# Local spot noise for procedural surface details synthesis

Arthur Cavalier, Guillaume Gilet, Djamchid Ghazanfarpour

Computers & Graphics (2019)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

# Local Spot Noise for Procedural Surface Details Synthesis

Arthur Cavalier, Guillaume Gilet, Djamchid Ghazanfarpour

*Univ. Limoges, CNRS, XLIM, UMR 7252, F-87000 Limoges, France*

ARTICLE INFO

*Article history*:
Received 27 May 2019
Accepted 12 October 2019

*Keywords:* Procedural texturing, procedural noise, antialiasing, image synthesis

ABSTRACT

To deal with the increasing demand for complex visual details in virtual worlds, procedural methods for content authoring are an expanding field in Computer Graphics. Focusing on on-the-fly texture generation, we present in this paper a content authoring process based on Locally Controlled Spot Noise. Through the control of both the impulses distribution and the spatially-defined kernel, this process can cover a wide range of appearances. In this context, we introduce a new kernel formulation that provides an efficient anisotropic filtering of the generated texture. Furthermore, our method allows users to interactively create the desired appearance by controlling both albedo and meso-geometry of the underlying surface, tackling on-the-fly normal map generation. Our method can be used as an artist friendly tool to model high-quality surface details with direct control over the final appearance in real-time.



## 1. Introduction

Procedural content generation is a widely studied field in Computer Graphics to alleviate the problem of complex scene authoring. We focus in this work on interactive procedural texturing, as it is a well known method to synthesize a high amount of details onto surfaces during rendering while remaining memory compact and suited to modern GPU implementations. The goal of this approach is to fill a target surface with a desired pattern resulting from the evaluation of a procedural function. Several methods, such as noises or patch based method [1, 2, 3], can be devised to generate complex patterns. To overcome the traditional difficulty of correlating the targeted visual appearance with arbitrary function parameters, most recent works focus on texture generation by precomputing the parameters of the procedural function using a by-example approach. However, results are highly dependent on the content of the input example and control over the final result remains unintuitive. It often requires editing the discrete input or manually tweaking the function parameters. In particular, finely controlling the visual variety of the final result remains a challenging task for most methods.

Another approach to solve the difficulty of appearance / procedural function parameters matching is to provide a definition relying on intuitive parameters. Spot noise, as introduced by van Wijk [4], relies on a small user-defined kernel distributed over the target surface and presents easy to understand parameters. As shown by Pavie et al. [5], locally controlled spot noise yields fine controls over a wide range of appearances by controlling both kernel function and distribution. However, this method did not address filtering issues which are essential to achieve high quality visual results during rendering of complex scenes. Furthermore, most visually appealing appearances exhibit subtle variations in shading and are not limited to albedo. Thus, procedural methods generating details should consider surface normal as well as surface color.

This paper presents how to overcome these limitations by extending the locally controlled spot noise formalization. Our main contributions are :

- A new kernel formulation relying on a lighter parameteri-

*e-mail:* arthur.cavalier@unilim.fr (Arthur Cavalier), guillaume.gilet@unilim.fr (Guillaume Gilet), djamchid.ghazanfarpour@unilim.fr (Djamchid Ghazanfarpour)

zation while preserving intuitive geometric parameters.

- An anisotropic filtering scheme to achieve high-quality results during rendering.
- An analytic normal map generation process while maintaining user control and preventing aliasing issues.

The remaining parts of the paper is organized as follows: in the next section we propose a brief overview of procedural patterns and noise generation methods. Before presenting our new kernel formulation and its filtering capabilities, we recall the previously introduced definition [5]. Finally, we discuss various applications and limitations of our spot noise model such as stylized rendering and non-stationary texture generation.

## 2. Related Work

Tackling the creation of procedural patterns over an arbitrary surface is commonly achieved by filling the target surface space with smaller primitives. These primitives could be discrete, using small texture elements (or patches), or procedurally generated. Patterns are usually distributed onto surface space using a distribution function ranging from totally periodic and regular to purely random, according to the desired regularity of the target.

### *2.1. Patch based methods and Tiling*

Patch-based methods are commonly used to create structured or semi-structured procedural textures, leveraging the inherent visual structure of the discrete patches. In such approach, the procedural pattern is evaluated by regularly tiling the target surface with patches [6, 7], or by precomputing Wang tiles[8, 9] to fill the synthesis domain. Patches are randomly arranged to break repetitions, but results may lack of details variety : the same tiles / patches (i.e. rigorously identical contents) are repeated over and over again even for irregular textures. This is particularly visible in case of entirely periodic tiling, where patches with identifiable visual characteristics can cause unpleasant repetitions. Attenuating these repetition artifacts can be done by increasing the number of patches, at the cost of memory consumption, or by carefully choosing patches with very similar visual characteristics [10], thus reducing the variety of the result.

To break alignment effects or add more randomness in the result, patches can be distributed over the target surface according to a procedural distribution function to create an infinite set of random positions. Point jittering is often used as distribution function for its simplicity and evaluation speed [11]. However, it does not take into account spatial dependencies (distance threshold between objects) so distributed objects may overlap, introducing the need for a blending operator. Direct Stochastic tiling [12] can produce some distance dependencies, to create for instance an infinite set of Poisson-disks, but still requires some tiles to be precomputed and stored. Rectilinear tessellation of the surface can be considered [13] to improve point jittering using a different instance of an object per cell with a random position computed on-the-fly. Fully procedural semi-structured pattern can be produced using both procedural object definitions and procedural distribution functions, but very few techniques propose to extract such objects directly from an input sample.

### *2.2. Procedural noise*

Procedurals noises have been widely studied in the past near-four-decades. We refer to the survey of Lagae et al. [1] for a more thorough overview of noise-based procedural texturing. Two types of approaches are commonly considered: lattice gradient noises and sparse convolution noises. The former are based on the interpolation of randomly oriented gradients evaluated on the vertices of a regular grid. Sparse convolution noises are based on the convolution of randomly distributed impulses by a spatial filter function (kernel). As uniform random distribution processes result in white noise in the frequency domain, control over sparse convolution noises is achieved by controlling the kernel function. As first introduced by Lewis [14, 15, 16], convolution kernels were originally designed in the spectral domain. Several parameterization were proposed, such as Gabor [17] or sinc [18] kernels. Such kernel functions have well-known properties in the spectral domain, can be evaluated in the spatial domain at a low computational cost and thus provide efficient control on the final appearance of the noise process.

Nonetheless, textures generated by sparse convolution methods suffer from loss of contrast as highlighted by Neyret and Heitz [19]. They introduced the spectrum of variance as a more relevant tool to remove the constrast oscillation issue than the Power Spectral Density (PSD). It was recently investigated by Tavernier et al. [20] who studied various components of Gabor noise [17] to optimize and normalize the evaluation process.

