CGI12 manuscript No.
(will be inserted by the editor)

Multiple Kernels Noise for Improved Procedural Texturing

G. Gilet, J-M. Dischler and D. Ghazanfarpour

The final publication is available at www.springerlink.com

Abstract Procedural texturing is a well known method to synthesize details onto virtual surfaces directly during rendering. But, the creation of such textures is often a long and painstaking task. This paper introduces a new noise function, called multiple kernels noise. It is characterized by an *arbitrary* energy distribution in spectral domain. Multiple kernels noise is obtained by adaptively decomposing a user-defined power spectral density (PSD) into rectangular regions. These are then associated to kernel functions used to compute noise values by sparse convolution. We show how multiple kernels noise: 1) increases the variety of noisy procedural textures that can be modeled and 2) helps creating structured procedural textures by automatic extraction of noise characteristics from user-supplied samples.

Keywords Procedural textures · Rendering · Noise-based texturing

1 Introduction

Since its introduction to the computer graphics community in the mid-eighties [13], procedural noise has been successfully used to model the most various natural phenomena: ocean waves, terrains, etc. It represents an important tool for many applications such as texturing, modeling and animation. The main characteristic of noise is its spectral energy distribution (or PSD for *power spectral density*). A good spectral control allows one to increase the variety of phenomena that can be modeled and permits a more accurate anti-aliasing. Most works concerning procedural noise have been done to improve spectral control. But in spite of these works, creating a procedural noise that is characterized by an arbitrary PSD yet remains an unsolved problem. A common approach in signal processing trivially consists in filtering by multiplication a discrete random spectral domain with a filter matching the desired PSD and then applying an inverse Fast Fourier Transform (FFT). But such an approach does not produce fully procedural noise functions. In computer graphics, noise aims at being a function returning pseudo random values and should have the following properties: infinity, continuity, low and constant computational complexity for an evaluation at any random location of space, extreme compactness, as well as easy extension to higher dimensions. The recently introduced Gabor noise [8] matches these properties. But the problem of providing noise featuring an arbitrary PSD is still not addressed.

Our aim is to build a noise function characterized by an arbitrary PSD using, as for Gabor noise, sparse convolution [10]. We study and compare different finite impulse response (FIR) kernels (not only the Gaussian kernel of Gabor noise) and show how an arbitrary PSD can be decomposed adaptively according to the spectral characteristics of these kernels. One key issue is that the user can directly balance the quality and precision of the resulting noise against computational cost.

Good spectral control allows us to improve the creation of procedural textures. Users generally have to browse the parameters of noise until a given desired result is

G. Gilet and D. Ghazanfarpour
Tel : +33 (0)555 45 72 50
Fax : +33 (0)555 45 76 97
XLIM, University of Limoges
123 Avenue Albert Thomas 87000 Limoges
E-mail: {Guillaume.Gilet,Djamchid.Ghazanfarpour}@unilim.fr

J-M. Dischler
Tel : (+33) 03 68 85 45 59
Fax : (+33) 03 68 85 44 55
LSIIT, University of Strasbourg
11, Bd Sebastien Brandt 67000 Illkirch Grafenstaden
E-mail: dischler@unistra.fr

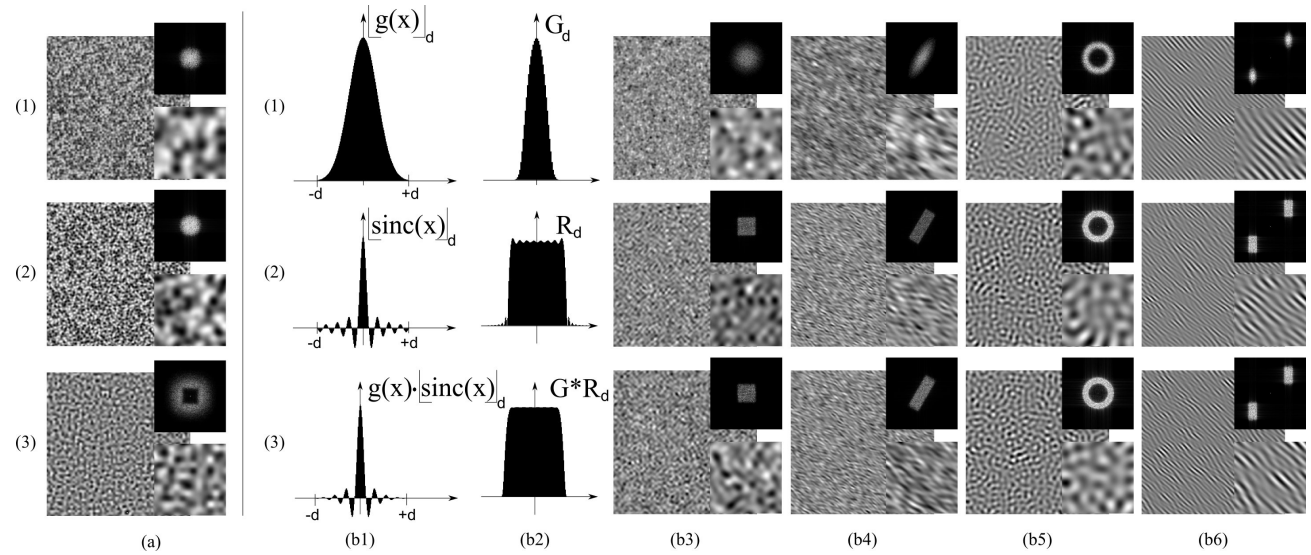

Fig. 1 Comparing procedural noises. Column (a): grid-based noises. Column (b1-b6): convolution noises based on different kernels (from top to bottom Gaussian / Gabor (1), sinc (2) and Gaussian-sinc (3)).

obtained more or less experimentally. As this task can be tedious and time-consuming, recent noise parameter fitting techniques [9,4] have attempted to automate it. But these methods only process limited spectral characteristics and focus on purely noisy patterns for texture synthesis. We propose new approaches, based on multi-scale decomposition and perturbation, that allow us to process efficiently a greater variety of textures. One key contribution is that we are able to determine adequate noise parameters automatically. We note that unlike [9, 4] our aim is not to provide an example-based texture synthesis technique. Instead, our aim is to facilitate the interactive creation of purely procedural textures.

The following section proposes a brief overview of related works. Then, we present noise functions based on sparse convolution and compare filter kernels. Based on an adaptive decomposition of spectral domain, we show how to create multiple kernels noise functions. Then, we introduce our procedural texture modeling technique exploiting our previously defined noise. Finally, before concluding, we present results and discuss limitations.

