COMPUTER GRAPHICS forum

DOI: 10.1111/j.1467-8659.2010.01827.x

COMPUTER GRAPHICS forum
Volume 29 (2010), number 8 pp. 2579–2600

A Survey of Procedural Noise Functions

A. Lagae[1,2], S. Lefebvre[2,3], R. Cook[4], T. DeRose[4], G. Drettakis[2], D.S. Ebert[5], J.P. Lewis[6], K. Perlin[7] and M. Zwicker[8]

[1]Katholieke Universiteit Leuven, Belgium
[2]REVES/INRIA Sophia-Antipolis, France
[3]ALICE/INRIA Nancy Grand-Est/Loria, France
[4]Pixar Animation Studios, USA
[5]Purdue University, USA
[6]Weta Digital, New Zealand
[7]New York University, USA
[8]University of Bern, Switzerland

Abstract
Procedural noise functions are widely used in computer graphics, from off-line rendering in movie production to interactive video games. The ability to add complex and intricate details at low memory and authoring cost is one of its main attractions. This survey is motivated by the inherent importance of noise in graphics, the widespread use of noise in industry and the fact that many recent research developments justify the need for an up-to-date survey. Our goal is to provide both a valuable entry point into the field of procedural noise functions, as well as a comprehensive view of the field to the informed reader. In this report, we cover procedural noise functions in all their aspects. We outline recent advances in research on this topic, discussing and comparing recent and well-established methods. We first formally define procedural noise functions based on stochastic processes and then classify and review existing procedural noise functions. We discuss how procedural noise functions are used for modelling and how they are applied to surfaces. We then introduce analysis tools and apply them to evaluate and compare the major approaches to noise generation. We finally identify several directions for future work.

Keywords: procedural noise function, noise, stochastic process, procedural, Perlin noise, wavelet noise, anisotropic noise, sparse convolution noise, Gabor noise, spot noise, surface noise, solid noise, anti-aliasing, filtering, stochastic modelling, procedural texture, procedural modelling, solid texture, texture synthesis, spectral analysis, power spectrum estimation

ACM CCS: I.3.3 [Computer Graphics]: Picture/Image Generation—I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism-Colour, shading, shadowing, and texture

1. Introduction

Efficiently adding rich visual detail to synthetic images has always been one of the major challenges in computer graphics. Procedural noise is one of the most successful fundamental tools used to generate such detail. Ever since the first image of the marble vase, presented by Perlin [Per85] (Figure 1), *Perlin noise* has seen widespread use both in research and in industry. Noise has been used for a diverse and extensive range of purposes in procedural texturing, including clouds, waves, tornadoes, rocket trails, heat ripples, incidental motion of animated characters and so on. It is widely used both in film production and video games, and is currently implemented in every major three-dimensional (3D) computer graphics software package, such as Autodesk 3ds Max and Maya, Blender, Pixar's RenderMan®, etc.

Procedural noise has many advantages: it is typically very fast to evaluate, often allowing evaluation of complex and intricate patterns on-the-fly, and it has a very low memory footprint, making it an ideal candidate for compactly generating complex visual detail. In addition, with a suitable set of

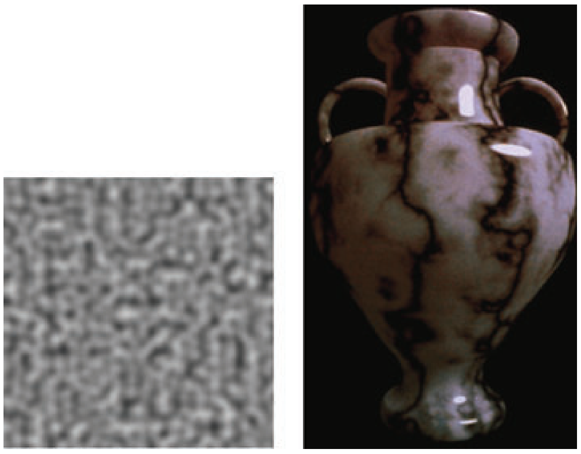

Figure 1: *Perlin noise. (a) Perlin's famous noise function, the first procedural noise function. (Figure from [Per02], ©ACM, 2002.) (b) Perlin's famous marble vase, one of the first procedural textures created using Perlin noise. (Figure from [Per85], ©ACM, 1985.)*

parameters, procedural noise can be used to easily generate a large number of different patterns. Finally, procedural noise is often randomly accessible, so that it can be evaluated independently at every point in constant time. This last property has always been a great advantage, but takes on even higher significance with the advent of massively parallel GPU's and multicore CPU systems.

The most recent survey on noise is in the book of Ebert *et al.* [EMP*02]. Since then there have been a multitude of recent research results in the domain, such as [CD05, BHN07, GZD08, LLDD09a], as well as many others. In this survey we provide a unified view of both previous techniques (e.g. [Per85, Pea85, Lew89, PH89, vW91, Wor96, Per02]), and this more recent work. We also believe that recent trends in hardware justify the need to take a fresh look at procedural noise. Since 1985, compute speed has increased much faster than memory bandwidth. In a sense, we can now consider that *cycles are free*, in reference to the fact that most programs in today's architectures spend a large amount of their time waiting for cache misses and other kinds of memory access. A direct consequence is that CPU-intensive algorithms are becoming more and more attractive; this is one of the main reasons that procedural methods are regaining popularity. Periodic critical re-examination of previous and recent methods is thus very important.

In this survey, we attempt to provide such a critical look at procedural noise methods. To provide a well-founded view of the field, we start with both an intuitive definition of noise and a formal definition based on stochastic processes in Section 2. In this section, we also define procedural techniques and the different tradeoffs implied. We then provide a high-level review and classification of existing procedural noise functions in Section 3. The following two sections examine the important issues of modelling details with noise (Section 4), and that of defining noise on a surface and how to perform filtering (Section 5). An important part of any survey is to analyse the strengths and weaknesses of the various available methods (Section 6) and to provide a comparison of the different tradeoffs offered by each approach, which we present in Section 7. We conclude in Section 8, providing directions we find interesting for future work.

2. Definition of Procedural Noise Function

In this section, we define *procedural noise function*, by defining *noise* both intuitively (Section 2.1) and formally (Section 2.2), and by defining the adjective *procedural* (Section 2.3).

2.1. Intuitive definition of noise

Noise is *the random number generator of computer graphics*. It is a random and unstructured pattern, and is useful wherever there is a need for a source of extensive detail that is nevertheless lacking in evident structure. Random patterns are often described in the frequency domain. Whereas in the spatial domain, a signal is determined by specifying the value for every location in space, in the frequency domain, a signal is determined by specifying the amplitude and phase for every frequency. However, for unstructured patterns, the phase is random and does not contribute useful information. Therefore, noise is often described by its power spectrum, which specifies the magnitude (squared) of each frequency and ignores the phase. This bears some similarity to how a chord in music is described by a set of simultaneously sounding notes, each with a specific frequency. A high value of a specific frequency in the power spectrum corresponds to a high contribution of the corresponding *feature size* in the spatial domain. Noise is completely characterized by its power spectrum, as explained in Section 2.2. Many tasks involving noise can be described as manipulations of the power spectrum of the noise, or spectral control. For example, modelling a noise corresponds to shaping its power spectrum, and filtering a noise corresponds to damping frequencies in the power spectrum that are too high.

Perlin and Hoffert [PH89] gave the following definition: *noise is an approximation to white noise band-limited to a single octave*. White noise contains all frequencies in equal mixture and with random phase, so it provides the raw material to generate unstructured signals with any combination of frequencies. A band-limited power spectrum is non-zero only within a specific range of frequencies. So it can be used as a basis in the frequency domain, that is a *spectral basis*, to shape a specific desired power spectrum for modelling or filtering.

2.2. Formal definition of noise

A more formal definition of noise will be useful in comparing and analysing different noise constructions. We will first recall several definitions from random processes (see, e.g. Papoulis and Pillai [PP02]) and Fourier analysis (see, e.g. Bracewell [Bra99]).

For a discrete-valued random process $y = N(x)$, the *nth-order probability density function (pdf)*

$$f_N(y_1, y_2, \ldots, y_n; x_1, x_2, \ldots, x_n) = P(N(x_1) = y_1, N(x_2) = y_2, \ldots, N(x_n) = y_n) \quad (1)$$

is the simultaneous probability that the noise takes on particular values y_k at n specified locations x_k. The first-order pdf is commonly referred to as the amplitude distribution or the signal histogram.

The *nth-order moments* are weighted averages of the corresponding nth-order pdfs. The first-order moment is the mean

$$E[N(x)] = \int y \, f_N(y) \, \mathrm{d}y. \quad (2)$$

The second-order moment is the expected product of the noise at two locations and is termed the autocorrelation or autocovariance (some authors define the autocovariance as the autocorrelation of the signal with the mean removed):

$$E[N(x_1)N(x_2)] = \iint y_1 y_2 \, f_N(y_1, y_2; x_1, x_2) \, \mathrm{d}y_1 \mathrm{d}y_2. \quad (3)$$

A *stationary* random function is one whose statistics are invariant to a shift in the origin of the coordinate system, and a random function is *isotropic* if its statistics are also invariant to rotation of the coordinate system. For a stationary and isotropic random function the autocorrelation reduces to a function of a single variable,

$$E[N(x_1)N(x_2)] = R(|x_1 - x_2|). \quad (4)$$

The autocorrelation evaluated at zero is simply the standard definition of variance, $R(0) = E[(N(\,\cdot\,) - E[N(\,\cdot\,)])^2]$. Importantly, the power spectrum of the noise is the Fourier transform of the autocorrelation function of the noise.

Most existing noise functions model or approximately model only the first- and second-order moments, rather than attempting to model the full nth-order pdf. In this respect, noise functions are distinguished from texture synthesis algorithms (see, e.g. Wei *et al.* [WLKT09]) that closely reproduce example textures and thus necessarily reproduce their statistics. It can be seen that even the second-order pdf requires a lot of information to specify and manipulate. For example specifying an arbitrary second-order pdf for a 2D noise over a 4×4 neighbourhood, $P(N(1, 1) = y_{11}, N(1, 2) = y_{12}, \ldots, N(2, 1) = y_{21}, \ldots, N(4, 4) = y_{44})$ involves 256^{16} numbers if the values are quantized to 8 bits. However, most noise functions have pdfs that are jointly normal (Gaussian). In this case the nth-order pdf is fully and uniquely determined by only the first- and second-order moments, so control of the noise requires specifying only the desired mean and autocorrelation function or power spectrum.

Most noises have an approximately Gaussian intensity distribution, but for different reasons. For example, for noises based on frequency filtering this is because convolution implies Gaussianity [Bra99, 17], and for sparse convolution noises, this is because high-density shot noise implies Gaussianity [Pap71]. More generally, noise algorithms typically involve a weighted sum of independent pseudo-random values. Because the pdf of a sum of random variables is the convolution of the pdf of the individual random variables [Bra99], the resulting pdf rapidly approaches Gaussian form.