However, controlling noise through the spectral definition of a kernel is quite non-intuitive for artists and often requires a time-consuming trial and error process. To alleviate this, a first solution is to use some spatial informations to influence spectral parameters. Charpenay et al. [21] studied how to influence Gabor noise parameters through the use of control maps. Another solution is to automatically extract the parameters of the kernel functions through spectral analysis of an input exemplar. The main idea of these by-example methods relies on the fact that, for a limited class of unstructured textures, a noise process having the equivalent PSD of an input exemplar will have the same visual statistical properties and generate a visually similar result. To achieve this, an approximated coverage of the target PSD is computed using the spectral definition of the convolution kernels, like Gabor [17, 22] or sinc kernels [18]. As the convolution kernel reproducing the entire target PSD can be quite complex, these methods lower the computational complexity of the noise process by randomly choosing each convolution kernel as a subset of the ideal kernel, relying on the random distribution of impulses to approximate the result. Galerne et al. [23] introduced the Texton Noise to correct this approximation, by using a single discrete precomputed image as kernel function. This image is the spatial definition of the ideal kernel offering the best approximation of the target PSD.

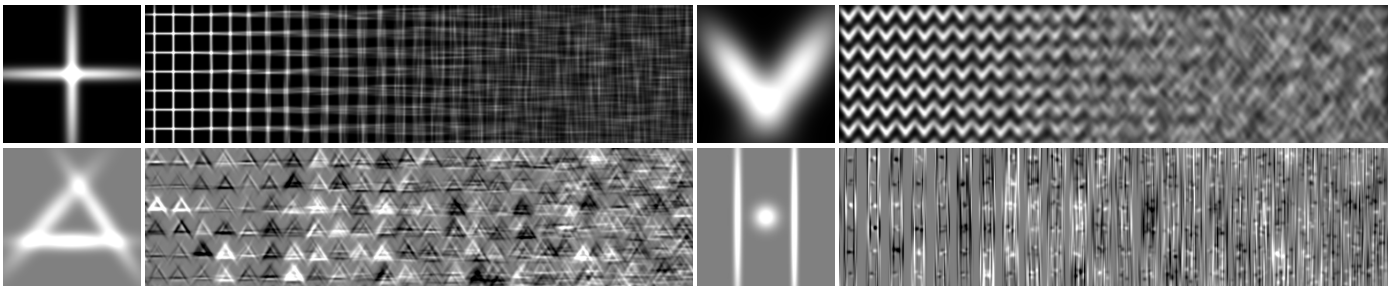


Fig. 1: Pavie et al. [5] proposed a spot noise with a fully controllable kernel, modelled by a sum of Gaussian functions. By constraining distribution, their method can generate various types of patterns, from stochastic to near-regular kernel distribution, changing the final appearance of the texture.

This method, relying for each pixel on 30 texture fetches of this shifted kernel, enables fast evaluation of a by-example random phase noise.

As these methods tackle only textures solely defined by their power spectrum, Gilet et al. [24] proposed to store some phase information of the input exemplar obtained during the spectral analysis and re-introduce them during noise evaluation. By using complex kernels with fixed phase information convoluted with regularly distributed impulses, some structural patterns of the input example can be reproduced, thus widening the range of by-example based procedural methods. Nonetheless, by fixing phase information, this method introduces a periodic structural component of the example. Authors used spatial deformations and turbulence on the output to break periodicity. Instead of using phase information to reproduced the input example structure, Guingo et al. [25] proposed to separately model the structural component by using spatial methods like patch-based methods and the stochastic component with random phase noises.

As stated above, by-example noise textures synthesis are a well suited method for fast texture synthesis without heavy memory consumption, though these methods, even Texton Noise [23], are too time consuming during evaluation in order to be used in critical real-time applications. Recently, Heitz and Neyret [26] introduced an efficient method for computing a by-example texture on-the-fly. By combining Patch-Based texture synthesis and triangle lattice evaluation, this noise requires computing and blending of only three contributions per pixels. This method proposed a blending scheme based on a precomputed "gaussianized" version of the input texture and a variance-preserving blending of three contributions computed on a triangle grid. Yielding high quality fast texturing, this method nonetheless has the same drawbacks of patch-based methods and works mostly for tile-able stochastic input textures. Finally, by rewriting Gabor noise as a single sine wave using a sum of phasors and studying its spectrum of variance, Tricard et al. [27] addressed the contrast oscillation problem and introduced a new tool for pattern synthesis, providing fine controls over the final appearance by editing the profile function. However, the generated patterns may contain visual singularities and the method does not tackle filtering issues.

Another method to solve direct visual control on the final result of the noise process is to define kernels in the spatial domain. Van Wijk [4] introduced a Spot Noise method relying on small user defined kernels convolved with random impulses, providing an intuitive noise process exhibiting micro-structured patterns. To increase user control on structured textures generation process, Pavie et al. [5] further extended this work by defining kernels as a sum of Gaussian functions and controlling the distribution of impulses to reintroduce large scale alignment of micro-structured patterns.

However, this previous work has some limitations, as it does not address filtering issues, which are relevant for high quality texture synthesis, and remains limited to the generation of color information. In this paper, we propose several improvements of this method such as anisotropic filtering, a more efficient definition of the kernel function and the conjoint generation of normal maps.

## 3. Locally controlled spot noise

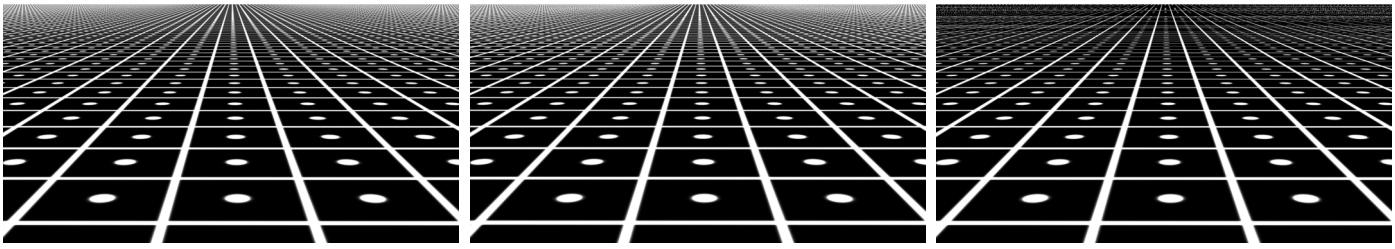
As presented by van Wijk [4], the spot noise model consists in the summation of the same kernel at random positions in texture space. This model can produce a large variety of textures by changing the input kernel, as shown in Fig.1. The Locally Controlled Spot Noise (or LCSN), introduced by Pavie et al. [5], consists in a spot noise formulation based on a kernel defined as a sum of elliptical Gaussian functions. This definition provides intuitive control of the final texture, as each kernel can be explicitly authored by setting up the corresponding Gaussian functions using simple geometric controls (translation, rotation and scaling). Finally, control over large scale structures and the overall appearance of the result can be achieved by controlling the distribution of impulses. We recall the spot noise model as a sum of $M$ kernels weighted by $w_i$ evaluated at impulses $\mathbf{p_i}$

$$\mathrm{lcsn}(\mathbf{p}) = \sum_{i=1}^{M} w_i k(\mathbf{p} - \mathbf{p}_i) \tag{1}$$

where $\mathbf{p}$ is expressed in homogeneous coordinates for point $\mathbf{x}$ in texture space (i.e. $\mathbf{p} = (\mathbf{x}, 1)^T$ ). Each kernel function $k$ is defined as a sum of $N$ elliptical $D$-dimensional Gaussian functions with arbitrary orientation and scale defined by:

$$k(\mathbf{p}) = \sum_{j=1}^{N} \lambda_j e^{-\frac{1}{2}\mathbf{p}^T \mathbf{V}_j^{-1} \mathbf{p}} \tag{2}$$

where $\lambda$ is the Gaussian magnitude and $\mathbf{V_j}$ is a $(D+1)\times(D+1)$ matrix such that $\mathbf{V_j}^{-1} = (\mathbf{M_j R_j S_j})^{-T}(\mathbf{M_j R_j S_j})^{-1}$. Here $\mathbf{M_j}$,$\mathbf{R_j}$