2 Related Work

Noise has been introduced by Perlin in [13] more than two decades ago. Since then, various statistical, spectral and visual aspects have been investigated in papers [13,10,11,17,15,2,5,8]. Mainly two approaches [7] have been proposed to define noise functions: grid-based techniques that use an interpolation function (for instance a polynomial function) and sparse convolution-based techniques that use a certain convolution kernel function along with a random point distribution. More recently hardware issues have been considered. Lagae et al. [8] propose to compute procedural noise at real-time rates using a convolution-based technique [10], and to control some aspects of its spectral energy distribution by using Gabor kernels.

The creation of procedural textures is not simple : users have to learn shader programming languages and need a good intuition concerning the behavior of noise. Whereas many methods have been proposed to synthesize texture images from examples (see [16]), methods that help users creating procedural textures are, conversely, rare. In what follows, we review the few works that we are aware of. Noise-based descriptions of displacement textures is addressed in [3]. Parameters of a sum of gradient noises are adjusted using a combined spectral and histogram analysis of profiles. An approach that produces parameters for procedural color texture shaders is proposed in [1]. It consists in browsing the multi-dimensional space of parameters using a pre-existing database of shaders. As a result the system outputs the best matching shader with the corresponding best matching parameters. But this system does not create new shaders. In [9], Lagae et al. express isotropic stochastic textures as a weighted sum of wavelet noises [2] using example images. The approach does not need any database. But it is limited by the small range of textures that it can deal with, since it assumes the input texture is purely noisy and isotropic. Gilet et al. [4] propose a technique based on a more advanced subdivision of spectral domain by fitting elliptical shapes related to Gabor noise [8]. A greater variety of purely noisy textures can be processed, such as anisotropic ones, but spectral control still remains limited.

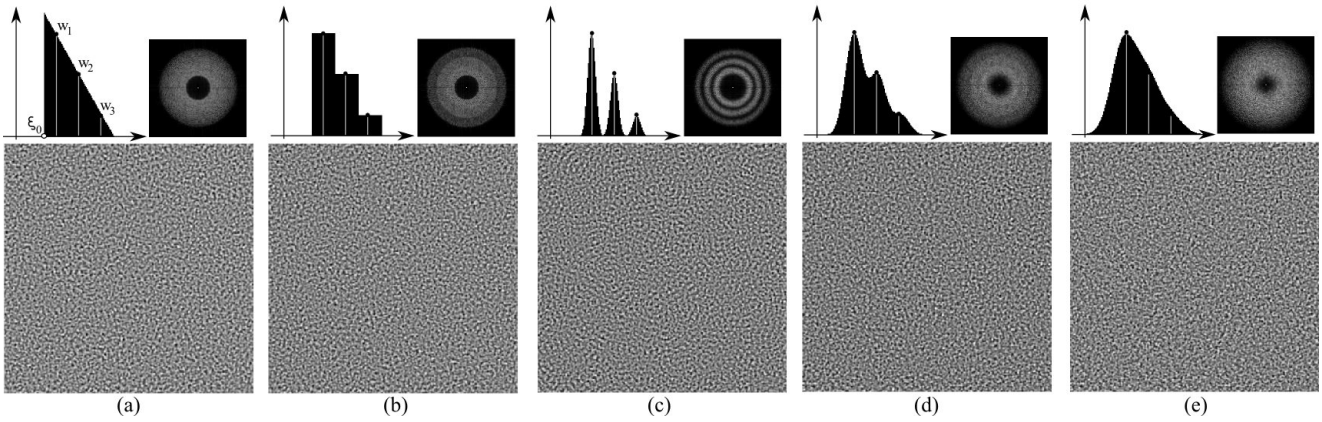

Fig. 2 Example of isotropic multiple kernels noise approximation using a set of $m = 3$ kernels. We compare the use of sinc and Gabor kernels. (a) desired PSD (we show a discrete noise image obtained by inverse FFT. It is not a function). (b),(c),(d) and (e) are approximations of (a) using different kernels: (b) sinc, (c), (d) and (e) Gabor with varying σ for the Gaussian. Unlike (a), these are continuous and infinite noise functions.

3 Convolution-based procedural noise

In the following, we assume the reader is familiar with the Fourier Transform (FT). Figure 1 compares procedural noise functions. For each noise, the upper right is an approximation of the PSD computed using Welch's method and the Fast Fourier Transform algorithm (FFT). The lower right is a close up view. Part (a) shows examples of grid noises, resp. from top to down, gradient noise (1) [13], simplex noise (2) [14] and wavelet noise (3) [2]. Parts (b1-b6) illustrate noises computed using sparse convolution [10]. The corresponding spatial kernel functions are shown in column (b1) and their FT in (b2). Convolution noise consists in convolving an approximation of white noise $n_w(x)$ with a spatial kernel function $k(x)$. Randomly distributed points x_j are used, so that the noise $n_k(x)$ can be expressed as a sum $n_k(x) = k(x) * n_w(x) \approx \sum \zeta_j(x_j)k(x - x_j)$, where $*$ represents convolution and ζ_j a random value. Because of the convolution theorem, the resulting PSD of n_k is $N_0|K|^2$, N_0 being the constant PSD of white noise and $|K|^2$ the PSD of k, i.e. $K(\xi) = \mathcal{F}(k(x))$, $\mathcal{F}$ denoting the FT . In other words, the choice of k allows one to control directly the PSD of noise, which represents a great benefit. A priori any spatial kernel function k can be used. But for practical reasons, k must have a finite support, so that, beyond a certain distance d, its value is zero. Furthermore, it must be an analytical and continuous function (at least C^0) to avoid visual artifacts and provide simple and rapid evaluations. We show three kernels matching these conditions in figure 1. We have focused our attention on these kernels since we found them to be the most useful for creating multiple kernels noise (see next section). Row (1) corresponds to a clamped Gaussian kernel $\lfloor g(x) \rfloor_d$, i.e. $e^{-\sigma x^2}$ for $x < d$ and 0 for $x >= d$. Row (2) shows a clamped cardinal sinus function defined by $\lfloor sinc(x) \rfloor_d = \frac{\sin(\pi \cdot x)}{\pi \cdot x}$ if $x < d$ and 0 otherwise. Row (3) corresponds to a clamped sinc multiplied by a Gaussian. Theoretically, in row (1), the FT of $e^{-\sigma x^2}$ is the Gaussian $G(\xi) = \sqrt{\frac{\pi}{\sigma}} e^{-\frac{\pi^2 \xi^2}{\sigma}}$. But since we have clamped g, some non-ideal frequencies are introduced and only an approximation, G_d, is obtained. Provided d is chosen correctly according to σ, these frequencies are completely negligible. The sinc function is well-known in sampling theory and recognized as an ideal reconstruction filter. Theoretically, in row (2), the FT of $sinc(ax)$ is the rectangular function $R = \frac{1}{a} rect(\frac{\xi}{a})$. But since we clamped the sinc, the obtained FT is only an approximation that we call $R_d(\xi)$. d has been chosen at a zero cross-point to make the kernel C^0 continuous. Introduced non-ideal frequencies depend on d. We can reduce spectral errors related to a low value of d by further multiplying the clamped sinc with a Gaussian as shown in row (3),i.e. the rectangle becomes smoother.