Using the preceding definitions, we take the following as a definition of noise:

A noise is a stationary and normal random process. Control of the power spectrum is provided, either directly, or through the summation of a number of independent scaled instances of (typically band-limited) noise.

Note that existing noise functions were not designed with this definition in mind; rather, this definition summarizes the properties of most existing noise functions.

In summary, noise is specified through its autocorrelation function, or equivalently the power spectrum. Controlling a noise using these statistical functions is appropriate, because their shape is unique and specifies the character of the noise, whereas values of the noise itself vary randomly. The choice of the second-order moments is perhaps a *sweet spot*. These statistics provide a significant amount of control and have a well developed mathematical theory. Although modelling a highly structured texture by successively reproducing further higher-order statistics is possible, the exercise may resemble constructing a square wave by Fourier summation—alternate approaches to the goal should be considered. It is also known that humans have difficulty distinguishing images that differ only in their higher order statistics [Jul62]. Although both the autocorrelation and power spectrum represent the second-order moments, the power spectrum is the generally chosen representation, perhaps because it is both familiar and easily interpreted.

In this survey, noise algorithms will generally be described for the 2D case. Generalizations to 1D, 3D and 4D are straightforward. The problem of defining noise at the surface of a 3D object (solid noise and surface noise) requires more care as discussed in Section 5.

2.3. Definition of *procedural* noise

The adjective *procedural* is used in computer science to distinguish entities that are described by program code rather than by data structures. Procedural techniques are code segments or algorithms that specify some characteristic of a computer-generated model or effect. For example, the procedural marble texture in Figure 1 uses algorithms and mathematical functions instead of a digital photograph to define the colour values. We thus define a *procedural noise function* as a procedural technique for simulating and evaluating noise.

The advantages of a *procedural* noise function are the following:

- A procedural noise function is extremely *compact*, normally requiring a few kilobytes of space compared to megabytes for noise images and volumes.
- A procedural noise function is inherently *continuous*, *multi-resolution*, and *not based on discretely sampled data*. A procedural noise function can produce noise at any resolution desired, from an overview to extremely close inspection at high resolution.
- A procedural noise function is *non-periodic*, filling the entirety of 2-D, 3-D to *n*-D space. In other words, it is unlimited in extent and can cover an arbitrary large area without seams and unwanted repetition.
- A procedural noise function is *parametrized*, so it can generate a class of related noise patterns rather than being limited to one fixed noise pattern. The parameters control the power spectrum of the noise, which characterizes the noise pattern.
- A procedural noise function is *randomly accessible*. It can be evaluated in a constant time, regardless of the location of the point of evaluation, and regardless of previous evaluations. This random accessibility and independent point evaluation make noise functions well suited to harness the power of multi-pipe GPUs and multicore CPUs.

These advantages are only *potential* advantages. They are not necessarily guaranteed, but should be considered as aspirations that result in the most useful procedural noise functions.

For more information on procedural techniques, see Ebert *et al.* [EMP*02], on which this discussion is based.

3. Overview of Procedural Noise Functions

In this section, we give a detailed overview of procedural noise functions. We classify noise functions into three categories: lattice gradient noises (Section 3.1), explicit noises (Section 3.2) and sparse convolution noises (Section 3.3). For each of these categories, we discuss a few representative noise functions in detail, and give an overview of related noise functions. We also discuss several related methods that do not qualify as noise functions (Section 3.4).

3.1. Lattice gradient noises

Lattice gradient noises generate noise by interpolating or convolving random values and/or gradients defined at the points of the integer lattice. The representative example of lattice gradient noises is Perlin noise.

3.1.1. *Perlin noise*

In 1985, Perlin introduced *Perlin noise*, his famous procedural noise function [Per85, Per02].

Perlin noise determines noise at a point in space by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice and then doing a splined interpolation. The pseudo-random gradient is given by hashing the lattice point and using the result to choose a gradient. Lattice points are hashed by successive application of a pseudo-random permutation to the coordinates to de-correlate the indices into the array of pseudo-random unit-length gradient vectors. The set of gradients consists of the 12 vectors defined by the directions from the centre of a cube to its edges. The interpolant is a quintic polynomial, which ensures a continuous noise derivative.

Since its introduction more than two decades ago, Perlin noise has found wide use in graphics. Perlin noise is fast and simple, and has continued to be the workhorse of the industry.

3.1.2. *Other lattice gradient noises*

Several variations, improvements, extensions and implementations of lattice gradient noises and Perlin noise have been presented.

Terminology. Ebert *et al.* [EMP*02] presented several instances of lattice gradient noises and a corresponding terminology. *Lattice noises* are defined as a noise functions based on the integer lattice. *Value noises*, *gradient noises* and *value-gradient noises* are defined as noise functions based on values, gradients or both. *Lattice convolution noises* are defined as a noise functions based on convolution. We collectively call these noises *lattice gradient noises*, because the most well-known noise in this category, Perlin noise, is a lattice gradient noise.

Other lattices. Several authors presented noise functions based on other lattices than the integer lattice. Wyvill and Novins [WN99] presented a lattice convolution noise, based on a more densely and evenly packed grid, inspired by sphere packing. Olano *et al.* [OHH*02] presented *simplex noise*, a Perlin-like noise based on a simplex grid. These other lattices lower computational complexity and eliminate undesired directional artefacts.

Physically-based simulations. Several authors presented noise functions for physically based-simulations. Perlin and

Neyret [PN01] presented *flow noise*, a Perlin-like noise function for generating time-varying flow textures with swirling and advection. Bridson *et al.* [BHN07] presented *curl noise*, a Perlin-like noise function for generating time-varying incompressible turbulent velocity fields. For more details about noise in physically-based simulations, see Bridson *et al.* [BHN07].

Better gradient noise. Kensler *et al.* [KKS08] presented *better gradient noise*, three mutually orthogonal improvements to Perlin noise. A modified hash function combined with a separate gradient table improves axial decorrelation. A different reconstruction kernel improves band-limitation. A projection method improves the quality of noise on 2D surfaces using solid noise. Note that these improvements apply to several lattice gradient noises.

Hardware implementations. Several authors presented hardware implementations of Perlin-like noise functions. Hart *et al.* [HCK99] presented a VLSI hardware implementation of Perlin noise. Both Hart [Har01] and Olano [Ola05] presented a GPU implementation of Perlin noise. Since 2003, *noise* is an integral part of the OpenGL Shading Language (GLSL) [Ros06]. Spjut *et al.* [SKB09] presented a CMOS hardware implementation of better gradient noise.

3.2. Explicit noises

Explicit noises generate noise in an explicit manner in a preprocess and store it. Explicit noises are not procedural noise functions in the strict sense, but are very relevant nevertheless, which is why we cover them here. Two representative examples of explicit noises are wavelet noise and *anisotropic noise*.

3.2.1. *Wavelet noise*

In 2005, Cook and DeRose introduced *wavelet noise* [CD05]. Cook and DeRose observed that Perlin noise is prone to problems with aliasing and detail loss, because it is only weakly band-limited, and introduced a new noise function that is almost perfectly band-limited.

In a preprocess, a tile of noise coefficients N is created. These coefficients represent the noise $N(x)$ as a quadratic B-spline surface. This is done by creating an image R filled with random noise, downsampling R to create the half-size image $R^{\downarrow}$, upsampling $R^{\downarrow}$ to a full size image $R^{\downarrow\uparrow}$, and subtracting $R^{\downarrow\uparrow}$ from the original R to create N. This is illustrated in Figure 2. The tile of noise coefficients N is thus created by taking R and removing the part that is representable at half-size. What is left is the part that is *not* representable at half-size, that is the band-limited part. The filters used in the downsampling and upsampling steps are obtained using wavelet analysis and correspond to the analysis and refinement coefficients of the uniform quadratic B-spline basis

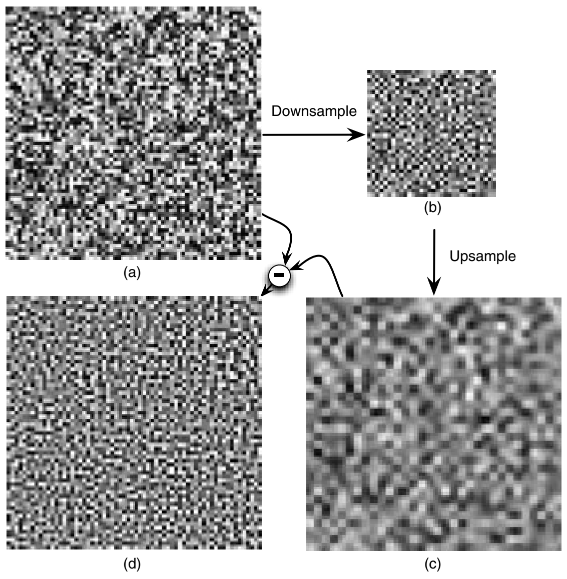

Figure 2: *Wavelet noise generation. (a) Image R of random noise. (b) Half-size image $R^{\downarrow}$. (c) Half-resolution image $R^{\downarrow\uparrow}$. (d) Noise band image $N = R - R^{\downarrow\uparrow}$. (Figure from [CD05],©ACM, 2005.)*

function. The extension to more dimensions is straightforward.

During runtime, once the coefficients n_i have been determined, a value of $N(x)$ for a given x can be computed using any evaluation method for quadratic B-splines. A small precomputed volume of noise coefficients is used and space is tiled with that volume.

Cook and DeRose also identified for the first time that sampling a 3D noise function at a 2D surface will not result in a band-limited texture, even if the 3D function is perfectly band-limited. This is discussed in more detail in Section 5.

3.2.2. *Anisotropic noise*

In 2008, Goldberg *et al.* introduced *anisotropic noise*[1] [GZD08][2]. Goldberg *et al.* observed that existing noise functions only support isotropic filtering, which involves a tradeoff between aliasing artefacts and loss of detail, and presented a new noise function that supports high-quality anisotropic filtering.

[1] When referring to the method, we will emphasize *anisotropic noise*.

[2] See Lagae *et al.* [LZD09] for errata and clarifications.

The main idea of *anisotropic noise* is to generate noise textures by tiling the frequency domain into oriented subbands. Anisotropic noise bands are not only narrowly band-limited in scale, but they also have a preferred orientation. The construction of *anisotropic noise* is based on steerable filters [SF95, PS00] that partition the frequency domain. They provide a number of properties that are crucial for noise generation. First, each filter defines a subband that is tightly localized in scale and orientation. Secondly, the filters implement an invertible transform. This implies that one can exactly recover a signal from its decomposition into subbands. Finally, the filters are steerable in orientation. This essentially means that a linear interpolation of the filters can generate a filter with exactly the same profile, but at an intermediate orientation. This is useful because it avoids interpolation artefacts when linearly blending the subbands for appropriate noise filtering.

