*To appear in the ACM SIGGRAPH conference proceedings*

# Procedural Noise using Sparse Gabor Convolution*

Ares Lagae† Sylvain Lefebvre‡ George Drettakis‡ Philip Dutré†
Katholieke Universiteit Leuven† REVES / INRIA Sophia-Antipolis‡

**Figure 1:** *(Left) We present a procedural noise that offers accurate spectral control. The user can interactively manipulate the power spectrum. (Middle) We apply the noise to a surface without the need for texture coordinates, and provide high-quality anisotropic filtering. Thanks to increased expressiveness of the noise, a simple colormap is enough to produce visually interesting textures. (Right) Since our surface noise does not require a texture parameterization, the surface can evolve dynamically and even sustain topology changes.*

## Abstract

Noise is an essential tool for texturing and modeling. Designing interesting textures with noise calls for accurate spectral control, since noise is best described in terms of spectral content. Texturing requires that noise can be easily mapped to a surface, while high-quality rendering requires anisotropic filtering. A noise function that is procedural and fast to evaluate offers several additional advantages. Unfortunately, no existing noise combines all of these properties.

In this paper we introduce a noise based on sparse convolution and the Gabor kernel that enables all of these properties. Our noise offers accurate spectral control with intuitive parameters such as orientation, principal frequency and bandwidth. Our noise supports two-dimensional and solid noise, but we also introduce setup-free surface noise. This is a method for mapping noise onto a surface, complementary to solid noise, that maintains the appearance of the noise pattern along the object and does not require a texture parameterization. Our approach requires only a few bytes of storage, does not use discretely sampled data, and is nonperiodic. It supports anisotropy and anisotropic filtering. We demonstrate our noise using an interactive tool for noise design.

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

**Keywords:** noise, procedural texture, rendering, shading

## 1 Introduction

Visual complexity is a key ingredient of compelling computer generated imagery. Randomness, or noise, was identified early on as a useful tool for generating that visual complexity. Since its introduction by Perlin [1985], noise has become an essential modeling tool in computer graphics. It is mainly used in procedural modeling and texturing [Ebert et al. 2002], but also in other areas of computer graphics, such as animation and simulation [Bridson et al. 2007].

Noise is used in many different contexts, and should therefore be as versatile as possible. This is witnessed by the *procedural* nature of noise, which is probably one of its most significant advantages: it is compact, described by only a few parameters, yet it can be quickly evaluated at any point in space. This makes it very suitable for rendering. The random nature of noise means that it is best described in terms of spectral content or energy in each frequency band. Controlling the appearance of noise, e.g., for modeling, therefore requires *spectral control*. Noise is mainly used for adding details to surfaces. This is typically achieved by traditional texture mapping using a two-dimensional noise, or using a solid noise. Solid noise is not subject to artifacts, such as distortions and seams, and is *setup free*. This means that surfaces can be textured without preprocessing, such as a texture parameterization, regardless of complexity. Setup-free surface noise is not only simple and convenient, but also essential, when surfaces are implicit, point sampled or dynamic. High-quality rendering of noise requires anisotropic filtering to avoid artifacts. This requires noise to be *anisotropic*. Of course, noise should also be *fast to evaluate*.

From the above, it is clear that we want a procedural noise with accurate spectral control, providing setup-free surface texturing. In addition it should be anisotropic and allow high-quality anisotropic filtering. Many previous noises have a subset of these properties (Sec. 1.1); none of them however provides them all. In this paper we introduce a noise that *does* have them all:

- We introduce a new procedural noise with accurate spectral control using Gabor kernels, by extending sparse convolution noise [Lewis 1989]. Our noise has a memory footprint of only a few bytes, is nonperiodic and is anisotropic.
- We introduce setup-free surface noise with high-quality anisotropic filtering, a method for mapping noise onto a surface that does not require a texture parameterization. Surface noise is complementary to solid noise: it maintains the aspect of a two-dimensional noise pattern along the object. Solid noise is supported as well.
- We demonstrate a powerful interactive design tool for noises and procedural textures, based on the intuitive properties of our noise (frequency, orientation and bandwidth).

Our new procedural noise is illustrated in Fig. 1 using our interactive tool for noise design. First, we create an anisotropic noise. The inset displays the widget for spectral control and the main image displays the corresponding noise. Next, we add an isotropic noise. Then, we create a procedural texture by combining the noise with

*with corrections (November 2009)

†e-mail: {ares.lagae,philip.dutre}@cs.kuleuven.be
‡e-mail: {sylvain.lefebvre,george.drettakis}@sophia.inria.fr

a color map, and we render a surface with the procedural texture using anisotropic filtering. Finally, we apply the noise to an impicit surface whose topology can change interactively.

### 1.1 Related Work

We group the desirable properties of noise into five categories, and discuss related work as a function of these properties.

**Procedural** Perlin noise [Perlin 1985] is likely the most popular noise in computer graphics. Key to the success of Perlin noise is that it is procedural. According to the advantages of a procedural representation identified by Ebert et al. [2002], noise should have compact storage requirements, should not be based on discretely sampled data, should be nonperiodic, and should be intuitively parameterized. These properties imply that noise should be evaluated on the fly rather than stored.

**Spectral Control** Noise is a modeling tool, and therefore controlling the appearance of noise is of key importance. The power spectrum of the noise, which describes the contribution of each frequency band, is a powerful and natural framework for controlling it [Perlin 1985; Lewis 1989; Perlin and Hoffert 1989]. Perlin noise achieves spectral control by using a weighted sum of band-limited octaves of noise. However, Cook and DeRose [2005] showed that Perlin noise is only approximately band limited, and is therefore subject to aliasing and detail loss. Although Perlin repeatedly improved his noise [Perlin 2002; Olano et al. 2002], this problem was not addressed. Therefore, most recent work focuses on constructing band-limited noise [Cook and DeRose 2005; Kensler et al. 2008]. However, band-limited noise is not a goal by itself, it is only a means for controlling the power spectrum of the noise. Moreover, a weighted sum of octaves of noise does not provide the desired spectral control. This was already recognized by Lewis [1989], who introduced solid procedural noises with improved control over the power spectrum.

**Surface Noise** Noise is commonly used for texturing surfaces. Traditional texturing requires a surface parameterization, which usually introduces distortions and discontinuities. This means that surface noise cannot always be obtained by simply mapping a two-dimensional noise texture onto a surface [Goldberg et al. 2008]. Perlin [1985] and Peachy [1985] introduced solid texturing, an attractive alternative for traditional texturing not subject to these problems. Solid texturing does not require a surface parameterization and is setup free. Therefore, noise on a surface is typically obtained using solid noise, even when not modeling a solid material. However, Cook and DeRose [2005] showed that mapping band-limited solid noise onto a surface does not result in band-limited surface noise. This means that surface noise also can not be obtained by simply mapping solid noise onto a surface. Moreover, solid noise and surface noise have a distinct appearance. Noise should therefore differentiate between two-dimensional noise, surface noise and solid noise, and, similar to solid noise, surface noise should be setup free.