Column (b3) shows corresponding samples of 2D noise functions obtained by extending the previous 1D kernels to dimension two. For the Gaussian, the extension is $e^{-(\sigma_x x^2 + \sigma_y y^2)}$ and for the sinc it is $sinc(a_x x) \times sinc(a_y y)$. As previously demonstrated, the PSD of the resulting noise is matching the corresponding kernel PSD of column (b2). The noises of column (b3) only differ by their PSD, which explains visual differences. Note that, unexpectedly, the improved spectral accuracy of the Gaussian-sinc (row 3) does not have an important impact in spatial domain. The noises of row 2 and row 3 are almost identical. Stated differently, for the chosen d, spectral errors do not introduce any noticeable visual artifacts. All noises of (b3) can be made anisotropic by using different scaling factors $\sigma_x \neq \sigma_y$ (resp. $a_x \neq a_y$).

Resulting ellipses and rectangles in spectral domain can be further arbitrarily oriented by applying a rotation in spatial domain, because rotations are preserved by the FT. Column (b4) illustrates this. We can also shift the spectral kernel to make it centered at any desired frequency position by multiplying k with a cosine. In this case, n_k is defined by:

$$n_k(x,y) = \sum_j \zeta_j(x_j,y_j)k(x-x_j,y-y_j)) \cdot cos(\theta) \quad (1)$$

$$\theta = \varphi \cdot (cos(\gamma)\cdot(x-x_j) + sin(\gamma)\cdot(y-y_j)))$$

with ($\varphi \in \Re$,$\gamma \in [0..2\pi]$) the frequency and orientation parameters of the cosine factor. When γ is chosen randomly, a ring in spectral domain is obtained (see column (b5)). Note that a Gaussian multiplied by a cosine is called a *Gabor function*. Row (1) therefore corresponds to Gabor noise as introduced in [8]. All three noises of column (b5) look very similar because of the global spectral similarity. Column (b6) finally illustrates the case of a fixed value for γ. The kernels are shifted at frequency position (ξ,ψ), φ and γ simply being the polar coordinates of (ξ,ψ).

4 Building multiple kernels noise functions with arbitrary PSD

We build multiple kernels noise n_S by defining a given desired PSD $|P_S|^2$. The user may either directly define the PSD or he may paint a discrete sample of the desired spatial noise. In the latter case, a discrete mean $|P_S|^2$ is obtained by computing sets of FFTs using Welch's method. In what follows, we first describe a property regarding sums of independent noises. Next, we use this property to approximate an arbitrary PSD using a weighted sum of strictly positive functions, thus defining n_S as a sum of independent convolution noises. Let us consider following sum of noises n_{k_i}:

$$n_S \approx \sum_{i=1}^{m} w_i n_{k_i} \quad (2)$$

w_i are positive weights. We want to evaluate the PSD of n_S according to the PSDs of the noises n_{k_i}. We use the Wiener-Khintchine theorem that states that the PSD of a signal f is linked to the autocorrelation $\mathcal{R}(f(x))$ by the FT, i.e. $|F(\xi)|^2 = \mathcal{F}(\mathcal{R}(f))$. The autocorrelation of a sum can be expressed as a sum of cross-correlations:

$$\mathcal{R}(\sum w_i n_{k_i}) = \sum w_i^2 \mathcal{R}(n_{k_i}) + 2\sum w_i w_j \mathcal{R}(n_{k_i}, n_{k_j})$$

By definition, the cross-correlation of two independent random stationary signals is null. The cross-correlations $\mathcal{R}(n_{k_i}, n_{k_j})$ are subsequently null. We obtain:

$$|\mathcal{F}(n_S)|^2 \approx N_0 \sum_{i=1}^{m} w_i^2 |K_i(\xi-\xi_i)|^2 \quad (3)$$

In other words, the PSD of a weighted sum of independent noises is the weighted sum of PSDs of these noises. We can exploit this property to approximate any multiple kernels noise n_S by a convolution of random points with sets of different kernels k_i. By substituting n_{k_i} in formula 2 by its definition (formula 1), we obtain:

$$n_S \approx \sum_{i=1}^{m} w_i \sum_j \zeta_{(i,j)} k_i(x-x_{(i,j)}, y-y_{(i,j)}))cos(\theta_i) \quad (4)$$

Since all noises of this sum are based on a convolution with random points, we can further replace the double sum by a single convolution, thus defining multiple kernels noise as follows:

$$n_S \approx \sum_j \zeta_j w_{\chi_j} k_{\chi_j}(x-x_j, y-y_j))cos(\theta_{\chi_j}) \quad (5)$$

with $\chi_j \in [1,m]$ being a uniform (equiprobable) random kernel index associated to j. Figure 2 illustrates an example of 1D PSD approximation with $m = 3$ kernels. (a) represents the desired PSD. In 1D, it is defined as a decreasing linear ramp starting from frequency $\xi_0 > 0$ (a band-pass filtered pink noise). Its isotropic 2D extension is characterized by a ring. The second row of (a) shows the corresponding noise image obtained by computing a discrete inverse FFT. Note that it is a *discrete noise image*, not a *noise function*. The noises from (b) to (e), on the other hand, are infinite and continuous functions. (b) illustrates an approximation of the desired PSD using rectangular functions R. (c), (d) and (e) use Gaussians G with increasing σ. By assuming $G = G_d$ and $R = R_d$ (meaning that we neglect spectral errors), a multiple kernels noise function is obtained using formula 5 with $k = \lfloor g(x) \rfloor_d$ (resp. $k = \lfloor sinc(x) \rfloor_d$). The previously described property is well illustrated by this example. The convolution with random points leads to a PSD close to the desired one (see 2D PSDs on the right side). The visual quality of our noise approximation strongly depends on the chosen kernels and their parameters. As can be seen in the 1D chart (e), the Gaussian kernel, linked to Gabor-noise, provides a good approximation of the 1D linear ramp in spectral domain. But in spatial domain, the resulting noise (e) appears to be *too noisy* when compared to the reference noise image (a). The reason is that some energy is diffused towards undesired frequencies because the Gaussian does not fall off sufficiently rapidly on ξ_0. On the other hand, the approximation obtained using the rectangular kernels in (b) results in a multiple kernels noise function that visually better matches (a). In the case of Gabor-noise, it is possible to reduce the width of the Gaussians by increasing σ, thus diffusing less energy. This is shown in parts (c) and (d). But when increasing too much σ, as in (c), the