Ours (1 spp) Reference Pavie et al. (1 spp)

Fig. 2: We compare our anisotropic filtering with Pavie et al. method. Regular patterns are usually prone to aliasing issues. By analytically prefiltering the spot noise with our kernel formulation, we achieve high quality results matching the brute force reference.

and $\mathbf{S_j}$ are respectively shift, rotation and scaling matrices, using homogeneous coordinates. To achieve structured patterns, LCSN relies on an uniform distribution of impulses constrained by a Kronecker delta function $\delta\left(\xi(\mathbf{p_j}) < d(\mathbf{p_j})\right)$ where $d$ is a scalar field represented by a probability. This parameter allows the user to control the density of impulses in given regions and can be defined using an arbitrary discrete or analytic function.

In practice, this procedural function is built on-top of a regular grid for acceleration purposes. The finite impulses process is generally a constrained jittered sampling of kernel shots using a by-cell seeding strategy to ensure continuity. The evaluation procedure consists in a sum of each kernel from the current cell and the neighboring ones computed at the considered pixel/fragment, similarly to sparse convolution methods. By definition, the main bottleneck of this spot noise formulation is the kernel complexity (i.e. number of Gaussians used) and the number of impulses chosen.

## 4. Improved kernel formulation

As seen in the previous section, Pavie et al. [5] uses a double-sum formulation for their spot noise, as each kernel is the sum of multiple Gaussian functions. In the paper, the authors do not discuss other types of Gaussian function parameterizations.

The need of homogeneous coordinates can easily be removed while maintaining a purely geometric parametrization. The Gaussian "shape" is independent of the translation, and directly depends on the rotation and the scale information. A common unnormalized Gaussian function with a scalar magnitude $\lambda$, a mean vector $\mu$ and a covariance matrix $\mathbf{\Sigma}$ can be used as a substitute. A Gaussian kernel can thus be defined as :

$$g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) = \lambda e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)} \tag{3}$$

yielding :

$$\mathrm{lcsn}(\mathbf{x}) = \sum_{i=1}^{M} w_i \sum_{j=1}^{N} g(\mathbf{x} - \mathbf{x_i}; \lambda_j, \mu_j, \mathbf{\Sigma_j}) \tag{4}$$

The iso-contour of the kernel is still determined by $(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \mu)$, but does not rely anymore on homogeneous coordinates, keeping the dimension to $\mathbf{D}$. The chosen covariance matrix $\mathbf{\Sigma_j}$ can be expressed with a rotation matrix $\mathbf{R_j}$ and a scale matrix $\mathbf{S_j}$, with the relation $\mathbf{\Sigma_j} = \mathbf{R_j S_j S_j R_j}^T$. It enables the user to rely on purely geometric parameters, a rotation angle, a scale vector and a shift vector.

As stated in Section 3, the complexity of the method is directly dependent of the number of Gaussians used to construct the kernel and the number of kernel shots. By changing the parameterization, we slightly enhance the performance of the spot noise model. This is further discussed in Section 7.

## 5. Anisotropic Filtering

In their first work, Pavie et al. [5] did not take into account filtering for their 2D noise. However, they did consider it when they extended their formulation in 3D ellipsoids for their shell textures synthesis [28]. To prefilter our new kernel formulation, we model our pixel footprint by a Gaussian function similar to Heckbert [29]. A Gaussian pixel footprint is a rather standard choice in the literature as it often provides closed-form solutions to computations with the appropriate kernel as in [17]. In texture space, this results in a normalized Gaussian centered in the projected pixel, without shift, using the Jacobian matrix $\mathbf{J}$ of the texture mapping as a covariance matrix and $\sigma$ as the width of the pixel footprint in image space (i.e. $\sigma = 0.5$).

$$k_P(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\mathbf{J}\mathbf{J}^T|}} e^{-\frac{1}{2}\left[\mathbf{x}^T(\mathbf{J}\mathbf{J}^T)^{-1}\mathbf{x}\right]} \tag{5}$$

with

$$\mathbf{J} = \sigma \begin{pmatrix} \frac{du}{dx} & \frac{du}{dy} \\ \frac{dv}{dx} & \frac{dv}{dy} \end{pmatrix}$$

In practice the Jacobian matrix is built using screen-space derivatives, in the rasterization pipeline, or ray differentials. Due to the formulation of the spot noise, the prefiltered noise is the sum of the prefiltered kernels, resulting in our case, in the sum of prefiltered Gaussians. In texture space, we need to compute the convolution of the Gaussian pixel footprint with each Gaussian of the kernel during the evaluation process. Because Gaussians are closed under convolution (please refer to the Appendix), it allows us to analytically compute the prefiltered Gaussian function.

$$g(\mathbf{x}; \lambda_1, \mu_1, \mathbf{\Sigma_1}) \circledast g(\mathbf{x}; \lambda_2, \mu_2, \mathbf{\Sigma_2}) = g(\mathbf{x}; \lambda_3, \mu_3, \mathbf{\Sigma_3}) \tag{6}$$

with

$$\begin{aligned}\mathbf{\Sigma_3} &= \mathbf{\Sigma_1} + \mathbf{\Sigma_2}\\ \mu_3 &= \mu_1 + \mu_2\\ \lambda_3 &= (2\pi)^{D/2}|(\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1}|^{1/2}\lambda_1\lambda_2\end{aligned}$$

As seen in figure 2, we achieve high quality filtering results for a few more calculations during each Gaussian evaluation.

However, the projected pixel footprint may degenerate into extreme shapes depending on the view angle, the scene and parameters. It means that the projected pixel footprint may overlap more neighboring cells than the ones considered during evaluation, resulting in an over-filtering artifact (see the dark regions in the top row of Figure (3)). This problem occurs in extreme situations like high texture space scaling, grazing angle, high grid resolution or at vanishing points.