In an off-line process, noise tiles are synthesized and stored as discussed earlier. Each oriented subband image is packed into one channel of a 32-bit RGBA image, yielding four orientations per texture. Typically, using four or eight bands, that is one or two textures, leads to a good trade-off between storage, rendering speed, and image quality. Note that noise subbands are pre-computed at a single scale only. All other scales are generated on the fly by simply scaling the pre-computed textures.

During rendering, a pixel shader computes the final noise value simply as a weighted sum of noise subbands at each pixel. By computing appropriate weighted combinations of the oriented subbands at each location, any desired frequency spectrum on the surface can be approximated.

Goldberg *et al.* used *anisotropic noise* to obtain surface noise by 2D texture mapping and compensating for parametric distortion, and for anisotropic analytic filtering. This is discussed in more detail in Section 5.

3.2.3. *Other explicit noises*

Two important categories of explicit noise are stochastic subdivision and Fourier spectral synthesis.

Stochastic subdivision. *Stochastic subdivision* was introduced by Fournier *et al.* [FFC82], who presented the midpoint displacement method, a stochastic subdivision algorithm to generate natural irregular fractal-like objects and phenomena, such as terrain. Lewis [Lew86, Lew87] presented *generalized stochastic subdivision*, a generalization of the work of Fournier *et al.* to arbitrary autocorrelation functions.

Fourier spectral synthesis. *Fourier spectral synthesis* generates a noise with a specific power spectrum by filtering white noise in the frequency domain (see, e.g. Bracewell [Bra99]). Fourier spectral synthesis was introduced in computer graphics by Anjyo [Anj88], Saupe [Sau88] and Voss [Vos88], who used it to generate random fractals to simulate natural phenomena. The mathematical texturing function of Gardner [Gar84] can also be seen as Fourier spectral synthesis. Fourier spectral synthesis is often used in methods for explicit noises, for example by van Wijk [vW91] for *spot noise* (Section 3.3.2), and by Goldberg *et al.* [GZD08] for *anisotropic noise*. Fourier spectral synthesis can also be useful to generate reference solutions for noise functions for which the expected power spectrum is known.

3.3. Sparse convolution noises

Sparse convolution noises generate noise as the sum of randomly positioned and weighted kernels. Three representative examples of are sparse convolution noise, spot noise and Gabor noise.

3.3.1. *Sparse convolution noise*

In a series of papers between 1984 and 1989, Lewis introduced *sparse convolution noise* [Lew84, Lew86, Lew89], a framework for noise functions that offers direct spectral control.

The construction of sparse convolution noise is simple: an arbitrary kernel k is convolved with a Poisson process noise γ,

$$N(x, y) = \iint \gamma(u, v)\, k(x - u, y - v)\, \mathrm{d}u\mathrm{d}v. \tag{5}$$

The Poisson process consists of impulses of uncorrelated intensity a_k situated at random independently chosen locations (x_k, y_k),

$$\gamma(x, y) = \sum_k a_k\, \delta(x - x_k, y - y_k). \tag{6}$$

The Poisson process is *sparse* rather than being defined at every pixel or point in space, hence the name *sparse convolution*. This use of the sparse impulse noise allows some computational efficiency as the convolution is effectively splatting the amplitude-scaled kernel only at the locations (x_k, y_k).

To evaluate the noise at a particular point it is necessary to splat only the kernels that overlap that point. This is accelerated by introducing a virtual grid where the size of a grid cell is equal to the radius of the kernel. The evaluation then considers only the kernels centred in the cell containing the point and those in the neighbouring cells. The coordinates of the cell are also used to seed a random number generator for generating the Poisson impulses located in that cell. More details and improved schemes for this step are given by Worley [Wor96] and Lagae *et al.* [LLDD09a]. Although Lewis [Lew89] describes several optimizations such as caching the constructed Poisson impulses under the assumption of

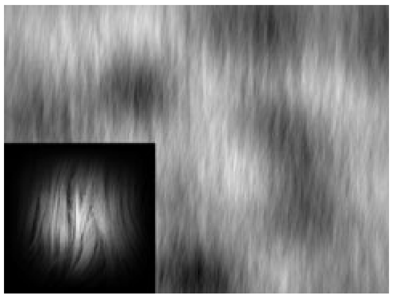

Figure 3: *Sparse convolution noise. Lower left panel: windowed sample from an image of hair. Right panel: approximate hair texture created using 2D sparse convolution using this kernel.*

coherent access, the sparse convolution noise is somewhat slower than a single octave of Perlin noise.

Because the power spectrum of the output of a convolution is the product of the inputs [Bra99], and the power spectrum of the Poisson impulse process is constant, the power spectrum of the sparse convolution noise is simply a scaled version of that of the kernel. Direct control of the desired power spectrum is thus obtained simply by choosing a kernel having that spectrum. For example, a noise sharing the power spectrum of a sample texture can be constructed by using a weighted sample of the texture as a kernel (Figure 3). (Note that the windowing operation slightly blurs the spectrum as discussed in the signal processing and filter design literature). A kernel with an arbitrary power spectrum can be constructed with the following steps: (1) generate a white random noise, (2) transform it to the frequency domain (because the transform of a white noise is also white, step 1 can in fact be skipped), (3) filter the transformed noise with the desired spectral profile, (4) transform to the spatial domain and (5) multiply by a spatial window to produce the kernel (again considering standard window design issues).

This generality in the choice of the kernel is not without problems however. The construction just mentioned typically results in kernels that do not monotonically decay away from the origin. Unless the density of the Poisson impulse process is high, *valleys* in the kernel are become visible as structures in the synthesized noise. Although this may be desirable for some purposes, it is objectionable in other situations and it violates the definition of noise as a *structureless* construct. Gabor noise (Section 3.3.3) avoids this problem while still providing spectral control.

Another way of looking at the issue is in terms of phase. As the density of the Poisson process is increased, the phase structure resulting from features in the kernel is increasingly randomized, whereas the power spectrum of the kernel is preserved. At a range of intermediate density values it is possible to directly synthesize noises with some textural features, as shown in Figure 3. However, the advent of successful texture synthesis methods in the last decade provides a better approach to this problem and clarifies that *noise* algorithms are most appropriate for the random phase case.

Although sparse convolution provided an approach to direct spectral control, it did not make any recommendation on *which* kernel to use. In the light of recent work we see that the implicit suggestion of allowing any kernel is in fact not as useful as choosing the right kernel.

3.3.2. *Spot noise*

In 1991, van Wijk introduced *spot noise* [vW91], a method to generate stochastic textures for the visualization of scalar and vector fields over surfaces. Spot noise can be seen as an explicit form of sparse convolution noise, computed by scan-conversion of the spots or by Fourier spectral synthesis (Section 3.2.3). Although spot noise is both an explicit noise as well as a sparse convolution noise, it is more relevant to sparse convolution noises, which is why we cover it here.

van Wijk discusses the relation between the spot and the texture in detail. van Wijk hinted at several important concepts which were only later introduced in the context of noise. For example, texture mapping on parametric surfaces, texture synthesis over curved surfaces as an alternative to solid noise, and local control by variation of the spot.

3.3.3. *Gabor noise*

In 2009, Lagae *et al.* introduced *Gabor noise* [LLDD09a, LLDD09b, LLD10]. Lagae *et al.* further developed the framework of sparse convolution noise by introducing the Gabor kernel.

The Gabor kernel in the spatial domain, g, is the multiplication of a circular Gaussian and a 2D cosine,

$$g\,(x, y) = K\,\mathrm{e}^{-\pi a^2 (x^2+y^2)} \cos\left[2\pi F_0\,(x \cos\omega_0 + y \sin\omega_0)\right], \tag{7}$$

where K and a are the magnitude and inverse width of the Gaussian, and F_0 and ω_0 the frequency and orientation of the cosine (Figures 4(a)–(d)). The Gabor kernel in the frequency domain, G, is a pair of circular Gaussians,

$$G(f_x, f_y) = \frac{K}{2a^2} \exp\left\{-\frac{\pi}{a^2}\left[(f_x \pm F_0 \cos\omega_0)^2 + (f_y \pm F_0 \sin\omega_0)^2\right]\right\}, \tag{8}$$

where the Gaussians are located at the frequency with polar coordinates (F_0, ω_0), and a is the width of the Gaussians [Figures 4(e)–(f)].

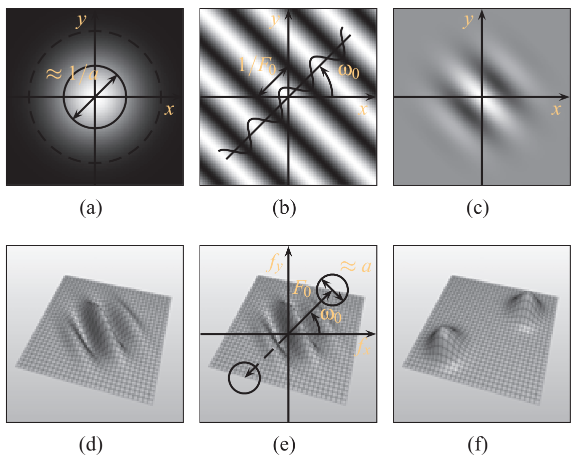

Figure 4: *The Gabor kernel used in Gabor noise. (a) Gaussian. (b) Cosine. (c) Gabor kernel. (d) Gabor kernel, 3D plot. (e) Fourier transform of Gabor kernel. (f) Fourier transform of Gabor kernel, 3D plot. (Figure from [LLDD09a], ©ACM, 2009.)*

Gabor noise is a sparse convolution noise with as kernel the Gabor kernel

$$N(x, y) = \sum_i w_i g(K_i, a_i, F_{0,i}, \omega_{0,i}; x - x_i, y - y_i), \quad (9)$$

where $\{w_i\}$ are the random weights, g is the Gabor kernel, and $\{(x_i, y_i)\}$ are the random positions. Depending on how the parameters $\{K_i\}$, $\{a_i\}$, $\{F_{0,i}\}$ and $\{\omega_{0,i}\}$ vary for different kernels, different kinds of Gabor noise are obtained. When the parameters are fixed, the power spectrum of the noise is that of the Gabor kernel, and an anisotropic band-limited noise is obtained, where ω_0, F_0 and a control the orientation, frequency and bandwidth of the noise. When the parameters are varied, the power spectrum of the noise is that of the Gabor kernel integrated over the parameters. For example, when $\{\omega_{0,i}\}$ is uniformly distributed over $[0, 2\pi)$, an isotropic band-limited noise is obtained, where F_0 and a control the frequency and bandwidth of the noise. Lagae *et al.* use graphical user interface widgets to specify the power spectrum of the noise by specifying how the parameters vary (Figure 5).

Lagae *et al.* used Gabor noise for setup-free surface noise and analytic anisotropic filtering of noise. This is discussed in more detail in Section 5.