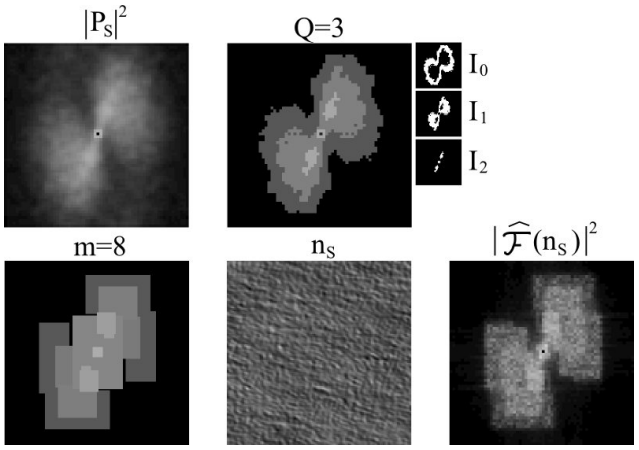

Fig. 3 Adaptive decomposition of the PSD into rectangular regions to define multiple kernels noise approximations. From left to right, top : desired PSD, $Q = 3$ energy classes; down : set of $m = 8$ regions each being associated to one kernel, spatial domain of obtained noise (based on sinc kernels), corresponding PSD computed using Welch's method.

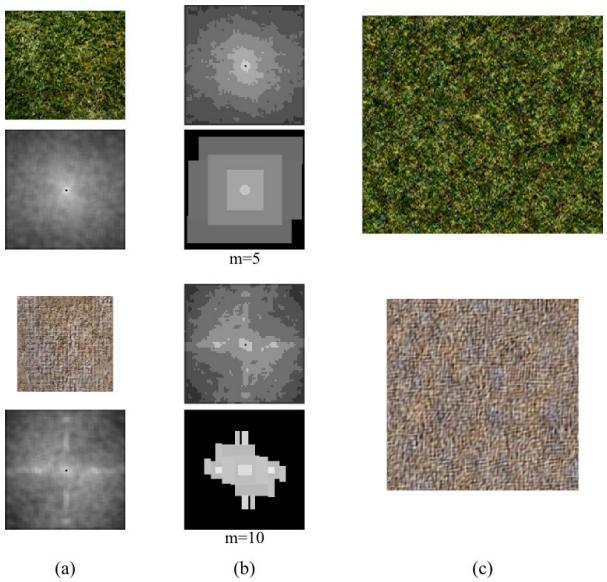

Fig. 4 Same principle as in [9,4]. Multiple kernels noise is directly used to define "by example" noisy procedural textures (in our case, the example can have any type of PSD as opposed to [9,4]). (a) example with corresponding PSD, (b) adaptive decomposition, (c) crop of the resulting procedural texture. It is an infinite and continuous function that entirely fits into the fragment shader using no texture memory.

global PSD approximation, and therefore the resulting noise, becomes less precise. An acceptable compromise is obtained for (d).

The previous principle can be generalized to an arbitrary 2D PSD. Our objective is to approximate the PSD using a sum of m user-defined convolution noises according to formula 3. The challenge is determine best parameters for the weights w_i, as well as frequency positions ξ_i, ψ_i and scales for the $K_i = R$ (or $K_i = G$). Since the target PSD can be considered as a gray level image, it could be efficiently decomposed using wavelet analysis. But wavelet transforms imply negative values, which are excluded in our case because PSDs are always strictly positive. In addition, wavelet transforms consider a constant error threshold all over the domain. This is not suitable in our case because low energy regions in frequency domain do less contribute in spatial domain, than high energy regions do.

Our approach is adaptive and allows us to differentiate high and low energy regions. It is also suitable for both kernels, G and R. Basically, it consists in decomposing the 2D PSD into rectangular zones for different energy levels. First, the energies $|P_S(\xi, \psi)|^2$ of the PSD are quantized into Q values E_q, $q \in [1, Q]$, sorted by increasing order and with $E_0 = 0$. This allows us to segment Q regions. Each region corresponds to the set of frequencies (ξ, ψ) such that their energy is in the range $[E_q, E_{q+1}]$.

This set of frequencies corresponds to a binary image $I_q(\xi, \psi)$. Each of these sets can be processed independently using an adaptive recursive technique starting with a root rectangle matching the bounding box of I_q. The criterion to decide whether or not it is necessary to further subdivide the rectangle consists in measuring the ratio of frequency coverage, i.e. the ratio between the number of frequencies in $I_q(\xi, \psi)$ and the total number of frequencies covered by the rectangle. If the ratio is below a certain threshold, then the rectangle is subdivided. Otherwise the procedure stops and a corresponding kernel is associated. For low energy regions we allow a higher error than for high energy regions, i.e. the error is made depending on q. Once a set of rectangular regions has been computed, one kernel is associated to each region. For the kernel R, which already has a rectangular footprint, the association is trivial. We can just use the length and width of the rectangle to define the scaling parameters of R. For the Gaussian kernel G, we also use the length and width to define anisotropy and such that the Gaussians slightly overlap neighboring regions, as done in figure 2. Finally, the weights w_i are obtained by using an iterative procedure that minimizes energy differences, i.e. such that $argmin(|P_S(\xi, \psi)|^2 - \sum_{i=1}^{m} w_i^2 |K_i(\xi - \xi_i, \psi - \psi_i)|^2)$, $\forall(\xi, \psi)$. The iteration works as follows: we start with w_i matching the energy of the corresponding region. This defines the noise n_S. We then compute an approximate mean PSD $\left|\widehat{\mathcal{F}}(n_S)\right|^2$ using Welch's method to compare it to the reference PSD $|P_S|^2$. According to energy differences inside regions, the weights w_i are adjusted by a factor ϵ. For all examples in this paper, we applied a

fixed number of 25 iterations.
To reduce the amount m of kernels, we also consider separately regions $I_q(\xi, \psi)$ that are centered at the origin. We test if they approximately match an elliptical disk or ring (using the same frequency coverage criterion as for a rectangle). Indeed, for such regions, a single scaled and rotated isotropic noise function (like the ones depicted in figure 1 columns (b3-b5)) might provide a better approximation compared to rectangles. Figure 3 illustrates an example of 2D PSD decomposition and obtained multiple kernels noise.

