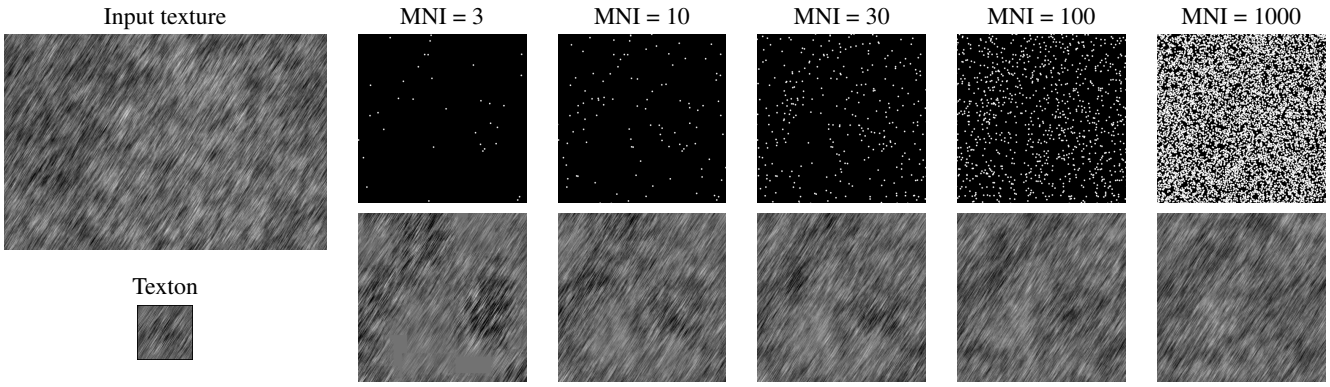

Figure 2: *Left column: Input texture and texton of size 128×128. Subsequent columns: Texton noises with various MNI with corresponding Poisson processes. Remark that the texton noise textures are visually close to the Gaussian limit as soon as MNI equals 30.*

colormaps, our anisotropic filtering procedure (see Section 5) applies directly to color texton noise.

Computation of Color Texton Interpolation Coefficients Relying on previous work on RGB Gaussian textures, we can adapt the computation of the texton interpolation coefficients $\boldsymbol{\alpha} = (\alpha_R, \alpha_G, \alpha_B)^T$ to take into account the color channel correlations of the RGB input $\boldsymbol{u}$. For that, one simply replaces the spectral projection of step 1.b of Algorithm 1 by $\hat{\boldsymbol{\alpha}} \leftarrow \frac{1}{\sqrt{\hat{b}}} \frac{\hat{\boldsymbol{t}}_{\boldsymbol{u}}^* \hat{\boldsymbol{\alpha}}}{|\hat{\boldsymbol{t}}_{\boldsymbol{u}}^* \hat{\boldsymbol{\alpha}}|} \hat{\boldsymbol{t}}_{\boldsymbol{u}}$, where * denotes the conjugate transpose of a complex vector and the Fourier transform of an RGB image is obtained by applying the Fourier transform to each color channel (Please refer to supplementary material for a rigorous justification).

Color Correlation Correction Due to the support projection (step 2. of Algorithm 1) that puts some coefficients to zero, the variance of each color channel is always smaller than the one of the original image. This loss of variance translates visually into textures with less contrast and is not desirable. We thus perform the simple linear color correction proposed in [GLM14, DMR15] once the coefficients are computed (Please refer to the supplementary material for technical details and illustrations). This ensures that the covariance of the color texton noise is equal to the cross-correlation of the input texture. Note that this color correction is done at the analysis step, and once the corrected coefficients are computed there is no need to store the cross-correlation matrices. Contrary to previous work relying on adapted color spaces such as PCA-like color space [LVLD10, GLLD12] or *principal variation color space* [GSV*14], the generation of texton noise is done directly in the standard RGB space and thus gains in both speed and simplicity.

5. Filtering Texton Noise

Filtering noise according to the viewing condition is crucial to limit aliasing. The main issue is to avoid a costly hypersampling procedure. State of the art noises allow for either isotropic [CD05] or anisotropic [GZD08, LLDD09] filtering by frequency attenuation or clamping adapted to the distance to camera. Besides, for color noise these linear filtering techniques are possible only when using a linear color space (e.g. RGB or PCA-like). Using any non-linear colormap to obtain a color noise requires to use more involved (and more costly) filtering procedures [LLC*10, GSV*14].

According to the survey [LLC*10], a major drawback of procedural noises is that classical filtering techniques for GPU textures such as MIP-mapping are not available. One important feature of texton noise is that it can be filtered using these standard techniques. Indeed, as can be seen from (1), texton noise is simply a sum of randomly scattered copies of the texton h. Hence, by linearity of the convolution, locally filtering texton noise can be achieved by locally filtering the texton h. Since the texton is a bilinear texture stored in classical texture memory, we can apply standard MIP-mapping lookup and anisotropic filtering to it when fetching the texture values. This results in a high-quality anisotropic filtering of our noise for only a slight additional computation cost since these filtering implementations are highly optimized in GPU. Regarding implementation, our anisotropic filtering strategy comes for free since it only involves a few OpenGL code lines when declaring the texton texture properties (generating MIP-map levels and enabling anisotropic filtering) with zero modification of the fragment shader dedicated to texton noise evaluation (note however that one could use any filtering method for bilinear textures instead of the standard implementation). Figure 3 illustrate that our proposed on-the-fly filtering is visually close to an ideal hypersampled noise. Figure 1 and the video in the supplementary material also illustrate the aliasing correction. We observed experimentally that the speed performance of the anisotropic filtering depends on the viewing conditions, from being as fast as unfiltered texton noise to being twice slower, while the hypersampling scheme presented in Figure 3 is 400 times slower than unfiltered texton noise. Hence, texton noise allows for high-quality on-the-fly anisotropic filtering with limited additional computational cost.