3.3.4. *Other sparse convolution noises*

Several extensions and implementations of sparse convolution noises have been presented.

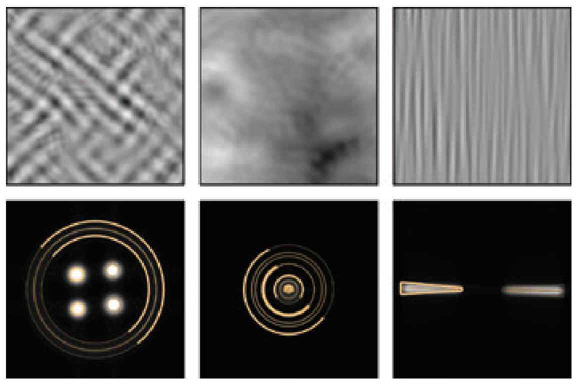

Figure 5: *Gabor noise. Several Gabor noise patterns. The top row shows the Gabor noise patterns, the bottom row shows the corresponding widgets. (Figure from [LLDD09a], ©ACM, 2009.)*

Shaped point processes. Lewis [Lew86] presented *shaped point processes*, one of the works which would eventually lead to sparse convolution noise. This work hinted at several important concepts that would only later be fully developed. For example, a bandpass kernel resembling the Gabor kernel (as in Gabor noise), filtering of noise (Section 5), and spatially varying noise (as in Lagae *et al.* [LLD10]).

GPU implementations. Sparse convolution noise can be implemented on the GPU using splatting (point rendering, scan conversion) or procedurally (using a shader). Frisvad and Wyvill [FW07] presented a GPU implementation based on point rendering of sparse convolution noise with a cubic kernel. Lagae et al [LLDD09a] presented a procedural GPU implementation of Gabor noise.

NPR Gabor Noise. Benard *et al.* [BLV*10] recently presented *NPR Gabor Noise*, a variant of Gabor noise for coherent stylization in non-photorealistic rendering (NPR). To preserve both the 2D aspect of the noise and the 3D motion of the scene, the noise parameters are defined in 2D screen space and the point distribution is defined in 3D object space.

3.4. Related methods

Several methods have been presented that are not procedural noise functions but are nevertheless highly related to procedural noise functions.

3.4.1. *Fractals*

Fractals [Man82] triggered rapid and great progresses of procedural noise functions. For more information on how to use procedural noise functions in practice, see, for example the work of Musgrave [MKM89] and Szeliski and Terzopoulos [ST89] on fractal terrains.

3.4.2. *Texture basis functions*

Texture basis functions are defined as functions to generate patterns that can be used as a basis for generating textures. The most well-known texture basis function is probably the one of Worley. Worley [Wor96] presented a *cellular texture basis function*, a texture basis function based on distances to feature points randomly scattered in space, which is good for creating textures such as flagstone-like tiled areas, organic crusty skin, crumpled paper, ice, rock, mountain ranges and craters. The implementation of Worley's cellular texture basis function is very similar to that of sparse convolution noise. There are several other methods that could also qualify as a texture basis functions, for example the method presented by Tzeng and Wei [TW08] for parallel white noise generation on the GPU.

3.4.3. *Object distribution functions*

We define *object distribution functions* as functions to generate patterns that consist of objects distributed over a background. Lefebvre and Neyret [LN03] presented *pattern-based procedural textures*, a method to generate procedural textures composed of randomly distributed objects on the GPU. Lagae and Dutré [LD05] presented a *procedural object function*, a texture basis function for objects distributed according to a Poisson disk distribution, which is good for creating textures such as polka dots. The tile-based methods used in this method [Lag09] are also useful in the context of procedural noise functions, for example for noise tiles [CD05, YL08].

4. Modelling with Procedural Noise Functions

Creating visually rich and interesting content from noise is not an easy task, essentially because the random nature of noise makes it difficult to control and predict the result. In addition, noise is often only the first component in a long chain of operations to achieve the end result. Most systems for modelling with noise are based on the concept of block shaders [AW90], in which a texture is described as a network of modules.

We mainly focus on the design of the noise patterns themselves, and refer the reader to the book *Texturing & Modeling: A Procedural Approach* [EMP*02] for a more detailed account of the most useful approaches to generate terrains, shapes and textures from noise and procedures.

In this section, we describe spectral control of noise (Section 4.1), direct editing of noise values (Section 4.2), and noise by example (Section 4.3).

4.1. Spectral control of noise

As explained in Section 2.1, noise patterns are best described in terms of frequency content, through their power spectrum. Controlling a noise pattern through its spectrum requires some training, but is convenient once the link between the spectrum and the visual aspect of the noise is understood.

We describe next the most common approaches for spectral noise control. The first approach consists of summing weighted layers of band-limited noise. The second approach discusses the specific case of sparse convolution noises, which are controlled through the choice of kernel.

We would like to note that in previous work the term *band-limited* is often used where the term *band-pass* would be more appropriate. Note that a *band-limited* power spectrum is zero beyond a specific frequency, whereas a *band-pass* power spectrum is zero outside of a frequency interval (see, e.g. Bracewell [Bra99] or Papoulis and Pillai [PP02]). In the preceding sections we use *band-limited* for consistency with previous work, but in the following sections we will use the appropriate term.

4.1.1. *Weighted sum of band-pass noises*

Most procedural noise functions directly produce band-pass noises. Each noise band corresponds to an elementary random pattern, with a frequency content limited to a specific range. Note that bands at different frequencies are easily obtained by scaling an initial band-pass noise.

The very reason for which procedural noise functions are designed to produce band-pass noises is to let complex patterns be defined by adding several bands of noise. Each band is multiplied by a weight controlling its contribution to the final result. This idea was introduced by Perlin [Per85]. The final pattern is obtained as

$$\sum_i w_i \, N(2^i x), \tag{10}$$

where N is a band-pass noise function and w_i is the weight of band i. Successive noise layers have a principal frequency related by a factor of two, which is why they are often called *octaves*. Perlin initially described a noise with $1/f$ spectral content with weights computed as $1/2^i$. However, in a typical noise modelling tool the weights are directly exposed to the user, as shown in Figure 6. The spectrum of the resulting noise pattern is obtained as the weighted sum of the band spectra.

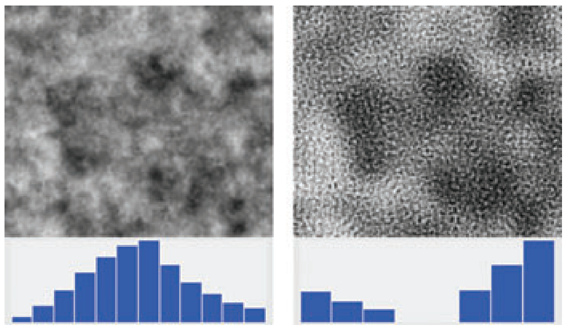

Figure 6: *Spectral control with wavelet noise. Three-dimensional noise patterns with 12 bands with a Gaussian distribution, and 8 bands with a white distribution. The blue bars are the band weights. These weights can be exposed to the user. (Figure from [CD05], ©ACM, 2005.)*

Noise bands that are band-pass have little overlap in the frequency domain and can be seen as a *spectral basis*, defining a space of noise patterns. Note however that in the basis analogy, only positive weights are effective. More specifically, it is not possible to cancel energy from a frequency band because the noise has random phase. Thus, a resulting noise spectrum that contains no energy at some frequencies can only be produced if the primitive noise function is band-pass rather than merely band-limited.

Note that the weights do not have to remain constant in space: By using different weights in different locations one can generate patterns smoothly transitioning between different aspects. This fact is exploited by Goldberg *et al.*[GZD08] to cancel mapping distortions and dynamically adapt the noise to viewing conditions (Sections 5.1 and 5.2).

4.1.2. *Sparse convolution noises*

Sparse convolution noises are controlled through the choice of the kernel, because the noise has the spectrum of the kernel (see Section 3.3.1). This choice may vary spatially to obtain different appearances in different areas [vW91].

Sparse convolution noises can produce band-pass noises with the appropriate kernel. They are thus compatible with the approach of summing noise bands. However, they can also be used to *directly* generate a noise with a specific spectrum, provided that a kernel having this particular spectrum is available. This is the case for sparse convolution noise, which directly produces a noise with the desired spectrum as illustrated in Figure 7(b). Gabor noise [LLDD09a] uses a kernel which can itself be controlled through a number of parameters. These are described through widgets directly manipulated by the user. These parameters give direct control over the spectrum generated by the noise, without having to change the kernel. Noise can evolve from anisotropic band-pass patterns to more elaborate patterns, as illustrated in Figure 5.

4.2. Editing noise values

In addition to spectral control, other techniques investigate how to control the noise values in the spatial domain. This is challenging to achieve without destroying the properties of the noise. Lewis [Lew87] generated a noise in a multiresolution coarse-to-fine scheme, with the fine scale values condition on previously specified values at the coarse scale. However, some of the coarse scale values can be directly specified by the user as shown in [Lew87, Figure 5]. Yoon *et al.*[YLC04, YL08] let the user directly specify a few values of a noise field. New random numbers are generated ensuring that the user constraints are satisfied *and* that the noise keeps its properties (value distribution, non-periodicity, band-pass).

4.3. Noise by example

To avoid manual noise design, several authors have focused on finding parameters from an image. This is, however, an extremely challenging problem. To the best of our knowledge no satisfactory solution exists for the general case of procedural noises as defined in Section 2. It is important to note that neighbourhood-based texture synthesis approaches as surveyed by Wei *et al.* [WLKT09] do not fall in this category. Consequently, this is also an exciting area of further research.