5 Procedural textures based on multiple kernels noise

In the works of [9] and [4], an example image is considered as a kind of noisy pattern and defined by using a weighted sum of noises at different scales so that the resulting sum leads to a power spectrum more or less similar to that of the example I. Similarly to Heeger and Bergen's approach [6], color images are processed by decomposition into principal components and histogram equalization. As our noise permits an arbitrary PSD approximation, we are straightforwardly able to use a similar technique. Two examples, grass and carpet, are shown in figure 4. We only show the PSD of the first principal component of colors. Compared to [9] and [4], our approach allows us a more precise approximation of any arbitrary PSD. This increases the range of stochastic textures that can be dealt with. Nevertheless, a spectral and histogramm definition is not suitable for processing most types of textures [12]. In the following, we therefore propose procedural texture models that are not only based on a sum of noises but rather use multiple kernels noise as a tool to improve visual diversity and quality of textures. We propose a perturbation-based technique and a multi-layered texture definition.

Perturbation is a simple technique introduced by Perlin [13]. In consists in using noise to add randomness to well defined regular models, thus making them look more natural. Our multiple kernels noise allows us to improve this technique, by making the choice of the noise parameters automatic. Instead of directly defining a procedural texture $T(x)$ by noise, as in figure 4, we define it by using a basis function $P(x)$, *perturbed* by noise $n_S(x)$, i.e. $T(x) = P(x) + \lambda n_S(x)$, λ being the magnitude of perturbation. An easy and intuitive way to define P consists in painting it and then converting it into a functional representation such as polynomials (splines) and/or sums of cosines. P may also be directly painted using vector graphics tools. Once P has been defined, the user can further paint / supply an example

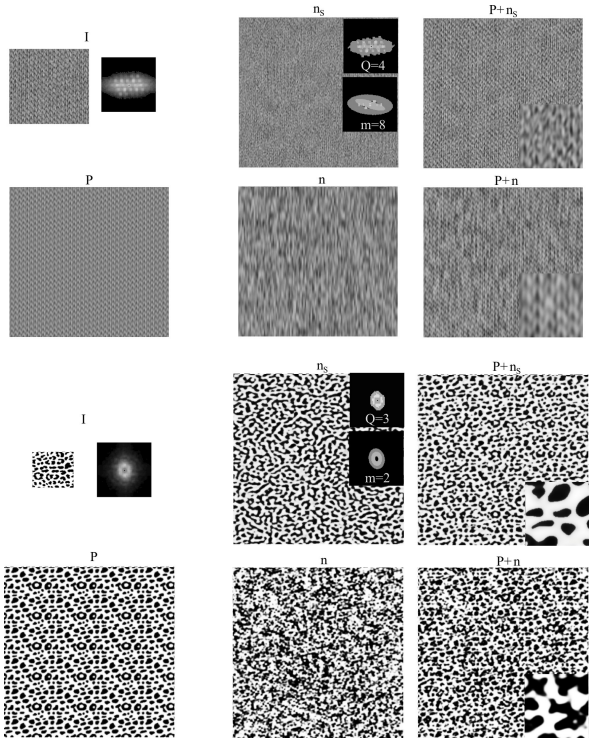

Fig. 5 Two examples of perturbation. I is the desired texture (with corresponding PSD), P is the user-defined regular pattern, n_S our multiple kernels noise computed from the PSD of I, $P + n_S$ a crop of the resulting texture, n a scaled Perlin noise and $P + n$ a texture using n. Our approach preserves spectral characteristics after perturbation (see $P+n_S$). The use of an arbitrary noise might not preserve these characteristics (see $P + n$).

of the desired perturbation, that is, a sample of the desired result $P + n_S$. We propose to use the PSD of this sample to directly define the PSD of n_S, which can be mathematically justified by assuming P and n_S are independent stationary signals (i.e. the cross-correlation is null). Figure 5 illustrates this principle. The first example is knitwear and the second one a binary structure (dots). For each example we compare our approach with the use of an empirically determined noise. For knitwear, P has been defined automatically by a sum of 25 cosines using the FFT. For the dots, the user has painted P by using vector graphics. As can be seen, multiple kernels noise allows us to preserve some visual characteristics, yet breaking regularity by adding randomness. Conversely, using an arbitrary noise perturbation does not necessarily provide good results. Here, we used a scaled Perlin noise [13].

Our multi-layered texture definition consists in expressing textures as hierarchical compositions of patterns at different levels, the lowest level being a *micro-texture*. A two-layered example is shown in figure 6. The top shows a crop of a satellite image of a coastal

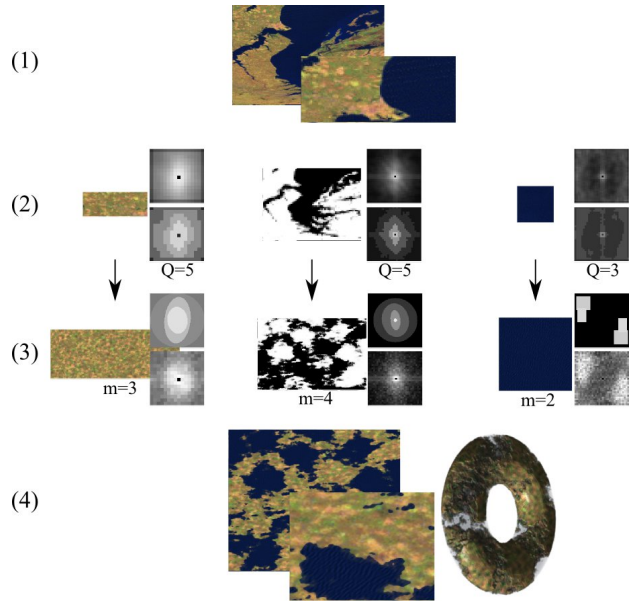

Fig. 6 Example of two-layered texture. We separate structural components from *micro-patterns* (row 2). Layers are represented using multiple kernels noise (row 3). Row 4: Recombining the layers to obtain a purely procedural texture that might be extended to 3D (see torus).