**Anisotropy** High-quality rendering requires anisotropic filtering to avoid aliasing. Band-limited isotropic noise can be isotropically filtered using frequency clamping, that is by disabling octaves in the sum of weighted octaves that would result in aliasing [Cook and DeRose 2005]. Hart et al. [1999] addressed isotropic filtering of procedural solid textures by prefiltering the color lookup table. However, Goldberg et al. [2008] showed that anisotropic filtering requires anisotropic noise. Anisotropic noise is also useful to model anisotropic phenomena.

**Fast to Evaluate** Noise was introduced in the context of off-line rendering, but has recently become more accessible to interactive applications [Hart 2001; Olano 2005; Frisvad and Wyvill 2007; Goldberg et al. 2008]. Noise is used in many different contexts, ranging from games to production rendering. Therefore, noise should be interactive and enable a speed vs. quality trade-off.

None of the existing noises has all of these properties, which is illustrated in Tab. 1 (Sec. 7). For example, wavelet noise [Cook and DeRose 2005] and the noise of Kensler et al. [2008] are band limited and support setup-free surface noise, but do not support anisotropic noise or anisotropic filtering, and the noise of Goldberg et al. [2008] supports anisotropic noise and anisotropic filtering, but does not support solid noise or setup-free surface noise.

In the following sections, we first explain how the Gabor kernel combined with sparse convolution produces band-limited anisotropic noise with accurate spectral control (Sec. 2). We then generalize to more complex spectra using a random sampling of the parameters of the Gabor kernels during sparse convolution (Sec. 3). We next discuss procedural, on-the-fly noise evaluation (Sec. 5), and detail our anisotropically filtered setup-free surface noise. Results and comparisons are given in Sec. 6 and Sec. 7.

## 2 Band-Limited Anisotropic Noise

In this section we present our procedural band-limited anisotropic noise. We review sparse convolution noise (Sec. 2.1), generalize sparse convolution noise as a random pulse process (Sec. 2.2), show that our choice of Gabor kernel as a pulse is well motivated (Sec. 2.3), and formulate our anisotropic noise as a random pulse process using a Gabor kernel (Sec. 2.4). We assume that the reader is familiar with Fourier analysis (e.g., Bracewell [1999]).

### 2.1 Sparse Convolution Noise

In 1989, Lewis introduced sparse convolution noise, a procedural noise that offers improved control over the power spectrum. Lewis defines sparse convolution noise $N$ as a convolution of a kernel $h$ and a Poisson impulse process

$$N(x,y) = \left[ h * \sum_i w_i \delta_{(x_i,y_i)} \right](x,y), \tag{1}$$

where $*$ denotes convolution, $\delta$ is the impulse and $\delta_{(x_i,y_i)}(x,y) = \delta(x-x_i, y-y_i)$. The Poisson impulse process consists of impulses of uncorrelated intensity $\{w_i\}$ at uncorrelated locations $\{(x_i,y_i)\}$. Lewis calls this process *sparse white noise*, because it consists of only a few scattered impulses and has a constant power spectrum. The power spectrum of sparse convolution noise is that of the kernel scaled by a constant, which follows from the Convolution Theorem. Lewis used the radially symmetric smooth cosine kernel

$$h(r) = 1/2 + 1/2\cos(\pi r) \quad (|r| < 1). \tag{2}$$

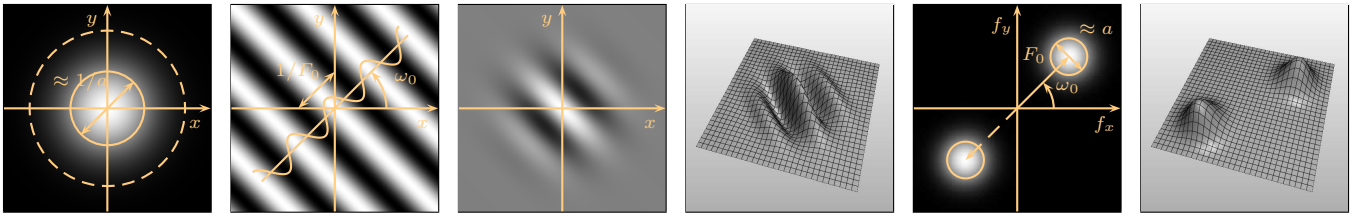
Lewis procedurally evaluates sparse convolution noise by introducing a grid, and generating the positions and weights of the kernels in each cell on the fly. The grid reduces the evaluation of the noise to the grid cells close to the point of evaluation. An example of sparse convolution noise is shown in Fig. 8(b).

### 2.2 The Random Pulse Process

Sparse convolution noise is a random pulse process in which the pulse is the radially symmetric smooth cosine kernel. The random pulse process $Y$ is defined as a sum of randomly weighted and positioned pulses

$$Y(x) = \sum_i w_i h(x - x_i), \tag{3}$$

where $h$ is the pulse, the random locations $\{x_i\}$ are governed by a Poisson distribution with mean $\lambda$, called the impulse density, and the weights $\{w_i\}$ are realizations of a random variable $W$. In this work, we assume that the random variable $W$ has zero mean. Note




(a) Gaussian envelope. (b) Harmonic. (c) Gabor kernel. (d) 3D plot of (c). (e) Fourier transform. (f) 3D plot of (e).

**Figure 2:** *The Gabor kernel. (a) Gaussian envelope. (b) Harmonic. (c) Gabor kernel. (d) Gabor kernel, 3D plot. (e) Fourier transform of Gabor kernel. (f) Fourier transform of Gabor kernel, 3D plot.*

that the location of the impulses follows a uniform random distribution, and therefore, the number of impulses per unit area, or impulse density $\lambda$, follows a Poisson distribution. Poisson processes and shot noise, the family of stochastic processes that the random pulse process belongs to, are well known in statistics [Papoulis and Pillai 2002, 10.2; van Etten 2005, 8].

The random pulse process has zero mean, and the variance $\sigma_Y^2$ of the process is

$$\sigma_Y^2 = \lambda E\left[W^2\right] \int_{-\infty}^{\infty} h^2(x)\, dx, \tag{4}$$

where $E$ denotes expectation value. The power spectrum of the process is

$$S_{YY}(f) = \lambda E\left[W^2\right] |H(f)|^2, \tag{5}$$

where $H$ is the Fourier transform of the pulse $h$. The total power equals $\sigma_Y^2$, which follows from Rayleigh's Theorem. The amplitude distribution of shot noise tends to a normal distribution as the impulse density $\lambda$ increases [Papoulis 1971]. Therefore, the amplitude distribution of the process can be approximated by a normal distribution with zero mean and variance $\sigma_Y^2$.

### 2.3 The Gabor Kernel

The key to accurate control over the power spectrum of the random pulse process is the choice of an appropriate pulse. Our main insight is that the pulse should be parameterized and have compact support in the spatial domain to enable efficient procedural evaluation, and also have compact support in the frequency domain to enable precise control over the power spectrum. We therefore chose the Gabor kernel [Gabor 1946] which has all of these properties. In some sense, this is the theoretically optimal choice of pulse in this context, since Gabor showed that the kernel has the smallest joint uncertainty in both domains.