Several interesting solutions exist for subclasses of textures. Ghazanfarpour and Dischler [GD95] express the noise as a sum of sine waves, similarly to Gardner [Gar85]. They select the set of sine waves from an example image, by thresholding the magnitude of its Fourier transform. This 2D function can then be extended to define a solid (3D) noise. The method is further refined in subsequent work [GD96], to support different aspects along different directions of a solid noise. In [DG97], the authors focus on geometric textures. They analyse 1D noise profiles and automatically generate procedures for them. These are then extended to 2D and 3D. The analysis step identifies main frequencies but also performs histogram matching between the example and the generated noise. These spectral approaches work best when the textures contain strong periodicities, with clearly identified features in the power spectrum. Lagae *et al.*[LVLD10] automatically compute weights of a sum of band-pass isotropic noise octaves, to produce an image closely resembling an example. The method produces results close to early by-example texture synthesis approaches [HB95], with the crucial difference that the result is a procedure and can be efficiently point-sampled. Nevertheless, this approach cannot faithfully reproduce structured or anisotropic patterns. In contrast, Galerne *et al.* [GGM09] randomize the phase spectrum of a given texture to obtain a homogeneous and

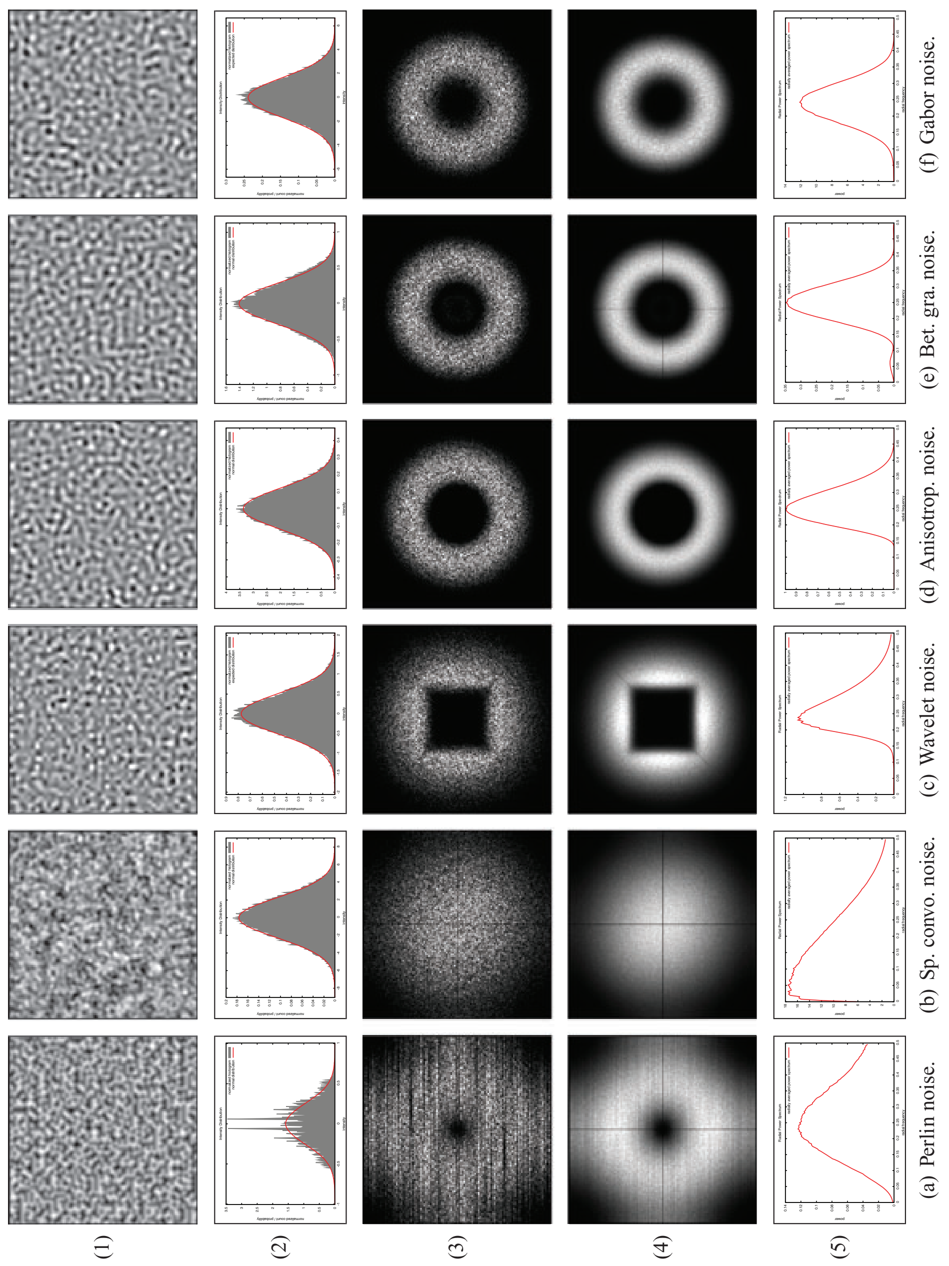

Figure 7: *Analysis of procedural noise functions. (a) Perlin noise. (b) Sparse convolution noise. (c) Wavelet noise. (d) Anisotropic noise. (e) Better gradient noise. (f) Gabor noise. (1) Noise. (2) Amplitude distribution. (3) Periodogram. (4) Power spectrum estimate. (5) Radially averaged power spectrum estimate. (Figure based on [LLDD09a], ©ACM, 2009.)*

featureless noise having the same power spectrum. Recently, Gilet *et al.* [GDS10] did a first attempt at extending the work of Dischler and Ghazanfarpour [DG97] and Lagae *et al.* [LVLD10] to anisotropic textures using Gabor noise. Gilet and Dischler [GD10] also proposed a 3D extension of the work of Bourque and Dudek [BD04] using a mixture of Gabor noise and cellular noise.

Other approaches focus on setting the parameters of existing procedural shaders from an example image: The goal is to make the shader produce an image resembling the example as closely as possible. Although these techniques do not primarily target noise patterns, they could be useful to automatically select parameters of a noise function. Bourque and Dudek [BD04] aim at a more generic approach, searching for closest matches in a database of images generated by sampling the parameter space of many shaders. Qin *et al.*[QY02] similarly optimize shader parameters using a genetic algorithm.

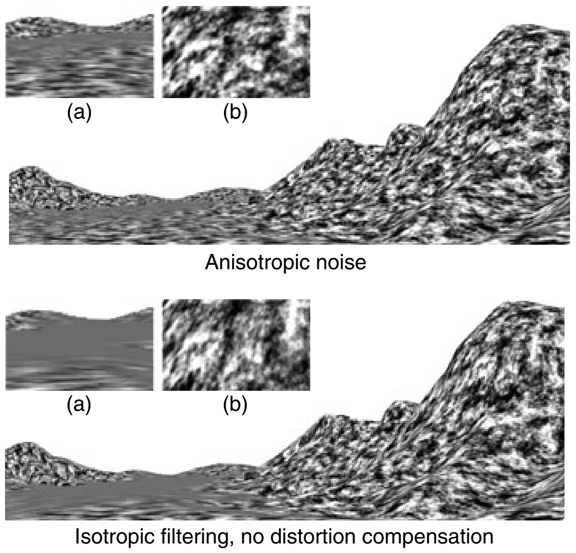

Figure 8: *Anisotropic filtering and compensation for parametric distortions with*anisotropic noise. *(a) Anisotropic noise leads to higher image quality compared to isotropic filtering, as shown by the difference between close-ups. (b) Anisotropic noise compensates for parametric distortions to enforce a uniform noise aspect along the surface, as shown by the difference between close-ups. (Figure from [GZD08],©ACM, 2008.)*

5. Procedural Noise Functions on Surfaces

Noise in computer graphics is especially useful to add visual details in renderings, through texturing. A texture obtained from a noise pattern inherits all its advantages: Non-periodicity, low memory cost, resolution and efficient random access.

In this section, we discuss how noise patterns are typically mapped on surfaces (Section 5.1) as well as the challenges this creates for anti-aliased rendering (Section 5.2).

Using noise to texture surfaces introduces two different, albeit closely related, challenges. A first difficulty is to find an appropriate mapping of the noise to the surface, while preserving the properties of the noise (frequency content, continuity). A second difficulty is to adapt the noise to the viewing conditions. Indeed, a noise with high frequency quickly produces disturbing aliasing artefacts when mapped onto a surface seen at an angle or in the distance.

5.1. Noise on surfaces

There are three methods for obtaining noise on a surface: mapping a 2D noise onto the surface using a planar parametrization, sampling a solid noise, or defining a noise directly on the surface. We refer to this latter case as *surface noise*. Note that a surface noise should retain the properties it exhibits in 2D—i.e. it should remain visually similar to its 2D equivalent even if mapped onto a complex curved surface.

5.1.1. *Mapping a 2D noise*

2D noise can be mapped onto surfaces through planar parametrization, exactly like regular texture maps. However, this can introduce distortions and seams, breaking important properties of the noise such as uniform frequency content, continuity and whether the noise is band-pass.

Goldberg *et al.*[GZD08] compensate for mapping distortions by locally adapting the noise content [Figure 8(b)]. This is a form of dynamic spectral control (Section 4.1), where the weights of the noise bands are driven to compensate for the distortions. The local distortion as well as its impact on the noise spectrum is estimated at every pixel. A noise with inversely pre-distorted frequency content is generated to appear uniform along the surface. This is done by updating, in every pixel, the weights of the summed noise bands to approximate the pre-distorted spectrum. This can be performed efficiently from a shader running on the GPU. This approach, however, only recovers from distortions and cannot hide the seams. Note that this idea was also suggested in the work of van Wijk [vW91, Figures 11 and 12].

5.1.2. *Sampling a solid noise*

Noise can be applied onto surfaces by sampling a 3D noise function at every surface point. All noise functions are easily generalized to 3D and higher dimensions. Explicit noises, however, quickly induce a large memory cost because they rely on pre-computed tables. The idea of sampling a 3D noise on surfaces was introduced by Perlin [Per85] and Peachy [Pea85]. This is often referred to as *solid texturing*. The approach, which popularized procedural textures, has several advantages: It is simple, memory consumption remains low, and the object appears as if carved out of a block of matter, an effect difficult to achieve otherwise. For a complete overview of solid texturing please refer to Dischler and Ghazanfarpour [DG01].

Cook and DeRose [CD05] observed that sampling a solid band-pass noise along a surface does not result in a band-pass noise on the surface. This is a consequence of the slice-projection theorem [Bra99, Mal93], which states that slicing in one domain corresponds to projection or integration in the other domain. Evaluating a 3D noise along a surface corresponds to slicing. Therefore, the power spectrum of the noise on the surface is given by integrating the band-pass power spectrum of the solid noise. However, this power spectrum is not band-pass anymore. Cook and DeRose additionally observed that the slice-projection theorem also provides a solution to this problem. Integrating a solid noise perpendicular to the surface corresponds to projection. Therefore, the power spectrum of the noise on the surface is given by slicing the band-pass power spectrum of the solid noise. This power spectrum is still band-pass. This provides a general method for obtaining a band-pass noise on a surface from a band-pass solid noise.

5.1.3. *Defining noise directly on the surface*

A last alternative is to define a noise directly on a surface, so that its features flow along the curvatures and naturally adapt to topology changes. This is difficult in general, but sparse convolution noises enable this approach: By locally splatting kernels the noise appears along the surface without having to resort to a global planar mapping. These ideas were hinted in earlier work [Cha07, 5.2] and further developed by Lagae *et al.* [LLDD09a]. In this latter work, the noise pattern is procedurally generated along a surface without any preprocessing such as computing a surface parameterization. At any evaluation point, only the 3D point coordinates and the surface normal are necessary to evaluate the 2D noise. For the case of anisotropic (oriented) textures, a direction field must also be provided to indicate the orientation of the texture. Several methods are available for the design of such fields (see, e.g. Fischer *et al.* [FSDH07]).