6. Surface Noise

Quoting the survey [LLC*10], there are three different ways to obtain noise on surfaces: 1) mapping a 2D noise onto the surface using a planar parameterization, which is straightforward but

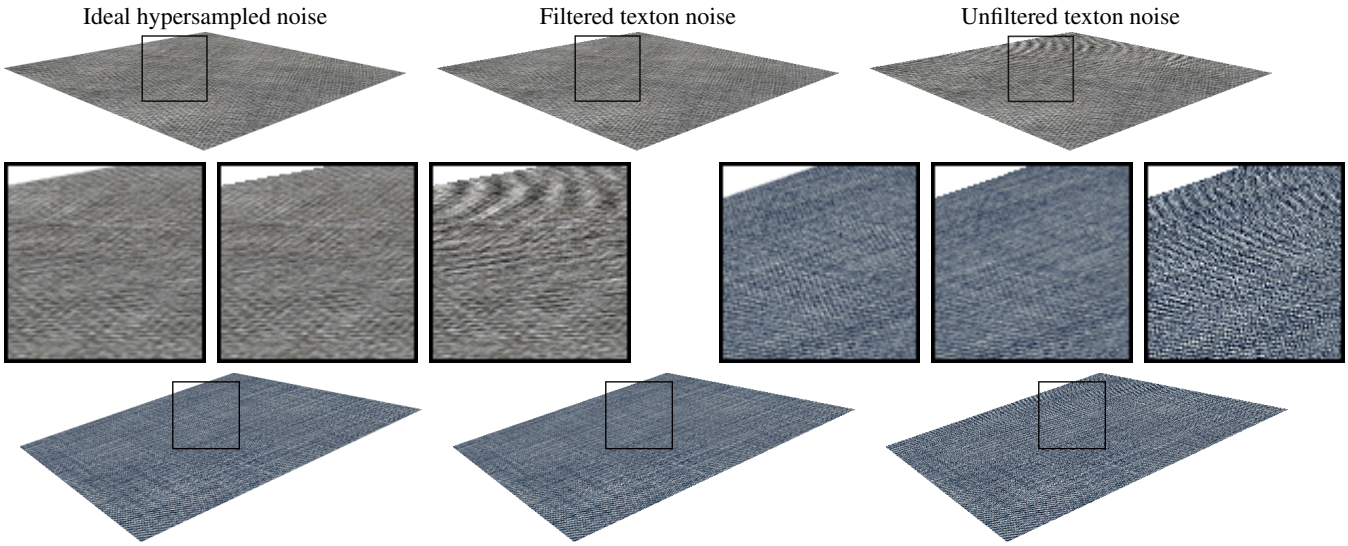

Figure 3: ***Anisotropic filtering:*** *Left: Ideal image obtained by hypersampling with a factor 400 (each pixel is the average of* 20×20 *evaluations), Middle: Texton noise filtered on-the-fly by using MIP-mapping and anisotropic filtering of the texton, Right: Unfiltered texton noise (each pixel is a single evaluation). The middle row displays close-up views delimited by the squares where the three left (resp. right) images correspond to the three top (resp. bottom) inner frames. Observe that the proposed on-the-fly filtering technique gives results which are visually very close to the ideal hypersampled noise.*

has limitations (*e.g.* compute the parameterization, deal with distortion), 2) define a noise directly on the surface using *surface noise* [LLDD09], 3) sampling a solid noise. Solutions 1) and 2) are competitive while solution 3) is complementary to the first two solutions. Indeed, some textures such as paint correspond to 2D textures applied on surfaces while other textures such as wood or stone correspond to 3D solid textures. In what follows we discuss the adaptation of surface noise for texton noise. Note that there is theoretically no obstacle to adapt texton noise to solid noise by example. Since the generation of such 3D texture examples from 2D images is challenging in itself, we left solid texton noise as a promising future work.

Surface noise, as defined by [LLDD09] and originally developed by [Cha07], is an elegant construction that enables to apply a 2D sparse convolution noise of the form

$$f_\lambda(x) = \sum_{x_i \in \Pi^2_\lambda} h(x - x_i),$$

with Π^2_λ a 2D Poisson point process and $h : \mathbb{R}^2 \to \mathbb{R}^d$, $d = 1$ or 3, a (possibly varying) kernel with small support, to a 3D surface $\mathcal{S}$ provided the surface curvature variation scale is larger than the kernel support. To do so, one supposes that at each point $y \in \mathcal{S}$ of the surface a local orthonormal basis (u_1^y, u_2^y, u_3^y) is known where the third basis vector u_3^y is the normal to $\mathcal{S}$ at point $y \in \mathcal{S}$. Additionally, the map $y \mapsto (u_1^y, u_2^y, u_3^y)$ must vary continuously if the surface is continuous (As continuous as possible since e.g. there is no continuous direction field on a sphere...). The main advantage of surface noise is that it does not require surface parameterization and does not involve any topological assumption on the surface $\mathcal{S}$. Suppose that the support of h is the square $[-\frac{r}{2}, \frac{r}{2}]^2$. Then, the surface noise value $f^s_\lambda(y)$ at point y is obtained by projecting into the tangent plane of $\mathcal{S}$ at point y all the points of a 3D Poisson point process Π^3_λ that are at distance less than $\frac{r}{2}$ and attaching to the projected point a kernel h with orientation (u_1^y, u_2^y). This leads to the equation

$$f^s_\lambda(y) = \sum_{x_i \in \Pi^3_\lambda} (1 - \tfrac{2}{r}|\langle y - x_i, u_3^y\rangle|)^+ h(\langle y - x_i, u_1^y\rangle, \langle y - x_i, u_2^y\rangle))$$

where the brackets $\langle \cdot, \cdot \rangle$ stand for the dot product in $\mathbb{R}^3$. The aim of the weight $(1 - \frac{2}{r}|\langle y - x_i, u_3^y\rangle|)^+$ is to favor the points that are closer to the tangent plane and that are more likely to be active for points close to y.