The Gabor kernel is a multiplication of a Gaussian envelope and a harmonic (Fig. 2(a)-(d)). We use a two-dimensional real Gabor kernel of the cosine type, parameterized by magnitude, frequency, and width. The Gabor kernel $g$ is

$$g(x,y) = Ke^{-\pi a^2\left(x^2+y^2\right)} \cos\left[2\pi F_0\left(x\cos\omega_0 + y\sin\omega_0\right)\right], \tag{6}$$

where $K$ and $a$ are the magnitude and width of the Gaussian envelope, and $(F_0, \omega_0)$ is the frequency of the harmonic in polar coordinates. $F_0$ and $\omega_0$ are the magnitude and the orientation of the frequency. The Fourier transform of the Gabor kernel $G$ is

$$\begin{aligned} G(f_x, f_y) = \frac{K}{2a^2}\Big\{ & e^{-\frac{\pi}{a^2}\left[(f_x - F_0\cos\omega_0)^2 + (f_y - F_0\sin\omega_0)^2\right]} \\ & + e^{-\frac{\pi}{a^2}\left[(f_x + F_0\cos\omega_0)^2 + (f_y + F_0\sin\omega_0)^2\right]}\Big\}, \end{aligned} \tag{7}$$

where $(F_0\cos\omega_0, F_0\sin\omega_0)$ is the frequency of the harmonic in Cartesian coordinates. The Fourier transform consists of two identical Gaussians, one located at the frequency of the harmonic, and its symmetric counterpart (Fig. 2(e)-(f)), which follows from the Convolution Theorem.

The parameters of the Gabor kernel have an intuitive meaning in both domains. The frequency $(F_0, \omega_0)$ of the harmonic corresponds to the location of the Gaussian in the frequency domain, and the width $a$ of the Gaussian envelope corresponds to the width of the Gaussian in the frequency domain. The frequency $(F_0, \omega_0)$ can therefore be interpreted as the principal frequency, and the width $a$ as the bandwidth, the range of frequencies around the principal frequency. The full width at half maximum, a parameter commonly used to describe the width of a function, of the Gaussian envelope and the Gaussian in the frequency domain is approximately $1/a$ (Fig. 2(a)) and $a$ (Fig. 2(e)).

Gabor kernels were introduced in 1946 by Gabor to study communication theory. Gabor kernels are frequently used in computer vision, e.g., for texture analysis and synthesis [Navarro and Portilla 1996], as well as in perception.

### 2.4 Band-Limited Anisotropic Noise

We can now define our new band-limited anisotropic noise $N$ as a random pulse process, with the Gabor kernel $g$ as a pulse

$$N(x,y) = \sum_i w_i g(x - x_i, y - y_i). \tag{8}$$

We derive the properties of this band-limited anisotropic noise from the properties of the random pulse process and the Gabor kernel. The noise has zero mean, and the variance of the noise is (Eq. (4))

$$\sigma_N^2 = \lambda E\left[W^2\right] \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g^2(x,y)\, dx\, dy. \tag{9}$$

The power spectrum of the noise is (Eq. 5)

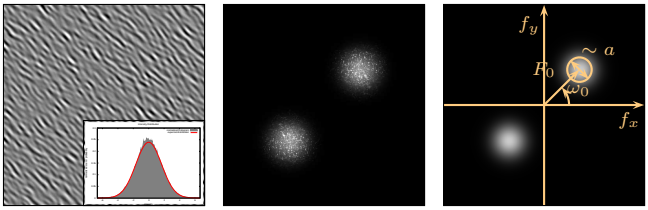
$$S_{NN}(f_x, f_y) = \lambda E\left[W^2\right] |G(f_x, f_y)|^2. \tag{10}$$

The intensity distribution of the noise can be approximated by a normal distribution with zero mean and variance $\sigma_N^2$.

The power spectrum of band-limited anisotropic noise is the power spectrum of the Gabor kernel scaled by the constant $\lambda E\left[W^2\right]$. This means that we can control the power spectrum of the noise using the intuitive parameters of the Gabor kernel. The magnitude $K$, the orientation $\omega_0$ and the magnitude $F_0$ of the frequency, and the width $a$ of the Gabor kernel correspond to the magnitude, orientation, principal frequency and bandwidth of the noise. An example of band-limited anisotropic noise is shown in Fig. 3. The accompanying video illustrates how the individual Gabor kernels are combined to form the noise.

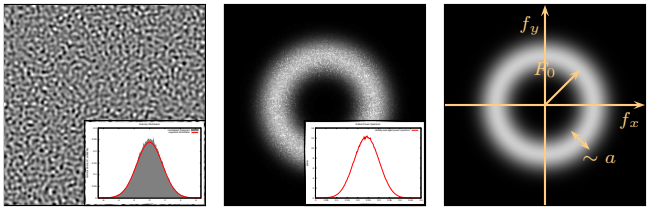
## 3 Noise with On-The-Fly Spectral Control

We achieve spectral control by using a different Gabor kernel for each impulse, with parameters randomly sampled from a parametric description of the desired power spectrum. This is a very powerful way to generalize the noise defined by Eq. (8), allowing us to define band-limited isotropic noise (Sec. 3.1) and noise with controllable band-limits (Sec. 3.2). Using our method, it is also easy to develop a system for interactive noise design (Sec. 3.3).




(a) Noise. (b) Power spectrum. (c) Analyt. pow. spec.

**Figure 3:** *Band-limited anisotropic noise. (a) Anisotropic noise with intensity histogram (gray) and expected intensity distribution (red). (b) Power spectrum. (c) Analytical power spectrum.*




(a) Noise. (b) Power spectrum. (c) Analyt. pow. spec.

**Figure 4:** *Band-limited isotropic noise. (a) Isotropic noise with intensity distribution. (b) Power spectrum with radially averaged power spectrum. (c) Analytical power spectrum.*

### 3.1 Band-Limited Isotropic Noise

We define our new band-limited *isotropic* noise $N$ as a random pulse process

$$N(x,y) = \sum_i w_i g(x - x_i, y - y_i; \omega_{0,i}), \tag{11}$$

where the pulse is a Gabor kernel with a random frequency orientation, and the frequency orientations $\{\omega_{0,i}\}$ are realizations of a random variable with a continuous uniform distribution on the interval $[0, 2\pi)$.

Similar to band-limited anisotropic noise, we derive the properties of this band-limited isotropic noise from the properties of the random pulse process and the Gabor kernel. The power spectrum of the noise, obtained by integrating the pulse over all possible orientations [van Wijk 1991], is

$$S_{NN}(f_x, f_y) = \lambda E\left[W^2\right] \frac{1}{2\pi} \int_0^{2\pi} |G(f_x, f_y; \omega_{0,i})|^2 \, d\omega_0. \tag{12}$$

The total power of the noise is equal to the total power of band-limited anisotropic noise. Therefore, the variance of the noise is equal to the variance of band-limited anisotropic noise. The intensity distribution of the noise can therefore also be approximated by a normal distribution with zero mean and variance $\sigma_N^2$.