region. We illustrate how to produce a procedural texture that represents a landscape with similar color characteristics using only a few interactive manipulations combined with our automatic PSD decomposition. The top-level corresponds to the shape of the coast. We call it *structural layer*. On the land side, fields are visible. They represent one micro-texture, the other micro-texture being the blue ocean side. We model structural layers by a function $S(x)$ that is used as index $S(x) \mapsto \iota$ to associate a micro-texture $\mu_\iota(x)$ to position x. $\mu_\iota(x)$ provides the color information at x. In the case of two micro-textures, as in figure 6, a *step* function is applied to S to obtain $\iota \in 0, 1$. In fact, it is generally more convenient to use a *smoothed step* to make transitions between micro-textures smoother using a weighted sum. In row (2) the left and right sides show the two user-selected micro-textures and the middle, the binary image corresponding to the coast. It has been segmented using color quantization. This row represents the user's work. All other steps are then automatic. In row (3) the layers are processed as in figure 4, i.e. using directly multiple kernels noise. Note that when layers contain structural components, they must be rather processed using the previously described perturbation technique. Row (4) illustrates the final landscape texture $T(x)$ obtained by recombining the layers, i.e. $T(x) = smoothstep_{th}(S(x)) \cdot \mu_1(x) + (1 - smoothstep_{th}(S(x))) \cdot \mu_2(x)$. th is a threshold value selected by the user to define the proportions of land- and ocean-parts. Given that S, μ_1 and μ_2 are based on noises, $T(x)$ is completely procedural. It can also be easily extended to 3D by extending the kernels k to 3D. This is shown in row (4) on the torus object. The ocean has been made transparent to highlight the volumetric nature of the solid texture. The two-layered decomposition that we just described can be generalized to an arbitrary number of hierarchical layers. Each pattern inside a given layer can be further separated into sub-layers, and so on. This way complex multi-colored and multi-layered textures can be easily painted and composed by users.

6 Results

All results in this section have been obtained with a PC using a NVidia GeForce GTX 480. Figure 7 shows examples of multiple kernels noise functions created by using user defined PSDs. Column (a) is the input PSD with corresponding discrete noise sample. The user either defined directly the PSD (as for row 4) or supplied a discrete noise image. Our aim is to evaluate our spectral decomposition technique (column b), and the visual quality of the resulting noise functions using the sinc (left) and Gabor (right) kernels. This figure also compares our result with previous noise parameter fitting techniques. Column (c) illustrates the use of shader parameter fitting [1]. We defined a generic shader based on a weighted sum of scaled and rotated Perlin noises. The parameters of this shader are two scaling factors, the rotation angle and the weights. [1] attempts to find best parameter values using an iterative optimization technique based on a gradient descent. This approach generally required more than an hour for computing parameters. Because of the limited spectral control of Perlin noise, results are generally of low quality. Column (d) illustrates the approach of [9], which uses sums of Wavelet noises to model stochastic patterns. As expected, it fails for all anisotropic cases and does not allow one to consider complex spectral characteristics. Parameters are, however, computed in a few milli-seconds only. Column (e) illustrates [4], which uses sums of scaled and rotated Gabor noises (resulting in ellipses in spectral domain). Because of the use of Gabor noise, this approach allows one to create noises with more complex spectral characteristics. But the fact that the spectral domain is approximated by only one or two ellipses for each region, still strongly limits the complexity of PSDs that can be dealt with. Parameters required from 5 to 10 seconds to be computed. Our approach (column b) is able to deal with all types of PSDs. The decomposition into rectangles is very fast and requires less than a second. The iterative