For texton noise synthesis, it is crucial to be able to normalize surface noise. Our main contribution regarding surface noise is to provide the following general formulae (the proof is given in Section A in appendix).

Proposition 3 (Surface noise mean and variance) With the above notation,

$$\mathbb{E}(f^s_\lambda(y)) = \lambda \frac{r}{2} \int_{\mathbb{R}^2} h(z)dz, \quad \mathrm{Var}(f^s_\lambda(y)) = \lambda \frac{r}{3} \int_{\mathbb{R}^2} h^2(z)dz,$$

and if h has support $[-\frac{r}{2}, \frac{r}{2}]^2$, the MNI, *i.e.* the mean number of non zero terms in the sum defining $f^s_\lambda(y)$, is λr^3.

The practical consequence of the above formulae are as follows. Since the surface noise must have the same variance than the usual normalized 2D noise g_λ (2), that is, $\int_{\mathbb{R}^2} h^2(z)dz$, the properly normalized surface noise is

$$g^s_\lambda(y) = \frac{f^s_\lambda(y) - \lambda \frac{r}{2} \int_{\mathbb{R}^2} h(z)dz}{\sqrt{\lambda \frac{r}{3}}} = \frac{f^s_\lambda(y) - \lambda \frac{r}{2} \sum_{k \in S} \alpha(k)}{\sqrt{\lambda \frac{r}{3}}}.$$

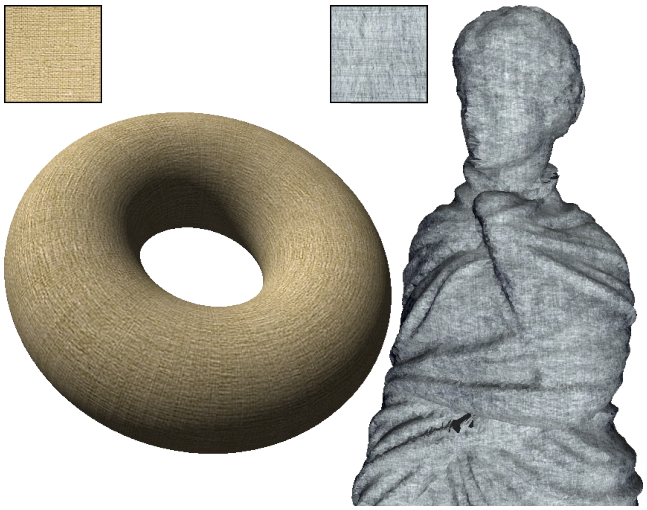

Figure 4: ***Surface noise:*** *Two examples of surface noise with texton noise and the original texture image. Surface noise permits to apply a noise texture on a surface without texture coordinates. This is of special interest for surfaces acquired by 3D scanner which are hard to parameterize due to irregularities (outlier, holes,...) like the example on the right (mesh obtained from a fraction of the Tanagra point set of [DAL*11]).*

Let us observe that due to the weight varying with the distance to the tangent plane, one needs to use a larger MNI to get a texton noise visually close to the theoretical Gaussian texture. Figure 4 illustrates surface noise for texton noise where we used a MNI of 60.

7. Noise Mixing

In this section, we show that texton noise can be used to obtain synthetic textures that progressively mix two texture samples $\boldsymbol{u}$ and $\boldsymbol{v}$. Spatially varying synthesis is an issue of interest for procedural noise [LLD11] but has never been achieved in the context of noise by example where noise spectra are more complex.

Following [XFPA14], two exemplars can be mixed by relying on barycenters of Gaussian texture models for the optimal transport distance. More precisely, for a mixing parameter $\rho \in [0,1]$, the barycenter of the ADSN models associated with the kernels $\boldsymbol{t_u} = \frac{1}{\sqrt{|\Omega|}}(\boldsymbol{u} - \text{mean}(\boldsymbol{u}))$ and $\boldsymbol{t_v} = \frac{1}{\sqrt{|\Omega|}}(\boldsymbol{v} - \text{mean}(\boldsymbol{v}))$ is the ADSN model associated with the kernel $\boldsymbol{\tau}_\rho$ defined by

$$\boldsymbol{\tau}_\rho = (1-\rho)\boldsymbol{\tau}_0 + \rho\boldsymbol{\tau}_1 \quad \text{with} \quad \hat{\boldsymbol{\tau}}_0 = \frac{\hat{\boldsymbol{t}}_{\boldsymbol{u}}^* \hat{\boldsymbol{t}}_{\boldsymbol{v}}}{|\hat{\boldsymbol{t}}_{\boldsymbol{u}}^* \hat{\boldsymbol{t}}_{\boldsymbol{v}}|} \hat{\boldsymbol{t}}_{\boldsymbol{u}} \ , \ \boldsymbol{\tau}_1 = \boldsymbol{t_v}.$$

Once the kernel $\boldsymbol{\tau}_\rho$ is computed, one can derive the corresponding texton interpolation coefficients $\boldsymbol{\alpha}_\rho$ with Algorithm 1. Let us remark that using the same white noise initialization in Algorithm 1 allows to compare the different textons associated with the interpolated models. Our main contribution in noise mixing is to show that these interpolated models can be embedded in shot noise synthesis with spatially varying textons. For that, we consider a partition $D_1, \dots, D_R$ of the synthesis domain corresponding to R different kernel functions $\boldsymbol{h}_1, \dots, \boldsymbol{h}_R$. In order to deal with kernels having non-zero sum, for each point x_j of the Poisson point process we introduce a random weight ε_j with uniform distribution on $\{-1,1\}$. Since we can use either $\boldsymbol{h}$ or $-\boldsymbol{h}$ in the shot noise without affecting the limiting covariance, introducing these random weights is an easy way to force a null expectation in the output random field without changing the texture model. Given the marked Poisson point process Π_λ^M of intensity $\frac{\lambda}{2}$ on $\mathbb{R}^2 \times \{-1,1\}$, we define the mixed shot noise