The power spectrum of band-limited isotropic noise is radially symmetric, because the noise is isotropic. In terms of radial frequency $f_r^2 = f_x^2 + f_y^2$, the power spectrum is

$$S_{NN}(f_r) = \lambda E\left[W^2\right] \frac{K^2}{2a^4} e^{\frac{-2\pi}{a^2}\left(f_r^2 + F_0^2\right)} \left[1 + I_0\left(\frac{4\pi F_0}{a^2} f_r\right)\right], \tag{13}$$

where $I_0$ is the modified Bessel function of the first kind. By approximating the Bessel function $I_0$ with an exponential [Abramowitz and Stegun 1964, 9.7.1], and because the region of interest of the power spectrum is the region near the principal frequency ($f_r \approx F_0$), we can approximate the power spectrum by

$$S_{NN}(f_r) \approx \lambda E\left[W^2\right] \frac{K^2}{4\sqrt{2}\pi F_0 a^3} e^{\frac{-2\pi}{a^2}(f_r - F_0)^2}. \tag{14}$$

Similar to band-limited anisotropic noise, this is a Gaussian with a principal frequency of $F_0$ and a width of $a$ (inset in Fig. 4(b)).

We control the power spectrum of band-limited isotropic noise in the same way as the power spectrum of anisotropic noise, although the orientation of the frequency $\omega_0$ is integrated out. Our band-limited isotropic noise is the first noise that allows precise control over the principal frequency and the bandwidth of the noise. An example of band-limited isotropic noise is shown in Fig. 4. Note how a ring is formed in the frequency domain due to the integration of the orientation.

### 3.2 Noise with Controllable Band-Limits

We generalize band-limited anisotropic and isotropic noise to noise with controllable band-limits, which has a power spectrum that covers a specific region of frequency space.

A region of frequency space can be described in polar coordinates by an annular sector $[F_{0,min}; F_{0,max}) \times [\omega_{0,min}; \omega_{0,max})$ (see Fig. 7 for examples). This region contains all frequencies with a magnitude between $F_{0,min}$ and $F_{0,max}$, and an orientation between $\omega_{0,min}$ and $\omega_{0,max}$. By randomly sampling the parameters $F_0$ and $\omega_0$ of the Gabor kernel associated with each impulse, a noise with a power spectrum equal to this region of frequency space is obtained. The bandwidth parameter $a$ controls the fall-off of the transition near the edge of the annular sector.

Noise with controllable band-limits generalizes band-limited anisotropic noise ($F_{0,min} = F_{0,max} = F_0$, $\omega_{0,min} = \omega_{0,max} = \omega_0$), band-limited isotropic noise ($F_{0,min} = F_{0,max} = F_0$, $\omega_{0,min} = 0, \omega_{0,max} = 2\pi$), but also includes band-limited isotropic noise with a precise control over the frequency range $[F_{0,min}, F_{0,max})$ and the fall-off $a$ of the band-limit, and non-band-limited noise ($F_{0,min} = F_{0,max} = 0$).

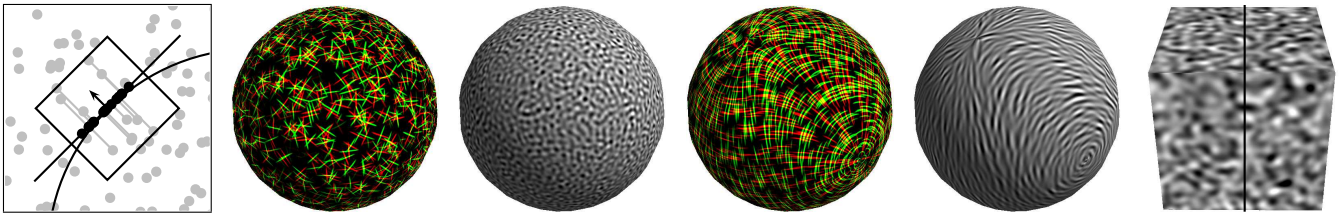
### 3.3 Interactive Noise Design

We have developed an interactive tool for noise design. The tool is based on widgets for band-limited anisotropic and isotropic noise, noise with controllable band-limits, and linear combinations of these. The widgets allow the user to manipulate the power spectrum of the noise on diagrams similar to the ones shown in Fig. 3(c) and 4(c). We have observed that this results in straightforward interaction. The best way to appreciate the interactive tool and widgets is to watch the accompanying video.

## 4 Procedural Evaluation

The procedural evaluation of our noise is similar to that of sparse convolution noise [Lewis 1989] and cellular noise [Worley 1996]. Similar to Lewis (see Sec. 2.1) and Worley, we introduce a grid, and generate the properties of the Gabor kernels in each cell on-the-fly using a pseudo-random number generator. The cell size of the grid equals the radius of the Gabor kernels, which reduces the evaluation of the noise to the cell containing the point of evaluation and the direct neighboring cells. In this section we discuss these components in more detail. Pseudocode is provided in the appendix, and compilable C++ example code is provided in the auxiliary material.

**Grid** In order to introduce a grid, we truncate the Gabor kernel, since it has an infinite support. We truncate the kernel where the Gaussian envelope reaches 5% of its peak value $K$, which corresponds to a kernel radius of roughly $1/a$ (dashed line in Fig. 2(a)). This introduces a small discontinuity in the noise, which in our experience does not affect the visual appearance, even when the noise derivative is used [Perlin 2002]. If required, this discontinuity could be alleviated by truncating at a larger kernel radius, or avoided by multiplying with a continuous window function. Note that, in contrast to the other parameters of the noise, the spatial variation of the bandwidth is restricted, because the cell size is coupled to the width of the Gabor kernel.



(a) Projection. (b) Iso. surf. frames. (c) Iso. surf. noise. (d) Aniso. surf. frames. (e) Aniso. surf. noise. (f) Discontinuity.

**Figure 5:** *Setup-free surface noise. (a) Projection of the three-dimensional Poisson distribution onto the tangent plane. (b) Randomly oriented local surface frames for isotropic surface noise. (c) Isotropic surface noise. (d) Local surface frames aligned with a vector field for anisotropic surface noise. (e) Anisotropic surface noise. (f) The discontinuous face normals used on the left result in a noise discontinuity, while the smooth vertex normals used on the right ensure continuity.*

**Pseudo-Random Number Generator** The properties of the Gabor kernels in each cell are generated on-the-fly using a pseudo-random number generator. We use the linear congruential generator

$$x_{n+1} = 3039177861\, x_n \pmod{2^{32}}, \tag{15}$$

introduced by Borosh and Niederreiter [1983]. This generator is fast, and, despite its simplicity, optimal with respect to statistical independence of successive pseudo-random numbers.

**Seeding Strategy** To ensure consistency, the pseudo-random number generator associated with a cell always has to be initialized with the same seed. If a periodic noise with period $N$ is required, we determine the seed for the cell with coordinates $(x, y)$ as

$$\text{seed}\,(x, y) = ((y\%N)\,N) + (x\%N) + o, \tag{16}$$

which corresponds to enumerating the cells in row-major order. If a nonperiodic noise is required, we use Morton order [Morton 1966]. This seeding strategy ensures that the noise always returns the same value when evaluated at the same location. The noise is randomized by adding a global random offset $o$. In contrast to Lewis and Worley, we do not use a random seeding strategy. This is not required because the sequences of pseudo-random numbers produced for every seed are uncorrelated.