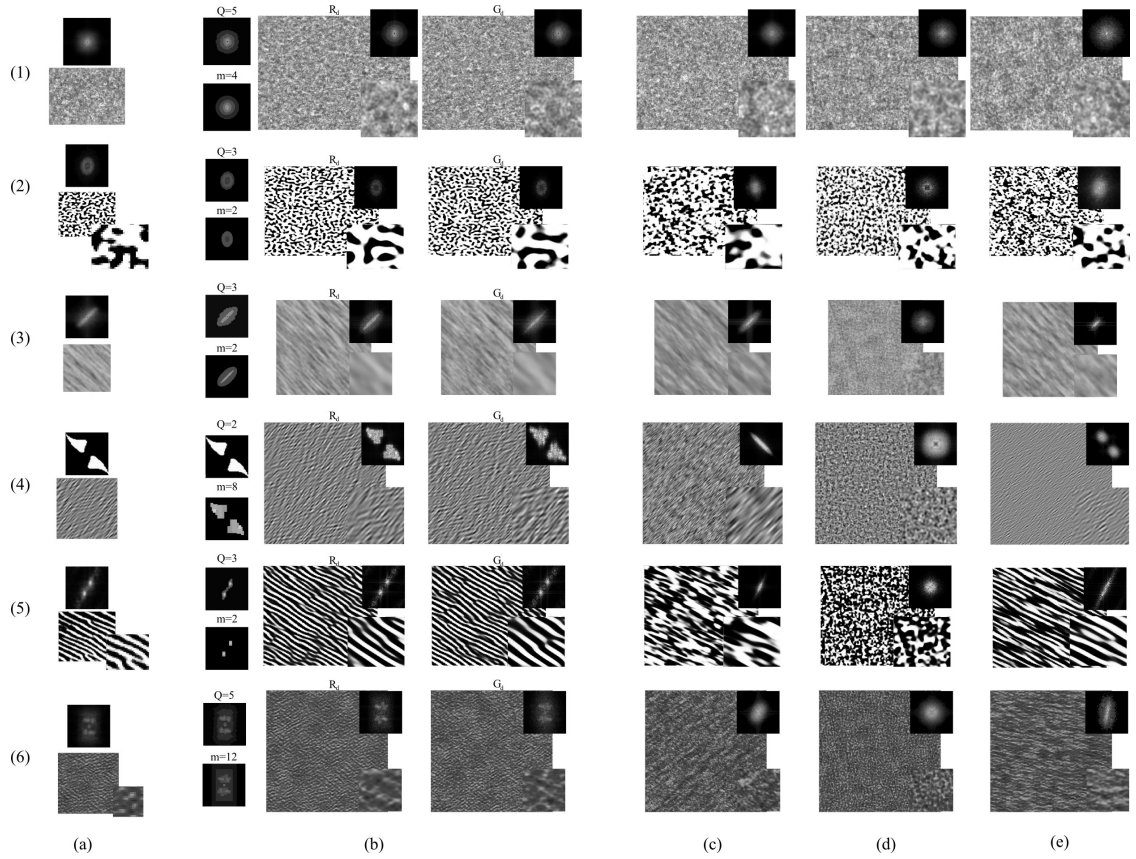

Fig. 7 Multiple kernels noise synthesis. (a) user supplied PSD and corresponding noise image. (b) approximation using our adaptive decomposition technique based on a sum of convolution noises using the sinc (left) and Gabor (right) kernels. (c) approximation using Gradient noise, parameters being computed using [1]. (d) approximation using wavelet noises [9]. (e) approximation using only one or two ellipses [4]. Unlike (a), (b)-(e) are infinite and continuous noise functions (see zooms demonstrating that there are no pixel artifacts).

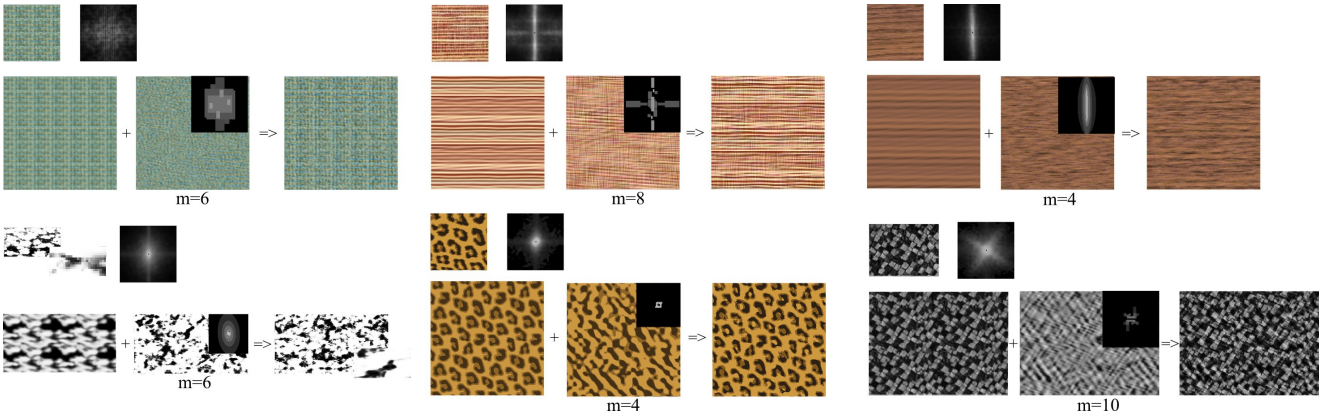

Fig. 8 Procedural textures based on multiple kernels noise perturbation.

w_i computation technique requires up to 20 seconds, depending on m. Looking more specifically at differences between the sinc and Gabor kernels, we can see that sometimes Gabor noise tends to diffuse energy, so that the result is perceived as too noisy. An additional difficulty related to the Gabor kernel is the choice of the σ_x and σ_y parameters. As already seen in figure 2, too small values introduce spectral errors, while too high values tend to diffuse energy. Finding a good compromise between energy diffusion and accuracy is not straightforward and depends on the PSD. When energy is concentrated on little spectral parts only, there are generally no visual differences between the two kernels (as for row 5 for instance).

Figure 8 illustrates the use of noise for defining procedural textures by *perturbation*. The top three examples, resp. representing knitwear, raffia weave and wood, use sums of resp. 40, 60 and 12 cosines for defining P. The three lower examples, resp. representing clouds, leopard skin and pavement, use splines for P. In all cases, our noise is computed automatically and allows us to add controlled randomness, while preserving main visual characteristics. Note that the resulting textures are continuous and free of pixel artifacts (see zoom on clouds).
Examples of procedural textures applied to 3D objects are shown in figures 9 and 10. In figure 10, (a),(b) and (c) illustrate a direct use of multiple kernels noise for bump texture synthesis and purely stochastic color texture synthesis. One key advantage of convolution noise is that is can be extended to higher dimensions by extending the kernels, as well as the point distributions, thus also extending the underlying procedural texture. The kernel scaling and position parameters can be copied from one of the two other dimensions. By using solid (3D) texturing, artifacts related to 2D surface parametrization, such as texture distortion and/or discontinuities, can be avoided as shown by the zoom on the neck of the bust (c). Subfigures (d), (e) and (f) illustrate texture synthesis using perturbation. Finally, (g), (h) and (i) show multi-layered textures. The corresponding micro-textures and structural layers were computed from photographs of materials and binary images.

All textures of this paper entirely fit into the fragment shader program and thus require no texture memory. On the performance side, a texture based on a single kernel (i.e. $m = 1$) leads to a performance of more than 600 frames per seconds for a framebuffer of size 800×800. However, framerate rapidly drops as texture complexity increases. With $m = 10$ the performance is 20 fps while with $m = 50$, the performance drops to 5 fps. Performances further decrease when bump mapping or displacement mapping is used. For instance, the gargoyle of figure 10(i) or the tree trunks depicted in figure 9, are displayed at 2-4 fps. They use on-the-fly tessellation for displacement mapping and three layers of color micro-textures. Low framerate is related to the fact that, as for all procedural shader-based textures, values are recomputed from scratch for each frame at rendered pixel level. Therefore, time-consuming procedural textures might not be suited for straightforward use in real-time applications. Computational requirements, but extremely low memory usage, might explain why procedural textures are currently rather used in high quality off-line rendering systems for media production.

Fig. 9 Complete scene where all objects are decorated with procedural textures. No texture memory is used. Because of the continuous definition of procedural textures, close zooms are possible without pixel-artifacts (see tree bark).

7 Conclusions

We have presented multiple kernels noise, a new noise function built from sums of convolution noises. One benefit of this noise is to facilitate the creation of procedural textures. In practice, users can define complex textures while avoiding the most experimental task, namely the determination of adequate noise parameters.
Our results show that the quality of multiple kernels noise is depending on whether or not there is a good match between the spectral characteristics of the kernel and the spectral energy inside the corresponding rectangular region. By exploring other types of kernels, we would like to reduce their amount without reducing accuracy. Indeed, one drawback of procedural textures is to trade memory consumption for higher computational resources. Some procedural textures are only hardly usable in real-time applications. Their main application remains high quality off-line rendering. Rich databases of procedural textures demonstrate their wide use. We hope our approach will help further growing the content of existing databases.