Solid texturing and surface noise produce different visual effects. The first creates the unique feeling that the object is sculpted out of solid matter, whereas the second lets anisotropic textures 'flow' around the object. This is important when texturing, for instance, objects made out of fibres (straw basket, woven cloth, etc.). Figure 9 illustrates this idea.

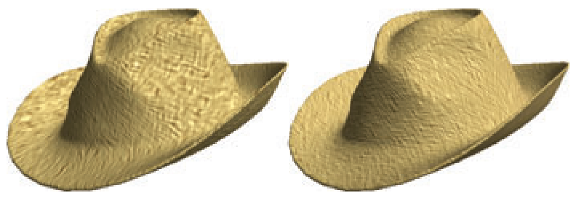

Figure 9: *Difference in aspect of solid noise and surface noise. Straw hat textured with both a solid noise (left) and a surface noise (right). Left: The straw orientation is fixed in space, resulting in stretch on the side of the hat. Right: The straw orientation flows around the surface, producing the appropriate effect. There are no texture coordinates in both cases. (Figure from [LLDD09a], ©ACM, 2009.)*

5.2. Filtering noise on surfaces

An important consideration when mapping noise on surfaces is filtering of the frequency content when objects are seen at an angle or from a distance. This is crucial for rendering quality. Super-sampling is generally only necessary at geometric edges because textures are filtered, for instance using MIP-mapping. A major drawback of procedural textures is that such filtered lookups may not be available, requiring the use of super-sampling on the entire image. Because textures contain very fine details, and are seen from very close to far away, super-sampling will often not be able to solve the problem entirely at reasonable cost. It is thus crucial to provide filtered sampling of procedural textures.

In Appendix, we provide the necessary background to understand filtering of signals mapped to surfaces. Although filtering is typically seen as a convolution in the spatial domain, it can also be interpreted as a multiplication in the frequency domain. More specifically, the spectrum of the filtered noise is given by the multiplication of the spectrum of the unfiltered noise and the spectrum of the filter in texture space. The filter in texture space varies in each screen pixel because it is view-dependent. Figure 10 illustrates these concepts.

We first describe how noise can be filtered (Section 5.2.1), and then discuss filtering of texture patterns obtained by applying transformations to noise values (Section 5.2.2).

5.2.1. *Filtering noise*

The key idea of filtering noise is to exploit the spectral control offered by the noise to directly generate noise with the filtered power spectrum, rather than explicitly filtering unfiltered noise.

When noise patterns are obtained as a weighted sum of band-pass noises, a first approach is to cancel the contribution of bands whose frequency is too high. This approach is often referred to as *frequency clamping* [NRS82]. This works best

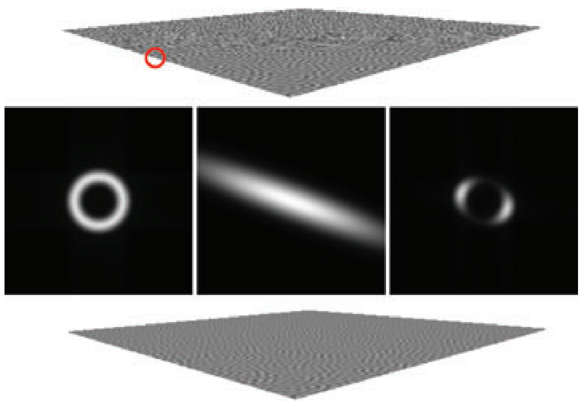

Figure 10: *Anisotropic filtering with Gabor noise. Top: Unfiltered noise mapped on a tilted plane. The noise pattern is incorrect in the distance due to aliasing. Middle, from left to right: The power spectrum of the unfiltered noise, the filter for pixels in the red circle area, the power spectrum of the filtered noise for these pixels. This last spectrum is simply the product, in the frequency domain, of the filter and the noise power spectrum. Bottom: Same noise but properly filtered. Aliasing is entirely removed. (Figure from [LLDD09a], ©ACM, 2009.)*

if the noise is narrowly band-pass (i.e. the ring in the spectrum has to be thin and well defined). Perlin noise [Per85] is only weakly band-pass, making frequency clamping difficult to tune. Cook and DeRose alleviate this issue by providing a noise with better defined band-limits [CD05].

Both of these noises, however, are isotropic and the clamping cannot account for the anisotropy of the filter. On tilted surfaces one must compromise between over-blurring or residual aliasing. Goldberg *et al.* [GZD08] obtain higher quality filtering since their pre-computed noise bands are oriented. Each band corresponds to a noise pattern with limited frequency content along a given orientation (Figure 11). By adapting the weights of the oriented bands with respect to the anisotropic filter, the noise content adapts to non-uniform perspective distortions [Figure 8(a)].

Lagae *et al.* [LLDD09a] exploit a unique property of their noise. The noise is obtained as a sum of Gabor kernels. Each Gabor kernel corresponds to a Gaussian in the frequency domain. It is possible to filter each individual kernel by computing the product, in the frequency domain, between the Gabor Gaussian and the filter Gaussian. Because Gaussians are closed under multiplication, the product is a third Gaussian. This new Gaussian can be interpreted as a filtered Gabor kernel. The parameters of this new kernel are used instead of those of the original, unfiltered kernel. This directly generates a noise with a filtered spectrum. Contrary to previous methods exploiting a discretization of the spectrum in distinct bands, this approach allows analytical filtering.

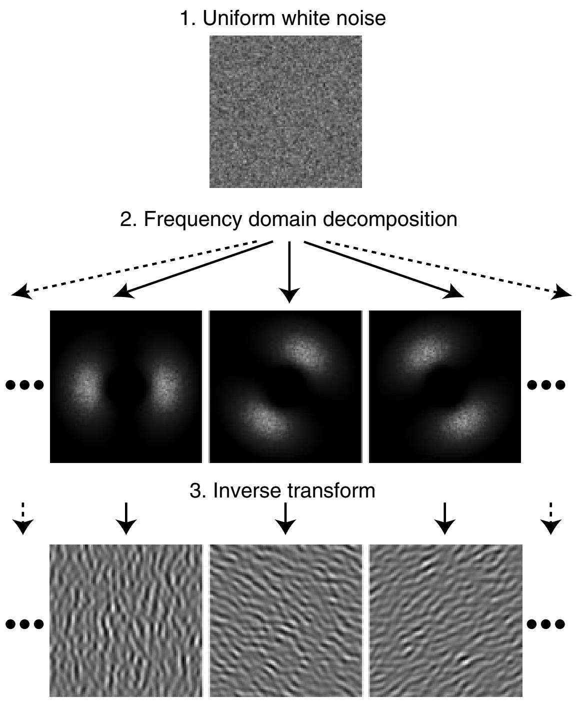

Figure 11: *Anisotropic noise generation. Illustration of spectral noise generation. The frequency domain decomposition has three orientations. Three oriented subbands at the same scale and their corresponding spatial domain images are shown, which are stored as textures. (Figure from [GZD08], ©ACM, 2008.)*

5.2.2. *Filtering noise-based procedural textures*

Noise patterns are rarely used directly to produce textures. Patterns are generated by applying several functions to the noise, such as absolute values or sine waves. In addition, the noise is often coloured by remapping its values to a piecewise linear colour ramp [EMP*02]. Figure 12 illustrates how a marble texture is built from a solid noise.

Because most of these additional operations are non-linear, starting from a filtered noise value does not guarantee that the resulting texture is also filtered. Although this approximation is acceptable when the function applied to the noise is very smooth, proper filtering is in general necessary. For example, consider a black and white pattern obtained by thresholding the noise. A correctly filtered version should progressively introduce blur in the transition areas. However, when only filtering the noise, the transitions will remain sharp due to the subsequent thresholding.

Although this problem remains unsolved in the general case, several approaches provide good approximations when the operations applied to the noise can be summarized in a

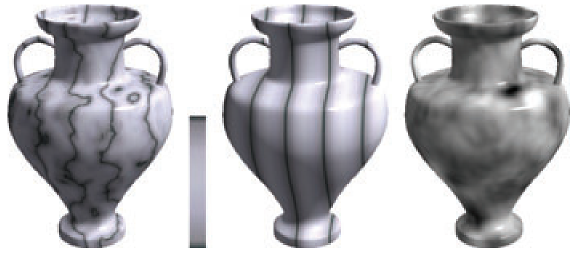

Figure 12: *Procedural texture creation. The marble vase (left) is obtained from two components: A colour map repeated along the x-direction in space (middle), perturbed by a solid noise (right). The final colour is obtained as $C(x + N(x, y, z))$ where C is the 1D colour map, N the noise and x, y, z the surface point coordinates. (Figure based on [LLDD09a],©ACM, 2009.)*

1D *colour table*. The final colour is obtained as $C(N(u))$, where $N(u)$ is the noise value at u and C the colour table.

Rhoades *et al.* [RTB*92] filter the colour table C rather than the noise function. They return the average colour over a small interval $[N(u) - \delta, N(u) + \delta]$. This is conveniently evaluated using MIP-mapping on the 1D colour table. The size of the interval δ is computed from the filter size and the maximum gradient of C with respect to u. Hart *et al.*[HCK99] further refine this approach using the local noise gradient (most noises are differentiable, either through finite differencing or analytically). Although this works well in many cases, one source of error is that the noise value and gradient are evaluated on the unfiltered noise. Lagae *et al.*[LLDD09a] rely on a similar mechanism. They achieve accurate filtering by estimating the noise value range δ from the loss in noise variance due to filtering. This is only possible because an analytical expression of the noise variance is available.

Several authors have investigated more general methods for filtering procedural textures. Heidrich *et al.* [HSS98] presented a method to obtain an average value of a procedural shader with an error bound over a finite area using affine arithmetic. Olano *et al.* [OKS03] presented a method for automatic shader level-of-detail using an automatic system for shader simplification.

6. Analysis of Procedural Noise Functions

In this section, we give a detailed analysis of procedural noise functions. We introduce analysis tools (Section 6.1) and present analysis results (Section 6.2).

6.1. Analysis tools

Motivated by our definition of noise as a stationary and normal stochastic process, we introduce analysis tools for estimating the power spectrum and the amplitude distribution of a noise function. The estimated statistics of a noise function can provide insight into the noise function, also in the case when the expected statistics are known. For example, differences in expected and estimated statistics might reveal mistakes in implementation.

6.1.1. *Power spectrum*

We estimate the power spectrum of a noise function using Bartlett's method of averaging periodograms [Bar78]. The periodogram is a simple estimator for the power spectrum, defined as the magnitude squared of the Fourier transform [PVTF02, 13.4]. However, the periodogram is very noisy. This is because the periodogram is a white noise process with as mean the power spectrum [PP02, 12.2]. Averaging periodograms of different instances of noise results in a less noisy estimate for the power spectrum of a noise function. We inspect both the power spectrum estimate as well as the periodogram, because averaging periodograms averages out noise but can also average out features. We radially average the power spectrum of an isotropic noise function, because the power spectrum of an isotropic noise function is radially symmetric. Note that these methods are also used for power spectrum estimation of Poisson disk distributions [Uli88, LD08].