**Cell and Kernel Properties** The number of impulses in a cell is randomly generated according to a Poisson distribution. We use the algorithm of Knuth for small means [Knuth 1997, 3.4.1]. We express the impulse density $\lambda$ in expected number of impulses $N$ per kernel area $\pi r^2$, which corresponds to an impulse density of $N/\pi$ impulses per cell. The properties associated with the Gabor kernels, such as their position and weight, are randomly generated according to the corresponding probability distributions. For the weights, we use a random variable $W$ with a continuous uniform distribution on the interval $[-1, +1]$. All random numbers are generated using the properly seeded pseudo-random number generator associated with the cell.

We have implemented the procedural evaluation for the CPU and GPU. Our current GPU implementation is a straightforward mapping of the procedural evaluation to a GPU pixel shader. A faster GPU implementation could be obtained by exploiting spatial coherence using recent GPU architectures, rather than independently evaluating noise in every pixel. It might also be worthwile to investigate the use of random number generators designed for the GPU [Tzeng and Wei 2008].

Our noise has a memory footprint of only 32 bytes (see example code). Our procedural evaluation allows a speed vs. quality tradeoff by adapting the impulse density and by using approximate transformation methods. We typically use 100 impulses per kernel in our CPU implementation, although good results are already obtained at lower impulse densities, and 25 to 50 impulses per kernel in our GPU implementation (values used in all video examples).

We have validated our procedural CPU and GPU implementations by verifying that the power spectrum converges to the analytical power spectrum (Fig. 3 and 4), and by comparing the results to those of a non-procedural CPU implementation based on frequency filtering of white noise (Fig. 8(f)-(h)).

## 5 Anisotropically Filtered Surface Noise

Our noise generalizes to arbitrary dimensions. However, many applications require noise on a surface, and high-quality rendering requires anisotropic filtering. In this section we therefore present surface noise (Sec. 5.1) that is anisotropically filtered (Sec. 5.2).

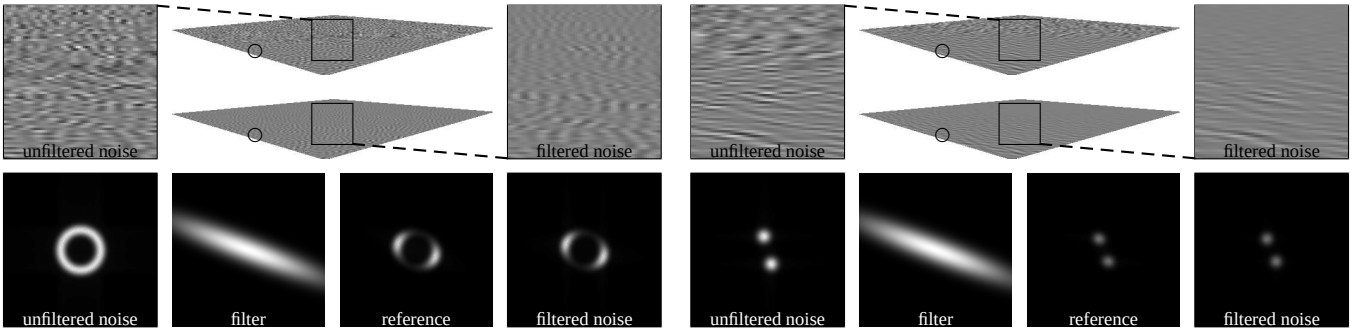
### 5.1 Setup-Free Surface Noise

Cook and DeRose [2005] obtain surface noise by performing a weighted line integral of solid wavelet noise along the direction perpendicular to the surface. Goldberg et al. [2008] obtain surface noise by mapping two-dimensional noise onto the surface and compensating for distortion introduced by the texture parameterization. Our noise is compatible with both approaches, but also admits a direct formulation of surface noise.

Similarly to Lefebvre et al. [2005], we locally project kernels onto the surface. We obtain surface noise at a point on a surface $p$ with surface normal $n$ by projecting a three-dimensional Poisson distribution on the plane determined by $p$ and $n$, and evaluating our two-dimensional noise in that plane (Fig. 5(a)). Isotropic surface noise is obtained by randomly orienting the local surface frame of each Gabor kernel (Fig. 5(b)-(c)), and anisotropic surface noise by aligning the local surface frame with a vector field that guides the anisotropy (Fig. 5(d)-(e)). The two-dimensional Poisson distribution in the tangent plane with impulse density $\lambda$ is obtained by projecting all points of a three-dimensional Poisson distribution with impulse density $\lambda/2r$ inside the cylinder with radius $r$, height $2r$, center $p$ and orientation $n$ onto the plane. The points are generated on-the-fly, similar as in Sec. 4, using a three-dimensional grid. The weight of a point is defined as one minus the distance to the plane divided by $r$, which avoids the generation of an additional random number. Note that this method is similar in spirit to the method of Cook and DeRose, but is more efficient because it avoids the explicit integration.

Our isotropic surface noise is setup free, and is not tied to a specific geometry representation. Our anisotropic surface noise is setup free as well. We generate the vector field that guides the anisotropy procedurally by computing a local surface frame from the surface normal and a global direction, but the vector field can also be designed by the user [Zhang et al. 2007; Fisher et al. 2007].

Our surface noise implicitly assumes that texture detail is small compared to geometric detail, a common assumption in texturing. A limitation of surface noise is that a discontinuity in the surface normal results in a discontinuity in the noise. If this behavior is not




(a) Isotropic noise. (b) Anisotropic noise.

**Figure 6:** *Anisotropic filtering. (a) Isotropic noise. (b) Anisotropic noise. (Top row) Tilted plane with unfiltered and anisotropically filtered noise with close-ups. (Bottom row) Power spectrum of the unfiltered noise, frequency domain filter in texture space, reference power spectrum and power spectrum of the anisotropically filtered noise of the circled pixel.*

desired, then a continuous normal should be used instead (Fig. 5(f)).

### 5.2 Anisotropic Filtering

Anisotropic texture filtering can be formulated as a convolution of a texture with a filter in the spatial domain, or as a multiplication in the frequency domain. We exploit the spectral control of our noise to achieve high-quality anisotropic filtering of surface noise.

We use the analytic formulation of anisotropic filtering of Heckbert [1989]. The filter in image space is the Gaussian

$$f(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}\left(x^2+y^2\right)}, \tag{17}$$

where $\sigma$ is the width of the Gaussian. The corresponding frequency domain filter in texture space is the Gaussian

$$F\left(J^T\,[f_u\ f_v]^T\right) = e^{-2\pi^2\sigma^2[f_u\ f_v]JJ^T[f_u\ f_v]^T}, \tag{18}$$

where $J$ is the Jacobian of the mapping from image to texture coordinates. The power spectrum of anisotropically filtered noise is obtained by multiplying the power spectrum of unfiltered noise with the frequency domain filter (power spectra in Fig. 6). In the frequency domain, our noise is a sum of Gaussians. The frequency domain filter is a Gaussian as well. Because Gaussians are closed under multiplication, anisotropically filtered noise is again a sum of Gaussians. This means that we can generate anisotropically filtered noise by simply adjusting the parameters of each Gabor kernel based on $J$ and $\sigma$ when evaluating the noise.