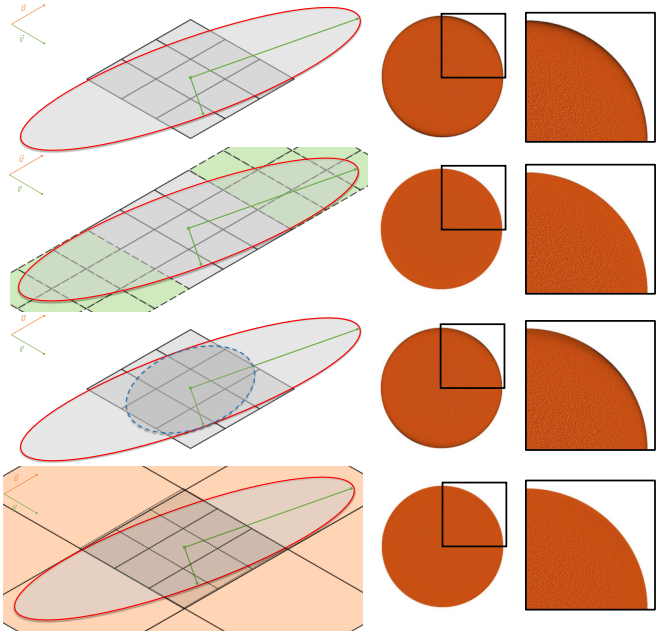


Fig. 3: Top row : In extremes cases, the pixel footprint (in red) may be larger than the neighborhood region considered during evaluation (in gray). If the missing cells are not taken into account, the convolution is biased (dark regions at the sphere boundary). Second row : Considering all the cells within the pixel footprint leads to the correct solution. However, this quickly becomes impractical in a real-time context. Third row : In practice, the pixel footprint is usually clamped to the considered neighborhood (in blue), reintroducing aliasing. Last row : A better practical solution is to precompute a mipmapped representative tile of a neighborhood region and smoothly transition between analytical filtering and this representation.

Adressing this problem requires to increase the number of cells being considered according to the pixel footprint. This has the drawback of increasing the calculations for the concerned pixels. Furthermore, when the pixel footprint degenerates to a flat and elongated shape, the number of cells to be considered becomes impractical for on-the-fly evaluation.

However, an assumption can be made for color textures as the resulting pixel intensity tends toward a mean color. So a first practical solution is to limit the pixel footprint to the considered neighborhood. This works well in most practical cases but at highly grazing angles and high texture space scaling, aliasing is reintroduced. Finally, another practical solution for the on-the-fly filtered evaluation is to precompute a representative tile of the spot noise and fade out between analytical filtering and hardware mipmapping (Figure 3 last row). However, efficiently filtering of on-the-fly procedural functions in these extreme cases still remains an open issue.

## 6. Analytic Normal Mapping

Normal mapping is a common technique used to simulate the presence of geometric details during shading without explicitly generating the actual geometry. We propose to create such maps with our method by taking advantage of the formulation of our spot noise.

Creating normal maps from procedural functions is usually achieved by considering the noise process as a heightfield on the surface and computing its derivatives [30, 31]. They can be computed using finite differences, thus requiring multiple evaluations of the function for each pixel and hampering performance of the process, or analytically. We chose to define our noise as a height field in the tangent frame of the underlying surface and use our formulation to compute the slopes of the heightfield in texture space. By applying the sum rule to the spot noise formulation, we can compute the final slope by adding the partial derivatives of each kernel, and by extension, of each Gaussian. Because of the spot noise formulation, we can directly compute partial derivatives during the evaluation of our procedural function:

$$\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial \mathbf{x}} = \sum_{i=1}^{M} w_i \sum_{j=1}^{N} \frac{\partial g(\mathbf{x} - \mathbf{x_i}; \lambda_j, \mu_j, \mathbf{\Sigma_j})}{\partial \mathbf{x}} \tag{7}$$

with

$$\frac{\partial g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma})}{\partial \mathbf{x}} = g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma})(-\mathbf{\Sigma}^{-1}(\mathbf{x} - \mu)) \tag{8}$$

to finally obtain the partial derivatives of our procedural texture according to $x$ and $y$. During shading, as shown in Fig 4, we can use these scalars to compute the perturbed normal $\omega_\mathbf{p}$ by using the following transformation :

$$\omega_\mathbf{p} = \frac{\left(-\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial x}, -\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial y}, 1\right)^T}{\sqrt{(\frac{\partial \mathrm{lcsn}(\mathbf{x})}{x})^2 + (\frac{\partial \mathrm{lcsn}(\mathbf{x})}{y})^2 + 1}} \tag{9}$$

Alternatively, during the evaluation of the BRDF, we can directly use the computed slope as a shift in a non-centered normal distribution function similar to the formulation proposed by Olano and Baker [32] (Equation (1)) and Dupuy et al. [33] (Equation (8) & (10)).

We remind that normal map filtering is still an open problem in Computer Graphics. Indeed, discrete normal map textures are usually used during shading computation and naively filtered using hardware mipmapping. This has the drawback of changing the appearance of the object when filtered (i.e. incorrect shading). Several approaches have been studied to overcome this issue [34, 32]. For example, moments-based approaches like [32, 33] compute second moments from partial

derivatives. In our case, we need to compute $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}^2$, $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y}^2$ and $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y}$. Because the partial derivatives are computed through a weighted sum, analytically computing moments requires the calculations of the squared sum and each crossed term. This is therefore not suited for on-the-fly evaluation. This specific problem isn't tackled in this paper and is left open for future works.

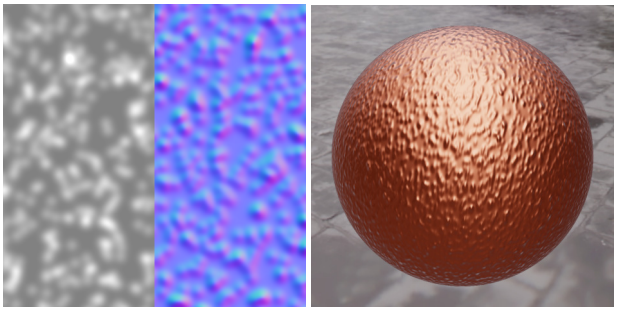


Fig. 4: Using the partial derivatives of the height function (left) we compute the normal map (middle) to generate subtle variations in shading at runtime.

However, Equation 8 admits a closed-form when convolved with a Gaussian footprint. Indeed, when the partial derivative of a Gaussian $g_1(\mathbf{x}) = g(\mathbf{x}; \lambda_1, \mu_1, \mathbf{\Sigma_1})$ is convolved by another Gaussian $g_2(\mathbf{x}) = g(\mathbf{x}; \lambda_2, \mu_2, \mathbf{\Sigma_2})$, it results in :

$$\int_{\mathbb{R}^2} \frac{\partial g_1(\mathbf{x}-\mathbf{t})}{\partial x} g_2(\mathbf{t}) d\mathbf{t} = -(a,b)^T \cdot (\mathbf{x} - (\mu_1 + \mu_3))\lambda_3(2\pi)|\mathbf{\Sigma_3}|^{1/2}$$

$$\int_{\mathbb{R}^2} \frac{\partial g_1(\mathbf{x}-\mathbf{t})}{\partial y} g_2(\mathbf{t}) d\mathbf{t} = -(b,c)^T \cdot (\mathbf{x} - (\mu_1 + \mu_3))\lambda_3(2\pi)|\mathbf{\Sigma_3}|^{1/2}$$

where

$$\begin{aligned}
\mathbf{\Sigma}_1^{-1} &= \begin{pmatrix} a & b \\ b & c \end{pmatrix} \\
\mathbf{\Sigma_3} &= (\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1} \\
\mu_3 &= \mathbf{\Sigma_3}(\mathbf{\Sigma_1}^{-1}\mu_1 + \mathbf{\Sigma_2}^{-1}\mu_2) \\
\lambda_3 &= g(\mu_1; \lambda_1\lambda_2, \mu_2, \mathbf{\Sigma_1} + \mathbf{\Sigma_2})
\end{aligned}$$

We refer the reader to the full derivation that could be found in the supplementary material. This allows us to directly compute the filtered slope of our spot noise at runtime, yielding a mean slope in the projected pixel footprint. By replacing partial derivatives in Equation 9 or by directly using it for shifting non-centered distribution function, we obtain a smooth approximation of the underlying bump surface.