$$\boldsymbol{f}_\lambda(x) = \sum_{(x_j,\varepsilon_j)\in\Pi_\lambda^M} \sum_{1\leqslant r\leqslant R} \varepsilon_j \boldsymbol{h}_r(x-x_j)\mathbb{1}_{x_j\in D_r},$$

where $\mathbb{1}_{x_j\in D_r} = 1$ if $x_j \in D_r$ and 0 otherwise. This non-stationary shot noise has null expectation, and one can show that $\boldsymbol{g}_\lambda = \frac{\boldsymbol{f}_\lambda}{\sqrt{\lambda}}$ converges to a Gaussian random field [Pap71]. Therefore, one can obtain a progressive mixing of $\boldsymbol{u}$ and $\boldsymbol{v}$ by dividing the synthesis domain into R vertical strips $(D_r)_{1\leqslant r\leqslant R}$ corresponding to the textons $(\boldsymbol{h}_r)_{1\leqslant r\leqslant R}$ associated with $(\boldsymbol{\tau}_{\frac{r-1}{R-1}})_{1\leqslant r\leqslant R}$. The noise mixing is then obtained by sampling $\frac{\boldsymbol{f}_\lambda}{\sqrt{\lambda}}$ and adding a linear interpolation of the mean values of $\boldsymbol{u}$ and $\boldsymbol{v}$.

Such results of noise mixing are shown in Figure 5 with $R = 21$ kernel functions. One can see that this method gives a convincing progressive mixing of the textures $\boldsymbol{u}$ and $\boldsymbol{v}$ and still keeps all the benefits of the texton noise simulation. In the case where $\boldsymbol{u}$ and $\boldsymbol{v}$ belong to the same texture subclass (like the wood textures of the first example), the intermediate texture pieces are likely to appear plausible samples of this texture subclass. Notice that, even if the texton choices are discretized in space, there is no undesirable discontinuity in the output texture because the kernel functions are locally blended. This mixing experiment takes profit of the spot noise flexibility and illustrates the interesting possibilities offered by local variations of the kernel functions.

8. Procedural Evaluation

Since texton noise is a sparse convolution noise with a single kernel having a finite support $[-\frac{r}{2},\frac{r}{2}]^2$, evaluating the noise at a point $x \in \mathbb{R}^2$ simply requires to simulate the Poisson point process on the domain $x + [-\frac{r}{2},\frac{r}{2}]^2$. To do so, we simply use a standard grid-based approach [Wor96, LLDD09]. Let us recall that it consists in partitioning $\mathbb{R}^2$ into grid cells of the form $C_k = ak + [0,a)^2$, $k \in \mathbb{Z}^2$, define a pseudo-random number generator PRNG_k for each cell C_k by using a seed depending on the coordinate of k, and simulate the Poisson process within C_k using PRNG_k. Then, to evaluate the noise at x one simulates on-the-fly the Poisson process on the cells that intersects $x + [-\frac{r}{2},\frac{r}{2}]^2$. The cell size a is generally taken to be equal to r [LLDD09, LLC*10]. Since $x + [-\frac{r}{2},\frac{r}{2}]^2$ always intersects 4 grid cells of size r, this results in evaluating in average $4\lambda r^2 = 4$ MNI points while only MNI points actually contribute to x in average. Using cells smaller than r limits the number of unused points but also increases the number of Poisson variate simulations for the number of points within a cell. A cell size $a = 0.6r$ was found to be a good compromise for our implementation. This acceleration is also valid for the 3D Poisson point process used for surface noise.

We experimented correlation issues with the linear congruential generator used in [GLLD12]. Following [Ree13], we used Xorshift