We obtain the parameters of the anisotropically filtered Gabor kernel $K'$, $a'$, $F_0'$ and $\omega_0'$ as follows. We express Gaussians using the notation

$$N(f;\mu,\Sigma) = e^{-\frac{1}{2}(f-\mu)\Sigma^{-1}(f-\mu)^T}, \tag{19}$$

where $\mu$ and $\Sigma$ generalize the mean and standard deviation of the Gaussian, f and $\mu$ are two-dimensional vectors, and $\Sigma$ is a $2\times 2$ matrix. Using this notation, the frequency domain filter in texture space is (Eq. (18))

$$N_F(f;0,\Sigma_F), \quad \Sigma_F^{-1} = 4\pi^2\sigma^2\left(JJ^T\right), \tag{20}$$

and the unfiltered Gabor kernel in the frequency domain is (Eq. (7))

$$N_G(f;\mu_G,\Sigma_G), \quad \mu_G = (F_0\cos\omega_0, F_0\sin\omega_0), \quad \Sigma_G^{-1} = \frac{2\pi}{a^2}I. \tag{21}$$

The anisotropically filtered Gabor kernel in the frequency domain is the product of the frequency domain filter in texture space $N_F$ and the unfiltered Gabor kernel in the frequency domain $N_G$

$$\begin{aligned} &N_F(f;0,\Sigma_F)\,N_G(f;\mu_G,\Sigma_G) = \\ &\qquad N(0;\mu_G,\Sigma_F+\Sigma_G)\,N_{FG}(f;\mu_{FG},\Sigma_{FG}), \\ &\Sigma_{FG} = \left(\Sigma_F^{-1}+\Sigma_G^{-1}\right)^{-1}, \quad \mu_{FG} = \Sigma_{FG}\left(\Sigma_G^{-1}\mu_G\right). \end{aligned} \tag{22}$$

This is again a Gaussian in frequency space, or a Gabor kernel. The parameters $(F_0', \omega_0')$ of the anisotropically filtered Gabor kernel are the polar coordinates of $\mu_{FG}$, and $a'$ and $K'$ are

$$a'^2 = 2\pi\sqrt{|\Sigma_{FG}|}, \quad K' = K\frac{a'^2}{a^2}N(0;\mu_G,\Sigma_F+\Sigma_G). \tag{23}$$

The Gaussian of Eq. (22) is in general anisotropic. We approximate this anisotropic Gaussian in frequency space with a Gabor kernel, which is an isotropic Gaussian. Although this approximation could be avoided by using Gabor kernels with an anisotropic Gaussian envelope, it works very well in practice. This is illustrated in Fig.6.

This method is not limited to a specific rendering paradigm. The partial derivatives of the Jacobian can be obtained using ray differentials [Igehy 1999] in renderers based on ray tracing, and using finite differencing in renderers based on rasterization.

Filtering procedural textures obtained by nonlinear transformations of noise is an unsolved problem. However, our noise is more expressive than previous noises, and in many cases removes the need for nonlinear transformations. In Sec. 6 we show that most procedural textures can be obtained using only our noise and a color lookup table. We anisotropically filter these procedural textures by filtering the noise, as described above, and the color lookup table, with a method similar to Hart et al. [1999].

## 6 Results

We show several noise patterns and procedural textures generated with our application for interactive noise design in Fig. 1, Fig. 7 and the accompanying video. Note that all surfaces are triangle meshes, and that surface noise is anisotropically filtered unless noted otherwise. We demonstrate a wide range of procedural textures generated using only our noise and a color map. This includes typical procedural solid textures (e.g., the marble vase and the wooden statuette in Fig. 7), but also cellular-like textures (e.g., the snake skin in Fig. 7). We show that in addition to anisotropic filtering, anisotropic noise is also useful for modeling anisotropic textures (e.g., the straw hat, the wooden chair and the tree bark in Fig. 7) and even structured textures (e.g., the textile cushion in Fig. 7). This extends the range of textures that can be generated procedurally. We demonstrate that our method for anisotropic filtering results in high-quality animations and does not introduce artifacts such as *swimming* (see video). We show that our surface noise does not require a texture parameterization by texturing an implicit surface that changes shape and topology interactively (see video and Fig. 1). This would not be possible with traditional texture mapping or with surface noise that is not setup free. We demonstrate that solid noise is not a substitute for surface noise by illustrating the difference between solid noise and surface noise (see straw hat


**Figure 7:** *Results. (Top row) Several noise patterns generated with band-limited anisotropic and isotropic noise. (Middle) Several procedural textures generated with our application for interactive noise design. The marble vase and the wooden statuette are generated with solid noise. The granite vase, textile cushion, straw hat, leather boot, rusty car, wooden chair, snake skin and tree bark are generated with filtered surface noise. (Bottom rows) The color maps, noises and widgets used for the procedural textures generated with surface noise.*

example in video). We achieve frame rates of roughly 30 frames per second at a resolution of $800 \times 600$ using an NVIDIA GeForce GTX 280 (see video), making our noise well-suited for interactive applications.

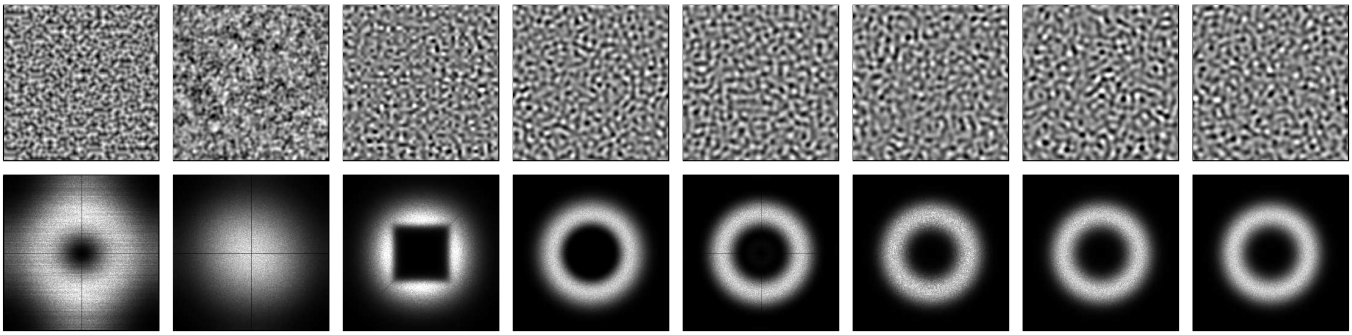
## 7 Comparison and Discussion

In this section we compare our noise with Perlin noise, sparse convolution noise, wavelet noise, the noise of Goldberg et al., and the noise of Kensler et al. The noises and their power spectra are shown in Fig. 8, and a feature matrix of the noises is shown in Tab. 1.

**Perlin Noise & Kensler's Noise** Perlin noise [Perlin 2002] and the noise of Kensler et al. [2008] do not support anisotropic noise or anisotropic filtering. Kensler et al. also use a projection method for surface noise. However, the projection does not preserve the Perlin noise lattice, and it is not clear how this affects the properties of the surface noise.

**Sparse Convolution Noise & Spot Noise** Our work is inspired by sparse convolution noise [Lewis 1984; Lewis 1989] but is also related to spot noise [van Wijk 1991], a method for synthesizing stochastic textures for visualizing scalar and vector fields. Both methods offer some form of spectral control. However, Lewis only used a radially symmetric smooth cosine kernel, and spot noise does not have the properties of a procedural noise function.