## 7. Results

In this section, we show results of our spot noise model. First, we compare the performance of our new parameterization against the previous model as seen in Figure 5. For the same quality, performances are slightly improved by removing the homogeneous coordinates. We can also compute the partial derivatives while remaining a little bit faster than the previous parameterization (generating only color). However, properly filtering the spot noise requires extra computations, thus slightly lowering performance compared to the aliased previous version.

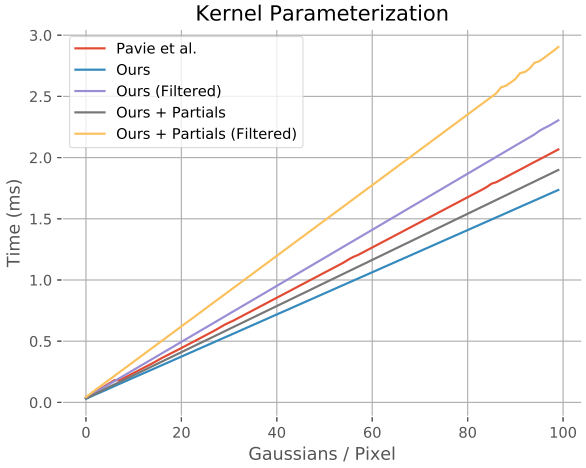


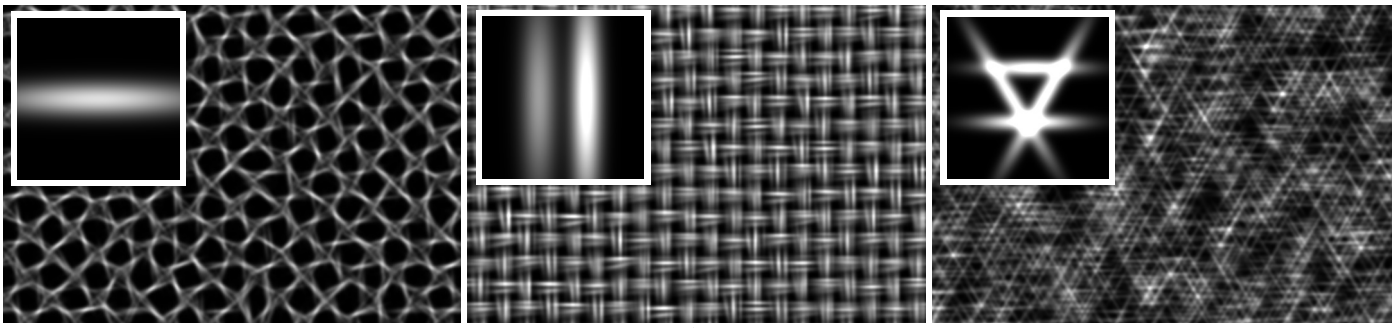

Fig. 5: Performance comparison between our parameterization and the Pavie et al. [5] one. Measures were done on 1920*1080 textures on a GTX 2080.

Table 1 shows the performance (in milliseconds) of three noise patterns on two high-end GPUs (a GTX 1080Ti and a GTX 2080). These three textures, shown in Fig 6, were generated in 2048*2048 with an increasing complexity. As seen in Section 3, the computational complexity of our method is directly linked to the overall number of Gaussian kernels impacting each pixel.

| Scene (fig. 6 ) | Gauss/cell | GTX 1080Ti | GTX 2080 |
|---|---|---|---|
| (a) Hive | 5 | 5.1 ms | 3.3 ms |
| (b) Fiber | 20 | 15.7 ms | 10.7 ms |
| (c) Sparkles | 40 | 28.8 ms | 19.4 ms |

Table 1: Performance of 2D LCSN evaluations in Fig 6 for 2048*2048 textures. Performance are linked to the number of Gaussians covering each pixel.

Next, we show how artistic control can be achieved through the use of external processes (textures, procedural functions, ...). Thanks to our purely geometric formulation, each parameter of our noise can easily be driven by a procedural or discrete (cf. Fig 7) texture to generate spatially varying patterns. This approach is similar to the one introduced by Charpenay et al. [21]. We show on the left image the control over the rotation matrix $R$ (varying horizontally) and local translation of each Gaussian (varying vertically). Similarly, the right image depicts the increasing randomness as the rotation angle is randomly chosen using a narrow (left) or wide (right) uniform distribution. Note that by interpolating values in parameter space before generating the final result, we achieve a smoothly varying appearance without blending artifacts.



(a) (b) (c)

Fig. 6: Results : three patterns generated with our spatially defined spot noise with increasing complexity (ie. number of Gaussians in kernel). (a) : One Gaussian kernel with a symmetric rotation for each odd cells and five impulses per cell. (b) : Two Gaussian kernels with symmetric rotation for each odd cells and ten impulses per cell. (c) : Four Gaussian kernels and ten impulses per cell.

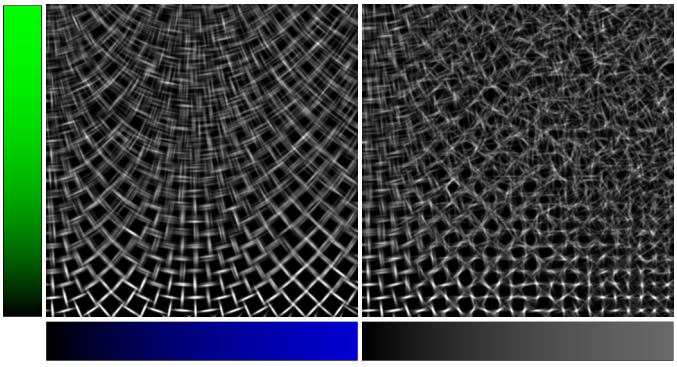


Fig. 7: Each parameter can be directly given by a control map (similar to [21]). Left: the green channel controls the spreading of the distribution while the blue channel drives the kernel rotation. Right: the rotation angle is randomly chosen using an increasing range induced by the alpha channel.

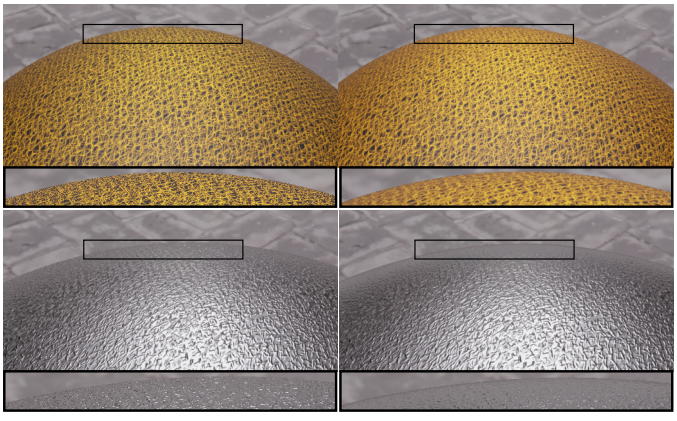


Fig. 8: Our analytic filtering scheme (right column) prevents aliasing artifacts (as seen on the left column) for albedo (top) and normal (bottom).