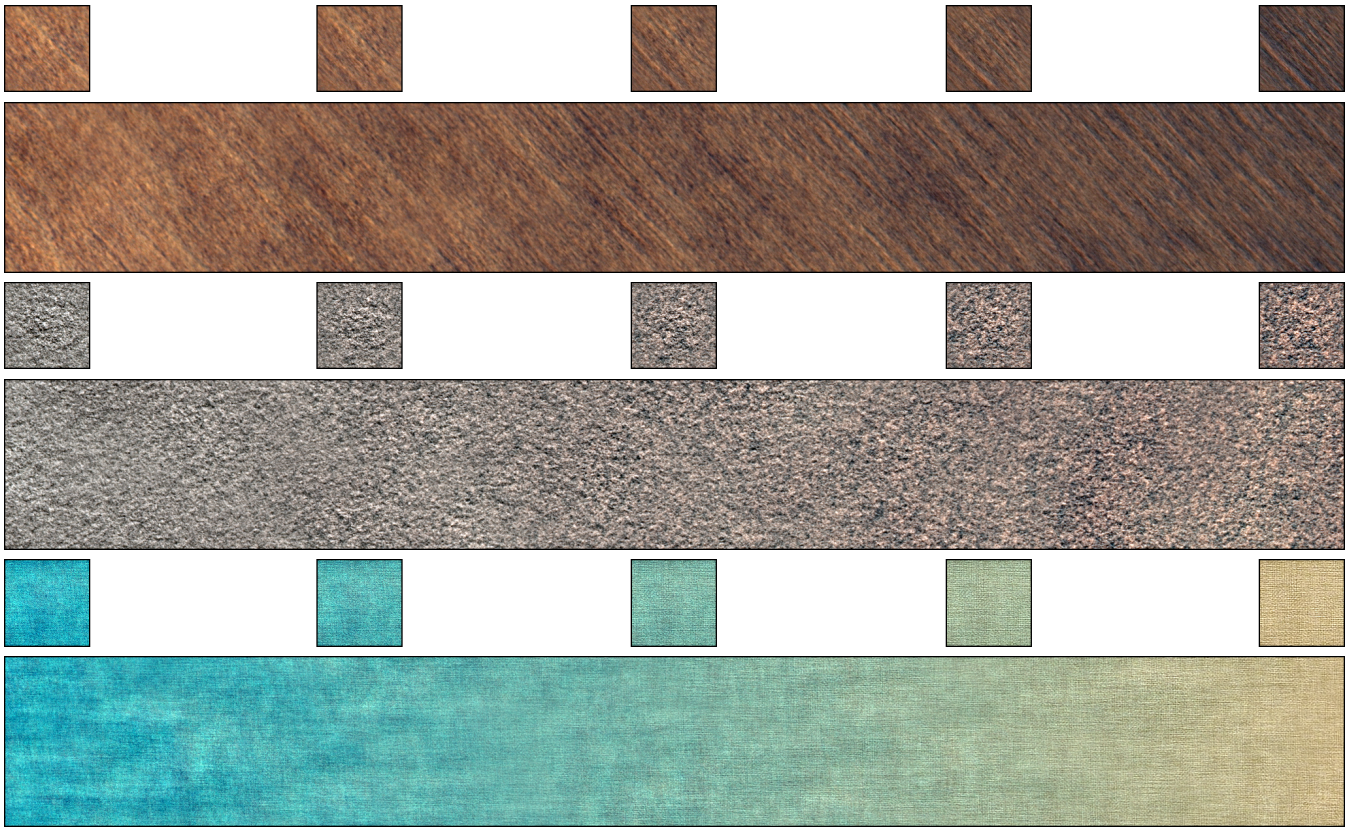

Figure 5: ***Noise mixing:*** *We show three examples of progressive texture mixing obtained by spot noise synthesis with varying textons. The different textons are obtained as optimal transport barycenters between Gaussian texture models. For each example, in the first row, we show five textons (with respective mixing parameter* $\rho = 0, 0.25, 0.5, 0.75, 1$*) among the 21 involved in the synthesis result shown in the second row. Notice that the texture variation appears continuous even if the texton choices are discretized.*

RNGs [Mar03] and processed the grid seed with a hash function. For the Poisson variate generation, since we only use high intensity Poisson processes, we used the same Gaussian approximation as [GLLD12].

9. Results

Implementation and reproducibility We provide as supplementary material: 1) Matlab source codes for texton computation (see Algorithm 1); 2) an OpenGL implementation for texton noise evaluation (for which we used parts of the implementation of [LD11]); 3) all the input textures and texton files used to produce the figures.

Mean number of impacts Except for Figure 2, all the results are obtained with a MNI of 30 for 2D texton noise and a MNI of 60 for surface noise.

Texton support Figure 6 illustrates the influence of the texton support size regarding texton noise quality and spectral approximation. The quality of the noise always increases with the size of the texton, but for some textures the quality improvement is negligible once the texton is large enough. Hence the support of the texton can be seen as a trade-off between texture detail and memory storage, but for some textures with simple small patterns, increasing the texton size does not improve the noise visual quality (see e.g. the first column of Figure 7). Let us stress that the support size does not influence the performance of the texton noise evaluation (30 texture fetches by evaluation points on average) as discussed below.

Synthesis results Figure 7 shows that any input Gaussian texture can be visually reproduced with a texton noise (see also Figures 1, and 4). We compare texton noise with Gabor noise by example [GLLD12] and Local Random Phase noise [GSV*14] in the supplementary material. This comparison shows that texton noise allows to synthesize Gaussian textures in the most precise and efficient way, while only LRP allows to synthesize more structured textures. Let us recall that texton noise is by construction limited to Gaussian textures (e.g. see the loss of details in the last example of Figure 7). Although let us add that texton noise outputs do not contain any repetition contrary to most tiling procedures (see the supplementary material for a comparison with the patch-based image quilting algorithm [EF01]).