**Wavelet Noise** Wavelet noise [Cook and DeRose 2005] does not support anisotropic noise or anisotropic filtering. Setup-free surface noise is obtained by performing a weighted line integral of solid noise. However, the storage requirements of solid wavelet noise are significant, a tile with a resolution of $256^3$ requires 64 MB of storage. The noise is only approximately isotropic. The square feature in the power spectrum (Fig. 8(c)), caused by the separable B-splines, indicates an undesirable axis-aligned directional preference in the noise.



(a) Perlin noise. (b) Lewis' noise. (c) Wavelet noise. (d) Goldberg's. (e) Kensler's. (f) Ours, CPU. (g) Ours, GPU. (h) Ours, valid.

**Figure 8:** *Isotropic noises and their power spectra. (a) Perlin noise. (b) Sparse convolution noise. (c) Wavelet noise. (d) The noise of Goldberg et al. (e) The noise of Kensler et al. (f) Our noise, CPU implementation. (g) Our noise, GPU implementation. (h) Our noise, validation.*

| | Perlin noise | Lewis' noise | Wavelet noise | Goldberg's noise | Kensler's noise | our noise |
|---|---|---|---|---|---|---|
| **procedural** | | | | | | |
| storage requirements[1] | $O(N)$ | $O(N)$ | $O(N^d)$ | $O(N^d)$ | $O(dN)$ | $O(1)$ |
| absence of discretely sampled data | ✓ | ✓ | ✓ | | ✓ | ✓ |
| non-periodicity | | | | | | ✓ |
| parameters[2] | | | | disc. sc. & or. | | cont. $K$ $a$ $F_0$ $\omega_0$ |
| **controllable** | | | | | | |
| bandlimited | | | ✓ | ✓ | ✓ | ✓ |
| spectral control | limited | limited | limited | limited | limited | accurate |
| design | weights | kernel | weights | weights | weights | widgets |
| **surface noise** | | | | | | |
| two dimensional noise | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| surface noise | | | ✓ | ✓ | ✓ | ✓ |
| solid noise | ✓ | ✓ | ✓ | | ✓ | ✓ |
| setup free surface noise | | | ✓ | | ✓ | ✓ |
| **anisotropic noise** | | | | | | |
| isotropic noise | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| anisotropic noise | | | | ✓ | | ✓ |
| isotropic filtering | | | ✓ | ✓ | | ✓ |
| anisotropic filtering | | | | ✓ | | ✓ |
| **fast to evaluate** | | | | | | |
| interactive | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| trade quality for speed | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

[1] Storage requirements are expressed in function of the period $N$ and the number of dimensions $d$.

[2] disc. sc. & or. : discrete scale and orientation — cont. $K$ $a$ $F_0$ $\omega_0$ : continuous $K$, $a$, $F_0$ and $\omega_0$.

**Table 1:** *Noise feature matrix. Feature matrix showing Perlin noise, sparse convolution noise, wavelet noise, the noise of Goldberg et al., the noise of Kensler et al. and our noise versus the properties identified in Sec. 1.1.*

**Goldberg's Anisotropic Noise** The noise of Goldberg et al. [2008] is impractical for solid noise, because it has storage requirements similar to wavelet noise. Surface noise is obtained by mapping two-dimensional noise onto the surface and compensating for distortions. Therefore, the noise requires a surface parameterization and a method for handling seams. This removes one of the major advantages of procedural texturing, that is that any surface can be textured without preprocessing. The noise stores discretely sampled data and is therefore potentially subject to discretization artifacts. Similar to wavelet noise, the noise is potentially subject to repeating patterns if only a single tile is used.

**Performance** Tab. 2 shows the performance of our 2D noise and unfiltered and filtered surface noise, in frames per second (FPS) and megapixels per second (MPixels/s), using an NVIDIA GeForce GTX 280. In comparison, the noise of Goldberg et al. [2008], one of the fastest noises, runs at 538 FPS and 258 MPixels/s, (4 octaves, $800 \times 600$). This means that, assuming 30 impulses per cell, our 2D noise is about 4 times slower, while our filtered surface noise is 40 times slower, but offers accurate spectral control, is setup free, and is truly procedural. Although our noise is not as fast as most other noises, it runs at interactive rates and enables a speed vs. quality trade-off.

| # impulses/cell | 20 | 30 | 40 | 50 |
|---|---|---|---|---|
| FPS | 290/111/50 | 201/76/34 | 156/61/26 | 126/53/22 |
| MPixels/s | 76.2/19.6/8.9 | 52.8/13.5/6.1 | 41.0/10.8/4.6 | 33.2/9.4/3.9 |

performance figures: 2D noise/unfiltered surface noise/filtered surface noise
2D noise: $512 \times 512$, surface noise: 175.516 visible pixels

**Table 2:** *Noise performance. Performance of our 2D noise and unfiltered and filtered surface noise, in frames per second and megapixels per second, using an NVIDIA GeForce GTX 280.*

**Interference Patterns** Very narrowly band-limited noise may exhibit interference patterns (see e.g., Fig. 7, top row, fourth pattern). This is because very narrowly band-limited noise consists only of harmonics with frequencies very close to the principal frequency. These harmonics slowly transition between perfectly in phase, causing constructive interference, and perfectly out of phase, causing destructive interference. We have verified that the interference patterns are not artifacts introduced by the procedural evaluation using our non-procedural CPU implementation based on frequency filtering.

**Spectral Control** Our noise offers more accurate spectral control than a weighted sum of octaves. A weighted sum can only roughly approximate a desired power spectrum, because of the coarse discretization of frequency space, and therefore offers only

limited spectral control. The resulting approximation errors can be visually noticeable and affect noise design and noise filtering. Although the approximation can be improved by using a finer discretization, this increases storage requirements and evaluation cost. Our noise samples a parameteric description of the desired power spectrum instead of discretizing frequency space, and therefore offers more accurate spectral control. This is demonstrated in Sec. 3.1, 3.2 and 5.2.

**Procedural** Our noise has a memory footprint of only a few bytes, does not rely on discretely sampled data, is nonperiodic, and has intuitive parameters, and is therefore truly procedural. Our noise is also mathematically elegant and its properties are characterized analytically.

## 8 Conclusion

We consider accurate spectral control and setup-free surface noise to be the two most important advantages of our new procedural noise. Setup-free surface noise allows the user to map any texture designed with our noise onto a surface, while maintaining the appearance of the texture, without precomputation and undesired artifacts such as distortions and seams. Accurate spectral control allows the user to benefit from increased noise expressiveness, more advanced modeling capabilities and improved rendering quality.

A promising direction for future research is modeling procedural textures by example by guiding our noise with parametric descriptions of power spectra derived from real world textures. Other interesting topics for future work are decoupling the bandwidth of the noise and the size of the grid to enable noise with unrestricted spatially varying bandwidth, extending our method for anisotropic filtering to solid noise, and investigating the possibilities of other kernels and Gabor kernels with more parameters.