Figure 8 highlights the increase in visual quality introduced by the analytic filtering scheme, for color noise or normal map generation. Note that, similarly to classical filtering of discrete normal maps, this is still an approximation that generates incorrect shading results (see [34]). As seen in Figure 10, our noise model is devised to be artist-friendly and provide intuitive control of the final visual appearance. Unlike by-example approaches based on the spectral definition of an example, users can finely tune each individual instance of the kernels. Furthermore, randomness of the final appearance can be intuitively and locally controlled, by providing for each element of the target surface the probability density function of each parameter.

Figure 10 (b) depicts such an application where the user can interactively create several kinds of patterns from regular to purely random (we refer the reader to the accompanying video). Our noise can be applied as a post processing step using either an image (as seen in Figure 9) or a framebuffer (Figure 10 (e)) to drive various parameters.

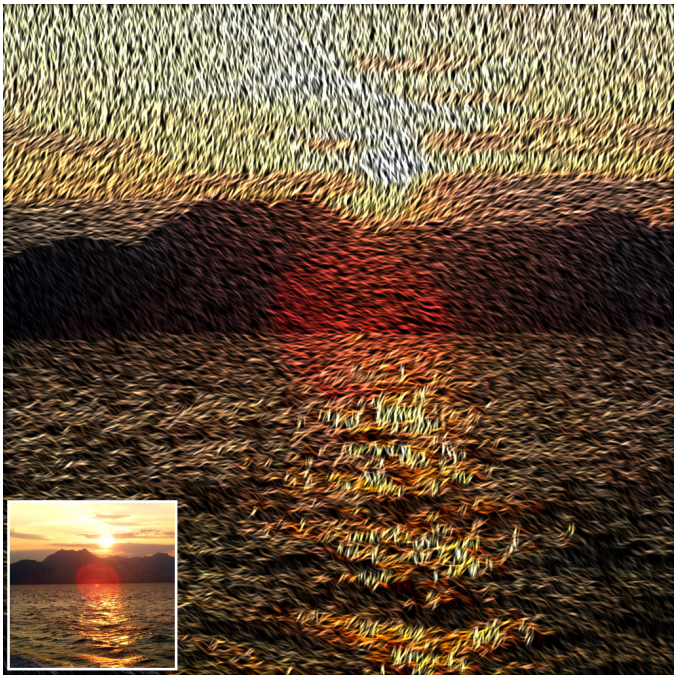


Fig. 9: Our spot noise can be used as a post process to achieve stylized results on an arbitrary input (here a photograph).

Finally, as highlighted by 10 (a) to (d), our spot noise can also be used as a primitive in a shader-graph to drive other procedural methods, yielding composite textures. Here, the spot noise is used as a texture space partition in conjunction with two high performance noises [26] to achieve a complex appearance relying on our normal map generation for shading purpose. We linearly blend the two procedural textures by using the amplitude of our spot noise as a blending mask and simulate geometric de-

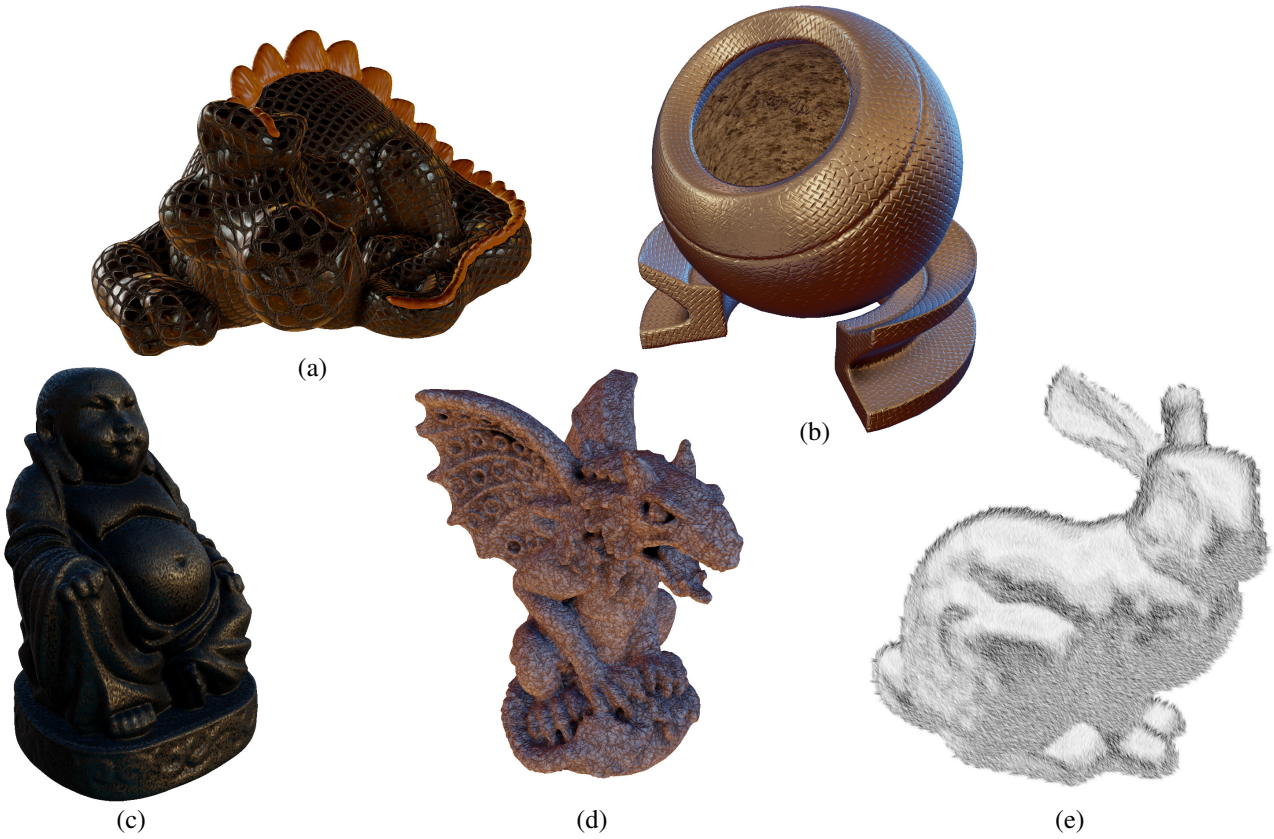



Fig. 10: Results. Our spot noise can generate a wide range of patterns. (a),(c) and (d) highlight non-stationary texturing using our spot noise and high-performance noise [26] (average frametime: 6.87ms). (b) shows patterns resulting from locally controlled parameters yielding a smooth transition in appearance (avg frametime : 4.6ms). (e) Stylized rendering can be achieved using our method in a post processing step (avg frametime : 0.34ms).

tails during shading using the partial derivatives of our model.