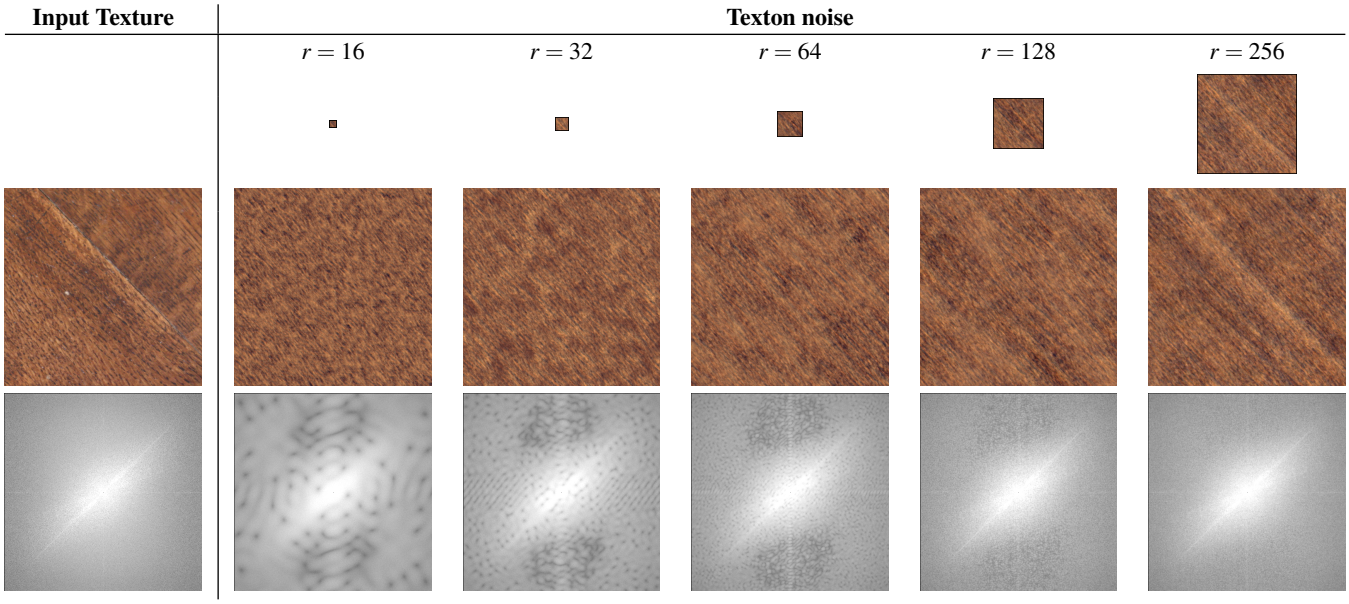

Figure 6: ***Influence of the texton support:*** *The support of the texton is a trade-off between texture detail and memory storage. Left column: Original image (512×512) and its power spectrum. Subsequent columns: Texton with support $r \times r$, corresponding texton noise, estimated power spectrum obtained by averaging 10 independent texton noise realizations. Remark that the power spectrum is closer and closer to the original one as the support size increases.*

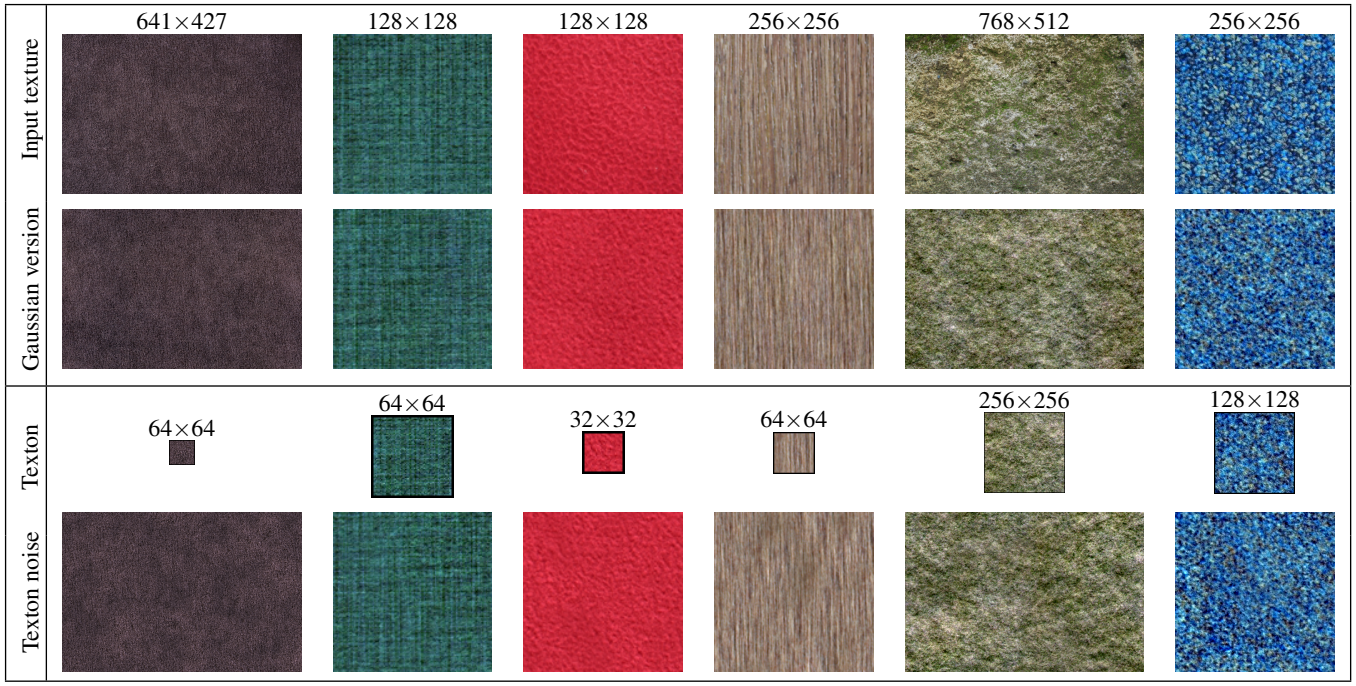

Figure 7: ***Synthesis results:*** *Texton noise enables to reproduce any Gaussian texture. First row: Input texture (with size); Second row: Gaussian version of the texture (see Section 3.1); Third row: Computed texton (with size); Fourth row: Texton noise.*

Output size	Texton 64×64	Texton 256×256
1024×1024	204 fps / 4.9 ms	189 fps / 5.3 ms
HD (1920×1080)	102 fps / 9.8 ms	98 fps / 10.1 ms
4K (3840×2160)	27 fps / 36.9 ms	26 fps / 38.9 ms

Table 1: *Performance of 2D texton noise evaluation (fps stands for frame per second and ms for millisecond).*

Performance One important feature of texton noise is that its evaluation speed does not depend on the texture spectrum complexity contrary to existing noise by example approaches [GLLD12, GSV*14]. We reproduce in Table 1 the performance of 2D texton noise evaluation run on a Quadro K5000 (1536 Cuda cores). These figures show that texton noise allows for real time evaluation of a full HD screen. These performances are two orders of magnitude faster than Gabor noise by example [GLLD12] and one order of magnitude faster than LRP noise [GSV*14] (these two references reported performance for small 128×128 images; extrapolation and weighting with the CUDA cores number of used graphics cards gives the following order of magnitude for 1024×1024 images: 2 fps for Gabor noise by example and 20 fps for LRP noise). One can also observe that the texton size influences only slightly the evaluation (texture fetches are presumably slower).

Regarding texton computation, our Matlab implementation of Algorithm 1 runs in 0.5 sec. for an RGB 128×128 image and in about 5 sec. for a 768×512 image. In comparison, the Gabor noise by example parameter computation takes about 2 min. for a 128×128 image. Performance for parameter computation of LRP noise has not been reported.