## Acknowledgments

We would like to thank the anonymous reviewers, and Bart Adams, Fernanda Andrade-Cabral, Fredo Durand, Eugene Fiume, Brian Smits, Joris Van Deun, Peter Van Gorp and Celine Vens. Ares Lagae is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO), and acknowledges K.U.Leuven CREA funding (CREA/08/017).

## References

ABRAMOWITZ, M., AND STEGUN, I. A. 1964. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, ninth Dover printing ed. Dover.

BOROSH, I., AND NIEDERREITER, H. 1983. Optimal multipliers for pseudo-random number generation by the linear congruential method. *BIT Numerical Mathematics 23*, 1, 65–74.

BRACEWELL, R. N. 1999. *The Fourier Transform and its Applications*, 3rd ed. McGraw-Hill.

BRIDSON, R., HOURIHAM, J., AND NORDENSTAM, M. 2007. Curl-noise for procedural fluid flow. *ACM Trans. Graphics 26*, 3, 46:1–46:3.

COOK, R. L., AND DEROSE, T. 2005. Wavelet noise. *ACM Trans. Graphics 24*, 3, 803–811.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers, Inc.

FISHER, M., SCHRÖDER, P., DESBRUN, M., AND HOPPE, H. 2007. Design of tangent vector fields. *ACM Trans. Graphics 26*, 3, 56:1–56:9.

FRISVAD, J. R., AND WYVILL, G. 2007. Fast high-quality noise. In *Proc. 5th international conference on Computer graphics and interactive techniques*, 243–248.

GABOR, D. 1946. Theory of communication. *J. Int. Electrical Engineers 93*, 429–457.

GOLDBERG, A., ZWICKER, M., AND DURAND, F. 2008. Anisotropic noise. *ACM Trans. Graphics 27*, 3, 54:1–54:8.

HART, J. C., CARR, N., AND KAMEYA, M. 1999. Antialiased parameterized solid texturing simplified for consumer-level hardware implementation. In *Proc. ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 45–53.

HART, J. C. 2001. Perlin noise pixel shaders. In *Proc. ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 87–94.

HECKBERT, P. S. 1989. *Fundamentals of Texture Mapping and Image Warping*. Master's thesis.

IGEHY, H. 1999. Tracing ray differentials. In *Proc. ACM SIGGRAPH 1999*, 179–186.

KENSLER, A., KNOLL, A., AND SHIRLEY, P. 2008. Better gradient noise. Tech. Rep. UUSCI-2008-001, SCI Institute, University of Utah.

KNUTH, D. E. 1997. *The Art of Computer Programming*, 3rd ed., vol. 2. Addison-Wesley.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Texture sprites: Texture elements splatted on surfaces. In *Proc. 2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 163–170.

LEWIS, J.-P. 1984. Texture synthesis for digital painting. In *Computer Graphics (Proc. ACM SIGGRAPH 84)*, vol. 18, 245–252.

LEWIS, J. P. 1989. Algorithms for solid noise synthesis. In *Computer Graphics (Proc. ACM SIGGRAPH 89)*, vol. 23, 263–270.

MORTON, G. M. 1966. A computer oriented geodetic data base and a new technique in file sequencing. Tech. rep., IBM Ltd.

NAVARRO, R., AND PORTILLA, J. 1996. Robust method for texture synthesis-by-analysis based on a multiscale Gabor scheme. In *Proc. SPIE*, 86–97.

OLANO, M., HART, J. C., HEIDRICH, W., MARK, B., AND PERLIN, K., 2002. Real-time shading languages. SIGGRAPH 2002 Course 36.

OLANO, M. 2005. Modified noise for evaluation on graphics hardware. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 105–110.

PAPOULIS, A., AND PILLAI, U. 2002. *Probability, Random Variables and Stochastic Processes*, 4rd ed. McGraw-Hill.

PAPOULIS, A. 1971. High density shot noise and Gaussianity. *J. Applied Probability 8*, 1, 118–127.

PEACHY, D. R. 1985. Solid texturing of complex surfaces. In *Computer Graphics (Proc. ACM SIGGRAPH 85)*, vol. 19, 279–286.

PERLIN, K., AND HOFFERT, E. M. 1989. Hypertexture. In *Computer Graphics (Proc. ACM SIGGRAPH 89)*, vol. 23, 253–262.

PERLIN, K. 1985. An image synthesizer. In *Computer Graphics (Proc. ACM SIGGRAPH 85)*, vol. 19, 287–296.

PERLIN, K. 2002. Improving noise. In *Proc. ACM SIGGRAPH 2002*, 681–682.

TZENG, S., AND WEI, L.-Y. 2008. Parallel white noise generation on a GPU via cryptographic hash. In *Proc. 2008 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 79–87.

VAN ETTEN, W. C. 2005. *Introduction to Random Signals and*

*Noise*. Wiley.

VAN WIJK, J. J. 1991. Spot noise texture synthesis for data visualization. In *Computer Graphics (Proc. ACM SIGGRAPH 91)*, vol. 25, 309–318.

WORLEY, S. 1996. A cellular texture basis function. In *Proc. ACM SIGGRAPH 1996*, 291–294.

ZHANG, E., HAYS, J., AND TURK, G. 2007. Interactive tensor field design and visualization on surfaces. *IEEE Trans. Visualization and Computer Graphics 13*, 1, 94–107.

## Pseudocode

```
 1  class pseudo_random_number_generator
 2  {
 3  public:
 4    void seed(unsigned s) { x = s; }
 5    unsigned next() { x *= 3039177861; return x; }
 6    float uniform_0_1() { return float(next()) / float(UINT_MAX); }
 7    float uniform(float min, float max)
 8      { return min + (uniform_0_1() * (max - min)); }
 9    unsigned poisson(float mean)
10    {
11      float g = exp(-mean);
12      unsigned em = 0;
13      double t = uniform_0_1();
14      while (t > g) {
15        ++em;
16        t *= uniform_0_1();
17      }
18      return em;
19    }
20  private:
21    unsigned x;
22  };
23
24  float cell(int i, int j, float x, float y)
25  {
26    unsigned s = morton(i, j) + random_offset;
27    if (s == 0) s = 1;
28    pseudo_random_number_generator prng;
29    prng.seed(s);
30    unsigned number_of_impulses = prng.poisson(number_of_impulses_per_cell);
31    float sum = 0.0;
32    for (unsigned i = 0; i < number_of_impulses; ++i) {
33      float x_i = prng.uniform_0_1(), y_i = prng.uniform_0_1();
34      float w_i = prng.uniform(-1.0, +1.0);
35      float omega_0_i = prng.uniform(0.0, 2.0 * M_PI);
36      sum += w_i * gabor(K, a, F_0, omega_0_i,
37          (x - x_i) * kernel_radius, (y - y_i) * kernel_radius);
38    }
39    return sum;
40  }
41
42  float noise(float x, float y)
43  {
44    x /= kernel_radius, y /= kernel_radius;
45    float sum = 0.0;
46    for (int i = -1; i <= +1; ++i)
47      for (int j = -1; j <= +1; ++j)
48        sum += cell(int(x) + i, int(y) + j, frac(x) - i, frac(y) - j);
49    return sum;
50  }
```

**Figure 9:** *Pseudocode for isotropic noise.*