## 8. Discussion & Limitations

As shown above, this spot noise generates a wide range of texture appearances (stochastic to near-regular) but the evaluation cost is directly dependent on the number of Gaussians per kernel and the number of kernels considered for a given pixel.

This method also relies on surface parameterization. A setup-free noise (as the one in [17]) will break/deform the structure unless the surface meets some specific properties.

Our filtering scheme yields high quality results, matching the brute-force reference, but is not suited to normal mapping. As stated in Section 6, correctly filtering normal mapping is more complex than computing a mean slope during shading.

A more complex filtering issue appears when using our spot-noise in a procedural hierarchy, such as driving our noise parameters using an analytic or discrete “control” map (cf. Section 7) or using our noise as spatial mask for other procedural methods. In this paper, we independently filtered each layer whereas accurate filtering should simultaneously consider all levels at each stage. This is however out of the scope of this paper.

## 9. Conclusion & Future Works

Our main objective is producing high quality images using on-the-fly procedurally generated textures. In this context we presented three significant improvements to the Locally Controlled Spot Noise : a lighter kernel formulation, which alleviates the evaluation process, an anisotropic filtering scheme, which prevents aliasing and an analytic bump mapping strategy in order to easily add surface details during shading. Furthermore, our method provides intuitive user controls through the purely geometric parameterization of our new kernel.

To achieve a high level of realism, virtual appearances should exhibit fine non-stationary details at multiple scales. As shown in our work, a promising venue of future research resides in the conjoint use of procedural stationary texture synthesis [26, 23] with spatially varying structured patterns generation to create elaborate surface textures, though filtering issues should be carefully considered.

In a similar fashion, the link between correctly filtering normal maps and the final appearance of the material have been widely studied in Computer Graphics. Efficient filtering of procedurally generated normal maps is a non-trivial issue and is an interesting topic for future works.

## Acknowledgments

The authors would like to thank Xavier Chermain for his corrections. Bunny Model is provided by Stanford repository, Mori model is courtesy of McGuire Graphic Archive. Buddha and Gargoyle models are provided by AIM@SHAPE. Dragon model is courtesy of UTIA and CGG. Guillaume Gilet acknowledges support from project HDWorlds from the Agence Nationale de la Recherche.

## References

[1] Lagae, A, Lefebvre, S, Cook, R, DeRose, T, Drettakis, G, Ebert, DS, et al. A survey of procedural noise functions. In: Computer Graphics Forum; vol. 29. Wiley Online Library; 2010, p. 2579–2600.

[2] Ebert, DS, Musgrave, FK. Texturing & modeling: a procedural approach. Morgan Kaufmann; 2003.

[3] Wei, LY, Lefebvre, S, Kwatra, V, Turk, G. State of the art in example-based texture synthesis. 2009,.

[4] van Wijk, JJ. Spot noise texture synthesis for data visualization. SIGGRAPH Comput Graph 1991;25(4):309–318. URL: http://doi.acm.org/10.1145/127719.122751. doi:10.1145/127719.122751.

[5] Pavie, N, Gilet, G, Dischler, JM, Ghazanfarpour, D. Procedural texture synthesis by locally controlled spot noise [c]. In: Wscg. 2016,.

[6] Efros, AA, Freeman, WT. Image quilting for texture synthesis and transfer. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01; New York, NY, USA: ACM. ISBN 1-58113-374-X; 2001, p. 341–346. URL: http://doi.acm.org/10.1145/383259.383296. doi:10.1145/383259.383296.

[7] Liang, L, Liu, C, Xu, YQ, Guo, B, Shum, HY. Real-time texture synthesis by patch-based sampling. ACM Transactions on Graphics (ToG) 2001;20(3):127–150.

[8] Cohen, MF, Shade, J, Hiller, S, Deussen, O. Wang tiles for image and texture generation. ACM Trans Graph 2003;22(3):287–294. URL: http://doi.acm.org/10.1145/882262.882265. doi:10.1145/882262.882265.

[9] Wei, LY. Tile-based texture mapping on graphics hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. ACM; 2004, p. 55–63.

[10] Vanhoey, K, Sauvage, B, Larue, F, Dischler, JM. On-the-fly multi-scale infinite texturing from example. ACM Trans Graph 2013;32(6):208:1–208:10. URL: http://doi.acm.org/10.1145/2508363.2508383. doi:10.1145/2508363.2508383.

[11] Glanville, S. Texture bombing. In: GPU Gems; chap. 20. 2004,.

[12] Lagae, A, Dutré, P. A procedural object distribution function. ACM Trans Graph 2005;24(4):1442–1461. URL: http://doi.acm.org/10.1145/1095878.1095888. doi:10.1145/1095878.1095888.

[13] Gilet, G, Dischler, JM, Ghazanfarpour, D. Multi-scale assemblage for procedural texturing. Computer Graphics Forum 2012;31(7):2117–2126. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03204.x. doi:10.1111/j.1467-8659.2012.03204.x.

[14] Lewis, JP. Texture synthesis for digital painting. SIGGRAPH Comput Graph 1984;18(3):245–252. URL: http://doi.acm.org/10.1145/964965.808605. doi:10.1145/964965.808605.

[15] Lewis, JP. Methods for stochastic spectral synthesis. In: Proceedings on Graphics Interface '86/Vision Interface '86. Toronto, Ont., Canada, Canada: Canadian Information Processing Society; 1986, p. 173–179. URL: http://dl.acm.org/citation.cfm?id=16564.16594.

[16] Lewis, JP. Algorithms for solid noise synthesis. SIGGRAPH Comput Graph 1989;23(3):263–270. URL: http://doi.acm.org/10.1145/74334.74360. doi:10.1145/74334.74360.

[17] Lagae, A, Lefebvre, S, Drettakis, G, Dutré, P. Procedural noise using sparse gabor convolution. ACM Transactions on Graphics (TOG) 2009;28(3):54.

[18] Gilet, G, Dischler, JM, Ghazanfarpour, D. Multiple kernels noise for improved procedural texturing. The Visual Computer 2012;28(6-8):679–689.

[19] Neyret, F, Heitz, E. Understanding and controlling contrast oscillations in stochastic texture algorithms using Spectrum of Variance. Research Report; LJK / Grenoble University - INRIA; 2016. URL: https://hal.inria.fr/hal-01349134.

[20] Tavernier, V, Neyret, F, Vergne, R, Thollot, J. Making Gabor Noise Fast and Normalized. In: Association, TE, editor. Eurographics 2019 - 40th Annual Conference of the European Association for Computer Graphics. Eurographics 2019 - Short Papers; Gênes, Italy; 2019, p. 1–4. URL: https://hal.inria.fr/hal-02104389. doi:10.2312/egs.20191009.

[21] Charpenay, V, Steiner, B, Musialski, P. Sampling gabor noise in the spatial domain. In: Gutierrez, D, editor. Proceedings of the 30th Spring Conference on Computer Graphics - SCCG. ACM Press. ISBN 978-80-223-3601-7; 2014, p. 79–82. URL: https://www.cg.tuwien.ac.at/research/publications/2014/charpenay-2014-sgn/.