6.1.2. *Amplitude distribution*

We estimate the amplitude distribution of a noise function using a histogram of noise values. We plot a Gaussian function with the expected variance, in case this is known for the noise function, or with the estimated variance, as a reference.

6.1.3. *Other analysis tools*

Yoon *et al.* [YLC04, YL08] presented several other analysis tools for measuring the quality of a noise within an optimization procedure. Yoon *et al.* used a Chi-square goodness-of-fit test to measure the quality of the amplitude distribution of the noise, a test based on autocorrelation to detect periodicity in the noise, and a band-pass test to measure how band-pass the noise is. The band-pass test is inspired by wavelet noise [CD05], and is based on the difference between the noise and a down-up-sampled version of the noise. However, for our purpose, these analysis tools are subsumed by the ones above.

6.2. Analysis results

We have analysed Perlin noise (using the implementation of [Per02], see Section 3.1.1), sparse convolution noise (as presented in [Lew89], see Section 3.3.1), wavelet noise (as presented in [CD05], see Section 3.2.1), *anisotropic noise* (as presented in [GZD08], see Section 3.2.2), better gradient noise (as presented in [KKS08], see Section 3.1.2) and Gabor noise (as presented in [LLDD09a], see Section 3.3.3).

The parameters of the noise functions were selected to produce a noise with a principal frequency of $1/32$ for the spatial domain and $1/4$ for the frequency domain. Note that both domains require different parameters for optimal visibility. The spatial domain images were tone-mapped by linearly mapping a range of three standard deviations to intensity, and the frequency domain images by linearly mapping the expected maximum of the power spectrum to an intensity value of 80%. One hundred periodograms were used to compute the power spectrum estimate.

We present the results of our analysis in Figure 7.

In row 1, we show the noise generated by the noise functions. Wavelet noise, *anisotropic noise*, better gradient noise and Gabor noise have a very similar aspect. Sparse convolution noise has a different aspect, because it is not designed to be band-pass. Perlin noise has a slightly different aspect, because the noise is zero at every integer lattice point, and because of an undesired axis-aligned anisotropy. Wavelet noise has a very subtle different aspect, because of an undesired axis-aligned anisotropy.

In row 2, we show the amplitude distribution of the noise functions. All noise functions except Perlin noise have an approximately Gaussian amplitude distribution. The amplitude distribution of Perlin noise contains undesired artefacts, because of the limited number of random gradient vectors, and because the noise is zero at every integer lattice point.

In rows 3–5, we show the periodogram, the power spectrum estimate, and the radially averaged power spectrum of the noise functions. All noise functions except Perlin noise and sparse convolution noise are approximately band-pass. Perlin noise is only weakly band-pass, which might lead to problems with aliasing and detail loss [CD05]. Sparse convolution noise is not designed to be band-pass. The horizontal features in the periodogram of Perlin noise are caused by an undesired correlation in the hash function [KKS08]. The axis-aligned square feature in the power spectrum of wavelet noise is caused by the separable B-spline [CD05]. Both features indicate an undesired axis-aligned anisotropy in the noise. Note that the horizontal features in the periodogram of Perlin noise are much less visible in the power spectrum, which is an example of a case where averaging periodograms can also average out features.

We conclude from our analysis that the noise functions are often very different in terms of visual aspect, power spectrum and amplitude distribution. However, as we will show in Section 7, every noise function represents a specific trade-off between a set of features, and this analysis only takes into account a small part of this set of features.

7. Comparison of Procedural Noise Functions

In this section, we give a detailed comparison of procedural noise functions, based on the previous sections.

We compare the same noise functions as the ones we have analysed in Section 6. We present the results of our comparison in Table 1. It is important to note that several developments presented in later methods are also applicable to earlier methods. For example, several developments in better gradient noise and Gabor noise are applicable to Perlin noise and sparse convolution noise respectively. In the table, we compare the methods as presented in the cited works, while in the discussion, we generalize.

In part (a) of the table, we compare to which degree the noise functions adhere to the definition of procedural noise function (Section 2.3). Storage requirements and periodicity are generally linked. Explicit noises have high storage requirements, whereas other noises have low storage requirements. Several authors presented methods to improve storage requirements and periodicity, for example, noise tiles [CD05, YL08, Lag09] and long-period hash functions [LD06]. Sparse convolution noises are non-periodic and have minimal storage requirements.

In part (b) of the table, we compare the noise functions in terms of modelling (Section 4). Band-pass noise functions achieve spectral control using a weighted sum of noise octaves, while sparse convolution noises achieve spectral control using the kernel.

In part (c) of the table, we compare the noise functions in terms of noise on surfaces (Section 5). All noise functions generalize to arbitrary dimensions, and can therefore support solid noise, although the high storage requirements of explicit noises can be problematic for solid noise. Sparse convolution noises can support surface noise, and all noise functions that support band-pass solid noise can support band-pass surface noise.

In part (d) of the table, we summarize the noise functions in terms of filtering (Section 5). All noise functions that are band-pass can support isotropic filtering, while only noise functions that support anisotropic noise can support anisotropic filtering.

In part (e) of the Table, we summarize the analysis of the noise functions (Section 6). Perlin noise does not have a Gaussian amplitude distribution and is only weakly band-pass. Sparse convolution noises are band-pass when the kernel is band-pass.

In part (f) of the Table, we compare the noise functions in terms of speed. In our experience, sparse convolution noises, wavelet noise and better gradient noise are generally slower than Perlin noise, whereas *anisotropic noise* is generally faster. Note that this is a subjective comparison. Sparse convolution noises offer a speed versus

Table 1: *Comparison of procedural noise functions.*

Category	Perlin noise [Per02] Lattice gradient	Sparse convolution noise [Lew89] Sparse convolution	Wavelet noise [CD05] Explicit	Anisotropic noise [GZD08] Explicit	Better gradient noise [KKS08] Lattice gradient	Gabor noise [LLDD09a] sparse convo.
(a) Definition—procedural						
Storage requirements[1]	$O(N)$	$O(1)$	$O(N^d)$	$O(N^d)$	$O(dN)$	$O(1)$
Contin, no discrete data	✓	✓	✓		✓	✓
Non-periodic[2]		✓				✓
Parameters[3]	Weights	Kernel	Weights	Weights (aniso.)	Weights	Kernel param.
(b) Modelling—spectral control						
Spectral control	Weight. sum	Kernel	Weight. sum	Weight. sum (aniso.)	Weight. sum	Kernel param.
Anisotropic noise				✓		✓
(c) Surfaces—noise on surfaces						
Solid noise	✓	✓	✓		✓	✓
Surface noise			✓	✓	✓	✓
Setup-free surface noise			✓		✓	✓
(d) Surfaces—filtering						
Isotropic filtering			✓	✓		✓
Anisotropic filtering				✓		✓
(e) Analysis						
Band-pass power spectrum			✓	✓	✓	✓
Gaussian amplitude distribution		✓	✓	✓	✓	✓
(f) Comparison—speed						
Speed[4]	0	−	−	+	−	−
Quality/speed tradeoff		✓				✓

Note: Comparison of Perlin noise, sparse convolution noise, wavelet noise, anisotropic noise, better gradient noise and Gabor noise. See Section 7 for a correct interpretation of this comparison. (Table based on [LLDD09a], ©ACM, 2009.)

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

[2] Non-periodicity does not take into account the inherent periodicity of computer calculations.

[3] Parameters that are required for spectral control.

[4] Speed is expressed relative to the speed of Perlin noise.

quality tradeoff, but remain slower for acceptable quality levels.

We conclude from our comparison that every noise function represents a specific trade-off between a set of features, and that the noise function that is best suited for a specific application depends heavily on the requirements of that particular application.

8. Conclusion

In this survey, we have provided a critical view on procedural noise methods, including recent solutions developed in the last 8 years. We started by providing a well-founded definition of noise, with a theoretical grounding in stochastic processes. These definitions allowed us to provide a unified classification of the most important procedural noise solutions, providing the reader with a coherent view of the field. In particular, we have classified procedural noise solutions into lattice gradient, explicit and sparse convolution noises. We underline the importance of spectral control when modelling patterns and visual detail with noise. In many cases, noise is applied to surfaces: we have distinguished the different ways to do this, notably via mapping of 2D noise, solid noise and local kernel splatting. Several important issues arise when applying noise to surfaces, most of which have been treated in recent work. In particular, these relate to filtering of noise, as well as the procedural textures based on noise.

We have used the power spectrum and amplitude distribution to analyse procedural noises. This analysis helps explain some of the difference between noises and, in some cases the differences in the resulting visual aspect. We also provided a comparison of the various noises, based on the features that each solution provides. These include storage requirements, the way the power spectrum is controlled, whether anisotropic noise is provided, how each noise can be applied to surfaces and consequent filtering solutions, and of course speed of computation. The conclusion of this comparison is that each solution presents a different tradeoff. Each application needs to determine the relevant importance of each feature to determine which noise is most appropriate for the problem at hand.

8.1. Future work

We next discuss several challenging directions for future work.

8.1.1. *Fine-grained control over the power spectrum*

In Section 2, we have explained that noise is completely characterized by its power spectrum, and in Section 4, we have discussed several methods for controlling the power spectrum of noise. However, these methods only offer coarse-grained control over the power spectrum. Fine-grained control over the power spectrum would bridge the gap between noise and stochastic texture. This is illustrated by Lewis [Lew86, Figure 6] and by Lagae *et al.* [LLDD09a, Figure 7]. Finer control over the power spectrum is an interesting direction for future work; it is particularly important for the development of noise-by-example methods.

8.1.2. *Non-Gaussian and non-stationary stochastic processes*

In Section 2, we have defined noise as a stationary and Gaussian stochastic process. Generalizing this definition directly suggests two ways to extend noise: non-stationary and non-Gaussian stochastic processes. Non-stationary stochastic processes correspond to spatially varying noise. Lewis [Lew86] already hinted at spatially varying sparse convolution noise, van Wijk [vW91] used spatially varying spot noise for visualizing scalar fields, and Lagae *et al.* [LLD10] recently explored spatially varying Gabor noise. However, all the above solutions have limitations. Developing a general and efficient solution allowing spatial variation of all noise parameters is an interesting direction for future research. Non-Gaussian stochastic processes correspond to more general random patterns. This could bridge the gap between noise and non-stochastic texture, or between noise and parametric texture synthesis [PNNT96, PS00]. Although some authors have investigated non-Gaussian processes, for example, Lewis [Lew86], Gagalowicz and Ma [GM85], and, more recently, Chainais [Cha07], research into non-Gaussian noise is very limited. The development of non-Gaussian noise would provide a powerful tool to model much richer patterns. It is however unclear whether such an approach is the most effective way to obtain such results. In addition, developing general solutions for non-Gaussian noise requires substantial future research.