10. Conclusion

This paper introduced a new noise model that allows for the reproduction of any Gaussian texture image. Since the texton summarizes the whole content of an image (or a prescribed power spectrum), texton noise produces visually good results with only 30 texture fetches per fragment in average. This makes texton noise the most efficient technique for noise by example, with a gain of one or two orders of magnitude in comparison with existing algorithms. In addition, texton noise is easily filtered on-the-fly by simply filtering the texton texture with standard MIP-mapping and anisotropic filtering while existing algorithms use involved frequency attenuation procedures with non-negligible additional cost. We also demonstrated that texton noise allows for noise mixing, which can potentially be of interest for texture artists.

We see two promising future work directions regarding texton noise. The first is to expand texton noise to 3D solid noise as well as dynamic textures. The second more challenging direction is to tackle the main limitation of texton noise (that is, being limited to Gaussian textures) by proposing new similar models that enable to generate more structured textures as done by local random phase noise [GSV*14].

References

[CD05] COOK R. L., DEROSE T.: Wavelet noise. In *SIGGRAPH '05* (New York, NY, USA, 2005), ACM, pp. 803–811. doi:10.1145/1186822.1073264. 3, 6

[Cha07] CHAINAIS P.: Infinitely divisible cascades to model the statistics of natural images. *IEEE Trans. Pattern Anal. Mach. Intell. 29*, 12 (2007), 2105–2119. doi:10.1109/TPAMI.2007.1113. 7

[CSHD03] COHEN M., SHADE J., HILLER S., DEUSSEN O.: Wang Tiles for image and texture generation. In *SIGGRAPH '03* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 287–294. URL: http://doi.acm.org/10.1145/1201775.882265, doi:10.1145/1201775.882265. 3

[DAL*11] DIGNE J., AUDFRAY N., LARTIGUE C., MEHDI-SOUZANI C., MOREL J.-M.: Farman institute 3d point sets - high precision 3d data sets. *Image Processing On Line 1* (2011). doi:10.5201/ipol.2011.dalmm_ps. 8

[DCWS03] DORETTO G., CHIUSO A., WU Y., SOATTO S.: Dynamic textures. *International Journal of Computer Vision 51*, 2 (2003), 91–109. doi:10.1023/A:1021669406132. 2

[DMR12] DESOLNEUX A., MOISAN L., RONSIN S.: A compact representation of random phase and Gaussian textures. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (2012), pp. 1381–1384. 2

[DMR15] DESOLNEUX A., MOISAN L., RONSIN S.: A texton for random phase and Gaussian textures. 2015. 6

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH '01* (New York, NY, USA, 2001), ACM, pp. 341–346. doi:10.1145/383259.383296. 9

[GDS10] GILET G., DISCHLER J.-M., SOLER L.: Procedural descriptions of anisotropic noisy textures by example. In *Eurographics (Short)* (2010). 3

[GGM11] GALERNE B., GOUSSEAU Y., MOREL J.-M.: Random phase textures: Theory and synthesis. *IEEE Trans. Image Process. 20*, 1 (2011), 257 – 267. doi:10.1109/TIP.2010.2052822. 2, 3

[Gla04] GLANVILLE R.: Texture bombing. In *GPU Gems*, Fernando R., (Ed.). Addison-Wesley, 2004, pp. 323–338. 3

[GLLD12] GALERNE B., LAGAE A., LEFEBVRE S., DRETTAKIS G.: Gabor noise by example. *ACM Trans. Graph. 31*, 4 (jul 2012), 73:1–73:9. doi:10.1145/2185520.2185569. 2, 3, 6, 8, 9, 11

[GLM14] GALERNE B., LECLAIRE A., MOISAN L.: A texton for fast and flexible Gaussian texture synthesis. In *Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)* (2014), pp. 1686–1690. 2, 4, 5, 6

[GSV*14] GILET G., SAUVAGE B., VANHOEY K., DISCHLER J.-M., GHAZANFARPOUR D.: Local random-phase noise for procedural texturing. *ACM Trans. Graph. (Proceedings of ACM SIGGRAPH ASIA 2014) 33*, 6 (nov 2014), 195:1–195:11. doi:10.1145/2661229.2661249. 2, 3, 6, 9, 11

[GSX00] GUO B., SHUM H., XU Y.: Chaos mosaic: Fast and memory efficient texture synthesis. *Microsoft research paper MSR-TR-2000-32* (2000). 3

[GZD08] GOLDBERG A., ZWICKER M., DURAND F.: Anisotropic noise. In *SIGGRAPH '08* (New York, NY, USA, 2008), ACM, pp. 1–8. doi:10.1145/1399504.1360653. 6

[Kin93] KINGMAN J. F. C.: *Poisson Processes*. Oxford Studies in Probability. Oxford University Press, 1993. 12

[LD11] LAGAE A., DRETTAKIS G.: Filtering solid Gabor noise. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2011) 30*, 4 (July 2011), 51:1–51:6. doi:10.1145/1964921.1964946. 9

[Lew84] LEWIS J.-P.: Texture synthesis for digital painting. In *SIGGRAPH '84* (New York, NY, USA, 1984), ACM, pp. 245–252. doi:10.1145/800031.808605. 2, 4

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Texture sprites: Texture elements splatted on surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), ACM, pp. 163–170. doi:10.1145/1053427.1053454. 3

[LKF*08] LAGAE A., KAPLAN C., FU C., OSTROMOUKHOV V., DEUSSEN O.: Tile-based methods for interactive applications. In *ACM SIGGRAPH 2008 classes* (2008), ACM, p. 93. URL: http://dl.acm.org/citation.cfm?id=1401254. 3

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., EBERT D., LEWIS J., PERLIN K., ZWICKER M.: A survey of procedural noise functions. *Computer Graphics Forum 29*, 8 (December 2010), 2579–2600. doi:10.1111/j.1467-8659.2010.01827.x. 1, 2, 6, 8

[LLD11] LAGAE A., LEFEBVRE S., DUTRÉ P.: Improving Gabor Noise. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1096–1107. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5620898. 8