[22] Galerne, B, Lagae, A, Lefebvre, S, Drettakis, G. Gabor noise by example. ACM Transactions on Graphics (TOG) 2012;31(4):73.

[23] Galerne, B, Leclaire, A, Moisan, L. Texton noise. Computer Graphics Forum 2017;:n/a–n/aURL: http://dx.doi.org/10.1111/cgf.13073. doi:10.1111/cgf.13073.

[24] Gilet, G, Sauvage, B, Vanhoey, K, Dischler, JM, Ghazanfarpour, D. Local random-phase noise for procedural texturing. ACM Transactions on Graphics (TOG) 2014;33(6):195.

[25] Guingo, G, Sauvage, B, Dischler, JM, Cani, MP. Bi-Layer textures: a Model for Synthesis and Deformation of Composite Textures. Computer Graphics Forum 2017;36(4):111–122. URL: https://hal.archives-ouvertes.fr/hal-01528537.

[26] Heitz, E, Neyret, F. High-Performance By-Example Noise using a Histogram-Preserving Blending Operator. Proceedings of the ACM on Computer Graphics and Interactive Techniques 2018;1(2):Article No. 31:1–25. URL: https://hal.inria.fr/hal-01824773. doi:10.1145/3233304.

[27] Tricard, T, Efremov, S, Zanni, C, Neyret, F, Martínez, J, Lefebvre, S. Procedural phasor noise. ACM Trans Graph 2019;38(4):57:1–57:13. doi:10.1145/3306346.3322990.

[28] Pavie, N, Gilet, G, Dischler, JM, Galin, E, Ghazanfarpour, D. Volumetric spot noise for procedural 3d shell texture synthesis. In: Proceedings of the conferece on Computer Graphics & Visual Computing. Eurographics Association; 2016, p. 33–40.

[29] Heckbert, PS. Fundamentals of texture mapping and image warping. Tech. Rep. UCB/CSD-89-516; EECS Department, University of California, Berkeley; 1989. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1989/5504.html.

[30] Blinn, JF. Simulation of wrinkled surfaces. In: ACM SIGGRAPH computer graphics; vol. 12. ACM; 1978, p. 286–292.

[31] Mikkelsen, MS. Bump mapping unparametrized surfaces on the gpu. Journal of Graphics, GPU, and Game Tools 2010;15(1):49–61.

[32] Olano, M, Baker, D. Lean mapping. In: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. ACM; 2010, p. 181–188.

[33] Dupuy, J, Heitz, E, Iehl, JC, Pierre, P, Neyret, F, Ostromoukhov, V. Linear Efficient Antialiased Displacement and Reflectance Mapping. ACM Transactions on Graphics 2013;32(6). URL: https://hal.inria.fr/hal-00858220.

[34] Bruneton, E, Neyret, F. A survey of non-linear pre-filtering methods for efficient and accurate surface shading. IEEE Trans Vis Comput Graph 2012;18(2):242–260.

[35] Petersen, KB, Pedersen, MS. The matrix cookbook. 2012. URL: http://www2.imm.dtu.dk/pubdb/p.php?3274; version 20121115.

## Appendix

*Closed-form solution for Gaussians convolution*

To obtain Equation 6, we need to compute the convolution $I$ of two Gaussian functions :

$$I = \int_{\mathbb{R}^D} g(\mathbf{t}; \lambda_1, \mu_1, \mathbf{\Sigma_1}) g(\mathbf{x} - \mathbf{t}; \lambda_2, \mu_2, \mathbf{\Sigma_2}) d\mathbf{t} \qquad (10)$$

We first recall closed-form solutions of the integral of a Gaussian function and the product of two Gaussian functions. Then, we compute the integral of the product of two Gaussian functions and, by substitution, solve Equation 10.

***Integral of a multivariate Gaussian***. Integrating a given multivariate Gaussian $g$ over $\mathbb{R}^D$ yields :

$$\int_{\mathbb{R}^D} g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) d\mathbf{x} = \lambda \cdot (2\pi)^{D/2} |\mathbf{\Sigma}|^{1/2} \tag{11}$$

***Product of two multivariate Gaussians***. The product of two multivariate Gaussians is another multivariate Gaussian [35]:

$$g(\mathbf{x}; \lambda_1, \mu_\mathbf{1}, \mathbf{\Sigma_1}) \cdot g(\mathbf{x}; \lambda_2, \mu_\mathbf{2}, \mathbf{\Sigma_2}) = g(\mathbf{x}; \lambda_3, \mu_\mathbf{3}, \mathbf{\Sigma_3}) \tag{12}$$

where

$$\begin{aligned}
\mathbf{\Sigma_3} &= (\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1} \\
\mu_3 &= \mathbf{\Sigma_3}(\mathbf{\Sigma_1}^{-1}\mu_1 + \mathbf{\Sigma_2}^{-1}\mu_2) \\
\lambda_3 &= g(\mu_\mathbf{1}; \lambda_1\lambda_2, \mu_\mathbf{2}, \mathbf{\Sigma_1} + \mathbf{\Sigma_2})
\end{aligned}$$

***Integral of the product of two multivariate Gaussians***. By applying equations 11 and 12, the integral results in :

$$\begin{aligned}
&\int_{\mathbb{R}^D} g(\mathbf{x}; \lambda_1, \mu_\mathbf{1}, \mathbf{\Sigma_1}) g(\mathbf{x}; \lambda_2, \mu_\mathbf{2}, \mathbf{\Sigma_2}) d\mathbf{x} \\
&\quad = g(\mu_\mathbf{1}; \lambda_1\lambda_2, \mu_\mathbf{2}, \mathbf{\Sigma_1} + \mathbf{\Sigma_2}) \int_{\mathbb{R}^D} g(\mathbf{x}; 1, \mu_\mathbf{3}, \mathbf{\Sigma_3}) d\mathbf{x} \\
&\quad = g(\mu_\mathbf{1}; \lambda_1\lambda_2(2\pi)^{D/2}|\mathbf{\Sigma_3}|^{1/2}, \mu_\mathbf{2}, \mathbf{\Sigma_1} + \mathbf{\Sigma_2})
\end{aligned}$$

***Convolution of multivariate Gaussians***. By substituting Equation 10 in Equation 12, we obtain :

$$\begin{aligned}
&\int_{\mathbb{R}^D} g(\mathbf{t}; \lambda_1, \mu_\mathbf{1}, \mathbf{\Sigma_1}) g(\mathbf{x} - \mathbf{t}; \lambda_2, \mu_\mathbf{2}, \mathbf{\Sigma_2}) d\mathbf{t} \\
&\quad = \int_{\mathbb{R}^D} g(\mathbf{t}; \lambda_1, \mu_\mathbf{1}, \mathbf{\Sigma_1}) g(\mathbf{t}; \lambda_2, \mathbf{x} - \mu_\mathbf{2}, \mathbf{\Sigma_2}) d\mathbf{t} \\
&\quad = g(\mathbf{x}; \lambda_1\lambda_2(2\pi)^{D/2}|\mathbf{\Sigma_3}|^{1/2}, \mu_\mathbf{1} + \mu_\mathbf{2}, \mathbf{\Sigma_1} + \mathbf{\Sigma_2})
\end{aligned}$$