References

1. Bourque, E., Dudek, G.: Procedural texture matching and transformation. Computer Graphics Forum **23**(3), 461–468 (2004)
2. Cook, R.L., DeRose, T.: Wavelet noise. ACM Trans. Graph. **24**, 803–811 (2005)
3. Dischler, J., Ghazanfarpour, D.: A procedural description of geometric textures by spectral and spatial analysis of profiles. Computer Graphics Forum **16**(3), 129–139 (1997)
4. Gilet, G., Dischler, J.M., Soler, L.: Procedural descriptions of anisotropic noisy textures by example. In: Eurographics (Short) (2010)
5. Goldberg, A., Zwicker, M., Durand, F.: Anisotropic noise. In: ACM SIGGRAPH 2008, pp. 1–8

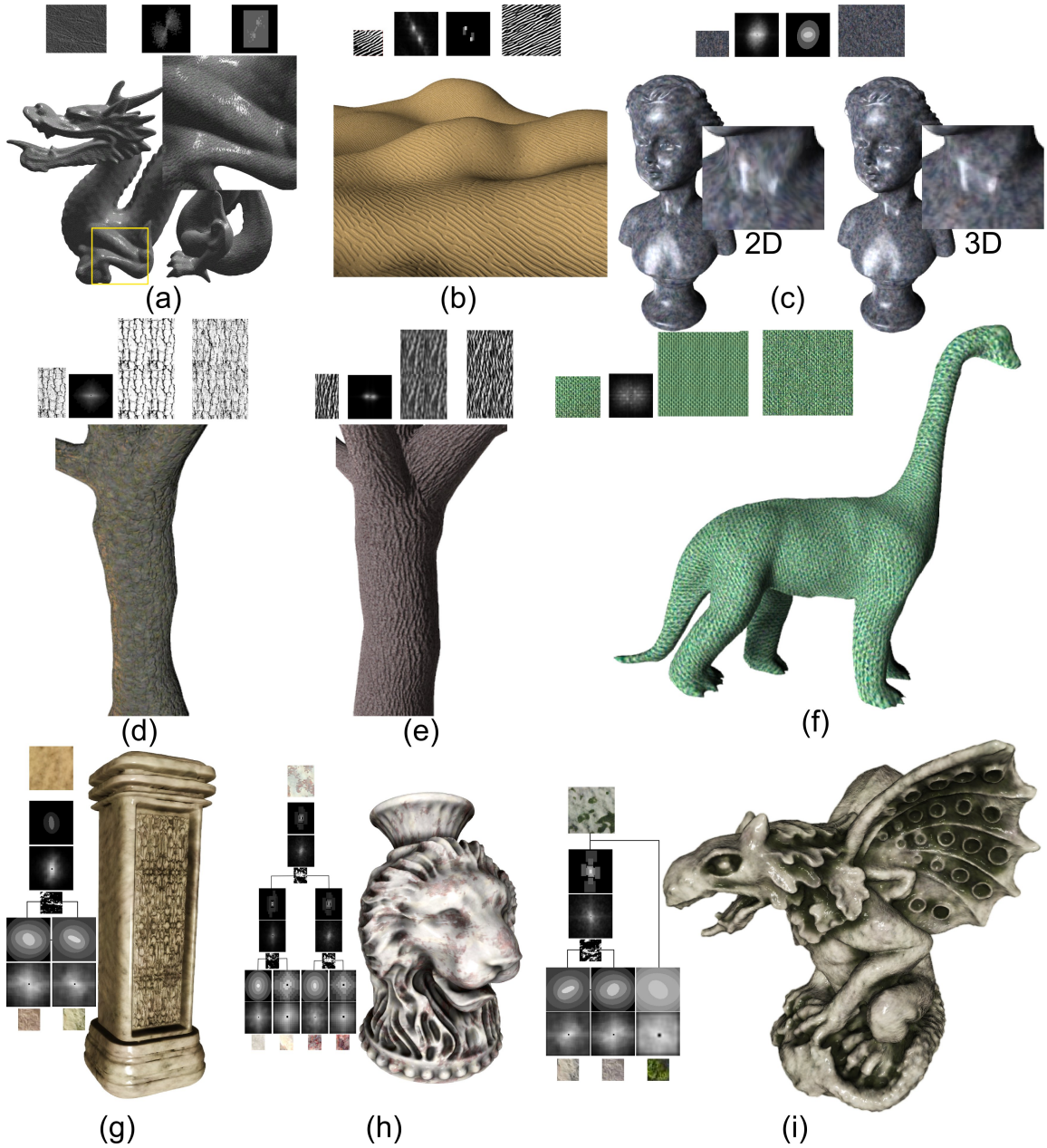

Fig. 10 Examples of procedural textures applied to 3D objects.

6. Heeger, D.J., Bergen, J.R.: Pyramid-based texture analysis/synthesis. In: SIGGRAPH, pp. 229–238 (1995)
7. 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), 2579–2600 (2010)
8. Lagae, A., Lefebvre, S., Drettakis, G., Dutré, P.: Procedural noise using sparse gabor convolution. In: SIGGRAPH 2009, pp. 1–10 (2009)
9. Lagae, A., Vangorp, P., Lenaerts, T., Dutré, P.: Procedural isotropic stochastic textures by example. Computers & Graphics **34**(4), 312–321 (2010)
10. Lewis, J.: Generalized stochastic subdivision. ACM Trans. Graph. **6**(3), 167–190 (1987)
11. Lewis, J.: Algorithms for solid noise synthesis. In: ACM Siggraph, pp. 263–270 (1989)
12. Navarro, R., Portilla, J.: Robust method for texture synthesis-by-analysis based on a multiscale gabor scheme. In: SPIE, pp. 86–97 (1996)
13. Perlin, K.: An image synthesizer. In: ACM Siggraph, pp. 287–296 (1985)
14. Perlin, K.: Noise hardware. In: M. Olano (ed.) Real-Time Shading SIGGRAPH Course Notes (2001)
15. Perlin, K.: Improving noise. ACM Trans. Graph. **21**(3), 681–682 (2002)
16. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report (2009)
17. Worley, S.: A cellular texture basis function. In: SIGGRAPH 96, pp. 291–294 (1996)