[LLDD09] LAGAE A., LEFEBVRE S., DRETTAKIS G., DUTRÉ P.: Procedural noise using sparse Gabor convolution. *SIGGRAPH '09 28*, 3 (August 2009). doi:10.1145/1576246.1531360. 3, 4, 6, 7, 8

[LN03] LEFEBVRE S., NEYRET F.: Pattern Based Procedural Textures. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (2003), I3D '03, ACM, pp. 203–212. doi:10.1145/641480.641518. 3

[LVLD10] LAGAE A., VANGORP P., LENAERTS T., DUTRÉ P.: Procedural isotropic stochastic textures by example. *Computers & Graphics (Special issue on Procedural Methods in Computer Graphics)* (2010). doi:10.1016/j.cag.2010.05.004. 2, 3, 6

[Mar03] MARSAGLIA G.: Xorshift rngs. *Journal of Statistical Software 08*, i14 (2003). URL: http://EconPapers.repec.org/RePEc:jss:jstsof:08:i14. 9

[Mat68] MATHERON G.: *Modèle séquentiel de partition aléatoire*. Tech. rep., CMM, 1968. 3

[Moi11] MOISAN L.: Periodic plus smooth image decomposition. *J. Math. Imag. Vis. 39* (2011), 161–179. doi:10.1007/s10851-010-0227-1. 5

[Pap71] PAPOULIS A.: High density shot noise and Gaussianity. *J. Appl. Probab. 8*, 1 (1971), 118–127. URL: http://www.jstor.org/stable/3211842. 4, 8

[Per85] PERLIN K.: An image synthesizer. In *SIGGRAPH '85* (New York, NY, USA, 1985), ACM, pp. 287–296. doi:10.1145/280811.280986. 1, 2

[Ree13] REED N.: Quick and easy GPU random numbers in D3D11. Blogpost, http://www.reedbeta.com/blog/2013/01/12/quick-and-easy-gpu-random-numbers-in-d3d11/, Jan. 2013. 8

[Ric77] RICE J.: On generalized shot noise. *Adv. Appl. Probab. 9* (1977), 553–565. URL: http://www.jstor.org/stable/1426114. 4

[SCSI08] SIMAKOV D., CASPI Y., SHECHTMAN E., IRANI M.: Summarizing visual data using bidirectional similarity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2008), IEEE, pp. 1–8. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4587842. 2

[VSLD13] VANHOEY K., SAUVAGE B., LARUE F., DISCHLER J.-M.: On-the-fly multi-scale infinite texturing from example. *ACM Trans. Graph. (Proc. of ACM SIGGRAPH Asia 2013) 32*, 6 (nov 2013), 208:1–208:10. doi:10.1145/2508363.2508383. 3

[vW91] VAN WIJK J. J.: Spot noise texture synthesis for data visualization. In *SIGGRAPH '91* (New York, NY, USA, 1991), ACM, pp. 309–318. doi:10.1145/122718.122751. 2, 4

[Wei04] WEI L.: Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2004), ACM, pp. 55–63. URL: http://dl.acm.org/citation.cfm?id=1058138. 3

[WHZ*08] WEI L.-Y., HAN J., ZHOU K., BAO H., GUO B., SHUM H.-Y.: Inverse texture synthesis. In *ACM Transactions on Graphics* (2008), vol. 27. URL: http://dl.acm.org/citation.cfm?id=1360651. 2

[Wor96] WORLEY S.: A cellular texture basis function. In *SIGGRAPH '96* (1996), ACM, pp. 291–294. doi:10.1145/237170.237267. 8

[XFPA14] XIA G.-S., FERRADANS S., PEYRÉ G., AUJOL J.-F.: Synthesizing and mixing stationary Gaussian texture models. *SIAM J. on Imaging Science 8*, 1 (2014), 476–508. 2, 3, 8

Appendix A: Proof of Proposition 3

As one can see from its expression (6), $f^s_\lambda(y)$ is a sum of the values of a function over a 3D Poisson point process. Thus its mean and variance are easily computed thanks to Campbell theorem [Kin93]. Using the (orthonormal) change of variable $(t,z_1,z_2) = (\langle y-x,u_3^y\rangle, \langle y-x,u_1^y\rangle, \langle y-x,u_2^y\rangle)$,

$$\begin{aligned}\mathbb{E}(f^s_\lambda(y)) &= \lambda \int_{\mathbb{R}^3} (1-\tfrac{2}{r}|\langle y-x_i,u_3^y\rangle|)^+ h((\langle y-x,u_1^y\rangle,\langle y-x,u_2^y\rangle))dx \\ &= \lambda \int_{\mathbb{R}} (1-\tfrac{2}{r}|t|)^+ dt \int_{\mathbb{R}^2} h(z)dz = \lambda \frac{r}{2}\int_{\mathbb{R}^2} h(z)dz,\end{aligned}$$

and similarly,

$$\begin{aligned}&\mathrm{Var}(f^s_\lambda(y)) \\ &= \lambda \int_{\mathbb{R}^3} \Big((1-\tfrac{2}{r}|\langle y-x_i,u_3^y\rangle|)^+ h((\langle y-x,u_1^y\rangle,\langle y-x,u_2^y\rangle))\Big)^2 dx \\ &= \lambda \int_{\mathbb{R}} \Big((1-\tfrac{2}{r}|t|)^+\Big)^2 dt \int_{\mathbb{R}^2} h^2(z)dz = \lambda\frac{r}{3}\int_{\mathbb{R}^2} h^2(z)dz.\end{aligned}$$

Finally, a point x_i corresponds to a non-zero term $(1-\frac{2}{r}|\langle y-x_i,u_3^y\rangle|)^+ h((\langle y-x_i,u_1^y\rangle,\langle y-x_i,u_2^y\rangle))$ if it belongs to the cube of side length r, center y and axis (u_1^y,u_2^y,u_3^y). This set has volume r^3 and thus the MNI is λr^3.