8.1.3. *Understanding and controlling phase*

In Section 2, we also noted that a noise is fully specified by its power spectrum, but has random phase. This characterization suggests that the distinction between a stochastic texture and a structured texture or pattern is manifested in the non-random phase of the latter. Identifying and controlling useful information in the phase might be another approach to bridging the gap between stochastic and structured textures. This is an unexplored topic.

8.1.4. *Faster noise and benchmarking noise*

Because the introduction of Perlin noise more than two decades ago, making noise faster continues to be important, especially for industrial applications. This is perhaps best illustrated by the famous quote: *90% of 3D rendering time is spent in shading, and much of that time is spent computing Perlin noise*[3]. In Section 7, we have not performed an extensive benchmark to compare the speed of different noise functions. This is mainly because of the practical issues involved, such as coding optimized implementations for different architectures (CPU, GPU). However, such a benchmark would be interesting. As mentioned in the introduction, the relative cost of computation and memory access has changed significantly over time, to the point where current program execution times are often dominated by memory access. This is one reason for the renewed interest in procedural noise, that is the hope of exchanging bandwidth for computation.

8.1.5. *Better authoring tools*

In Section 4, we have mentioned some authoring tools for noise. However, the work done in graphical user interfaces for noise, in noise editing, and in noise by example is very limited.

8.1.6. *Filtering procedural textures based on noise*

In Section 5, we have discussed filtering of procedural textures based on noise. Although solutions for specific cases are available, the general problem of filtering procedural textures based on noise is still unsolved.

Acknowledgements

The authors thank the anonymous reviewers and Ian Jermyn and Bruno Galerne. We are grateful to the NVIDIA Professor Partnership Program for equipment donations. Ares Lagae is a Postdoctoral Fellow of the Research Foundation—Flanders (FWO), and acknowledges K.U. Leuven CREA funding (CREA/08/017). Sylvain Lefebvre receives support from the Agence Nationale de la Recherche (SIMILAR-CITIES ANR-2008-COORD-021-01).

[3] Eric Enderton, Industrial Light & Magic (personal communication, 1994).

References

[Anj88] Anjyo K.: A simple spectral approach to stochastic modeling for natural objects. In *Eurographics '88: Proceedings of the European Graphics Conference and Exhibition* (Nice, France, 1988), pp. 285–296.

[AW90] Abram G. D., Whitted T.: Building block shaders. In *Computer Graphics (Proceedings of ACM SIGGRAPH'90)* (1990), vol. 24, pp. 283–288.

[Bar78] Bartlett M. S.: *An Introduction to Stochastic Processes: With Special Reference to Methods and Applications* (3rd edition). Cambridge University Press, Cambridge, UK, 1978.

[BD04] Bourque E., Dudek G.: Procedural texture matching and transformation. *Computer Graphics Forum*, *23*, 3 (2004), 461–468.

[BHN07] Bridson R., Houriham J., Nordenstam M.: Curl-noise for procedural fluid flow. *ACM Transactions on Graphics*, *26*, 3 (2007), 46:1–46:8.

[BLV*10] Benard P., Lagae A., Vangorp P., Lefebvre S., Drettakis G., Thollot J.: A dynamic noise primitive for coherent stylization. *Computer Graphics Forum (Proceedings of the 20th Eurographics Symposium on Rendering)*, *29*, 4 (2010), 1497–1506.

[Bra99] Bracewell R. N.: *The Fourier Transform and its Applications* (3rd edition). McGraw-Hill, New York, USA, 1999.

[CD05] Cook R. L., DeRose T.: Wavelet noise. *ACM Transactions on Graphics*, *24*, 3 (2005), 803–811.

[Cha07] Chainais P.: Infinitely divisible cascades to model the statistics of natural images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*, 12 (2007), 2105–2119.

[DG97] Dischler J.-M., Ghazanfarpour D.: A procedural description of geometric textures by spectral and spatial analysis of profiles. *Computer Graphics Forum*, *16*, 3 (1997), C129–C139.

[DG01] Dischler J. M., Ghazanfarpour D.: A survey of 3d texturing. *Computers and Graphics*, *25*, 1 (2001), 135–151.

[EMP*02] Ebert D. S., Musgrave F. K., Peachey D., Perlin K., Worley S.: *Texturing and Modeling: A Procedural Approach* (3rd edition). Morgan Kaufmann Publishers, Inc., Massachusetts, USA, 2002.

[FFC82] Fournier A., Fussell D., Carpenter L.: Computer rendering of stochastic models. *Communications of the ACM*, *25*, 6 (1982), 371–384.

[FSDH07] Fisher M., Schröder P., Desbrun M., Hoppe H.: Design of tangent vector fields. *ACM Transactions on Graphics*, *26*, 3 (2007), 56:1–56:9.

[FW07] Frisvad J. R., Wyvill G.: Fast high-quality noise. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia* (Perth, Australia, 2007), pp. 243–248.

[Gar84] Gardner G. Y.: Simulation of natural scenes using textured quadric surfaces. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), vol. 18, pp. 11–20.

[Gar85] Gardner G. Y.: Visual simulation of clouds. *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, *19*, 3 (1985), 297–304.

[GD95] Ghazanfarpour D., Dischler J.-M.: Spectral analysis for automatic 3-d texture generation. *Computers and Graphics*, *19*, 3 (1995), 413–422.

[GD96] Ghazanfarpour D., Dischler J.-M.: Generation of 3d texture using multiple 2d models analysis. *Computer Graphics Forum*, *15*, 3 (1996), 311–323.

[GD10] Gilet G., Dischler J.-M.: An image-based approach for stochastic volumetric and procedural details. *Computer Graphics Forum (Proceedings of the 20th Eurographics Symposium on Rendering)*, *29*, 4 (2010), 1411–1419.

[GDS10] Gilet G., Dischler J.-M., Soler L.: Procedural descriptions of anisotropic noisy textures by example. *EG 2010—Short Papers*(2010).

[GGM09] Galerne B., Gousseau Y., Morel J.-M.: *Random Phase Textures: Theory and Synthesis*. Tech. Rep. 24, Centre de Mathématiques et de Leurs Applications, 2009.

[GM85] Gagalowicz A., Ma S. D.: Model driven synthesis of natural textures for 3-d scenes. In *Eurographics '85: Proceedings of the European Graphics Conference and Exhibition* (Nice, France, 1985), pp. 91–108.

[GZD08] Goldberg A., Zwicker M., Durand F.: Anisotropic noise. *ACM Transactions on Graphics*, *27*, 3 (2008), 54:1–54:8.

[Har01] Hart J. C.: Perlin noise pixel shaders. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (Los Angeles, CA, USA, 2001), pp. 87–94.

[HB95] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis/synthesis. In *Proceedings of ACM SIGGRAPH 1995* (Los Angeles, CA, USA, 1995), pp. 229–238.

[HCK99] HART J. C., CARR N., KAMEYA M.: Antialiased parameterized solid texturing simplified for consumer-level hardware implementation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (Los Angeles, CA, USA, 1999), pp. 45–53.

[Hec89] HECKBERT P. S.: *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, CS Division, U.C. Berkeley, 1989.

[HSS98] HEIDRICH W., SLUSALLEK P., SEIDEL H.-P.: Sampling procedural shaders using affine arithmetic. *ACM Transactions in Graphics*, *17*, 3 (1998), 158–176.

[Jul62] JULESZ B.: Visual pattern discrimination. *IEEE Transactions on Information Theory*, *8* (1962), 84–92.

[KKS08] KENSLER A., KNOLL A., SHIRLEY P.: *Better Gradient Noise*. Tech. Rep. UUSCI-2008-001, SCI Institute, University of Utah, 2008.

[Lag09] LAGAE A.: *Wang Tiles in Computer Graphics*. Morgan & Claypool Publishers, California, USA, 2009.

[LD05] LAGAE A., DUTRÉ P.: A procedural object distribution function. *ACM Transactions on Graphics*, *24*, 4 (2005), 1442–1461.

[LD06] LAGAE A., Dutré P.: Long-period hash functions for procedural texturing. In *Proceedings of the Vision, Modeling, and Visualization 2006* (Aachen, Germany, 2006), pp. 225–228.

[LD08] LAGAE A., DUTRÉ P.: A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum*, *27*, 1 (2008), 114–129.

[Lew84] LEWIS J. P.: Texture synthesis for digital painting. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)* (1984), vol. 18, pp. 245–252.

[Lew86] LEWIS J. P.: Methods for stochastic spectral synthesis. In *Proceedings on Graphics Interface '86/Vision Interface'86* (Vancouver, British Columbia, Canada, 1986), pp. 173–179.

[Lew87] LEWIS J. P.: Generalized stochastic subdivision. *ACM Transactions on Graphics*, *6*, 3 (1987), 167–190.

[Lew89] LEWIS J. P.: Algorithms for solid noise synthesis. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)* (1989), vol. 23, pp. 263–270.

[LLD10] LAGAE A., LEFEBVRE S., DUTRÉ P.: *Improving Gabor Noise*. IEEE Transactions on Visualization and Computer Graphics, 2010. (to appear).

[LLDD09a] LAGAE A., LEFEBVRE S., DRETTAKIS G., Dutré P.: Procedural noise using sparse Gabor convolution. *ACM Transactions on Graphics*, *28*, 3 (2009), 54:1–54:10.

[LLDD09b] LAGAE A., LEFEBVRE S., DRETTAKIS G., Dutré P.: *Procedural Noise using Sparse Gabor Convolution - Auxiliary Material*. Report CW 545, Department of Computer Science, K.U. Leuven , 2009.

[LN03] LEFEBVRE S., NEYRET F.: Pattern based procedural textures. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (Monterey, CA, USA, 2003), pp. 203–212.

[LVLD10] LAGAE A., VANGORP P., LENAERTS T., Dutré P.: Procedural isotropic stochastic textures by example. *Computers & Graphics (Special issue on Procedural Methods in Computer Graphics)*, *34*, 4 (2010), 312–321.

[LZD09] LAGAE A., ZWICKER M., Dutré P.: *On Anisotropic Noise*. Report CW 547, Department of Computer Science, K.U. Leuven , 2009.

[Mal93] MALZBENDER T.: Fourier volume rendering. *ACM Transactions on Graphics*, *12*, 3 (1993), 233–250.

[Man82] MANDELBROT B. B.: *The Fractal Geometry of Nature*. W. H. Freeman, New York, USA, 1982.

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)* (1989), vol. 23, pp. 41–50.

[NRS82] NORTON A., ROCKWOOD A. P., SKOLMOSKI P. T.: Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space. In *Computer Graphics (Proceedings of ACM SIGGRAPH 82)* (1982), vol. 16, pp. 1–8.

[OHH*02] OLANO M., HART J. C., HEIDRICH W., MARK B., PERLIN K.: Real-time shading languages. SIGGRAPH 2002 Course 36 , 2002.

[OKS03] OLANO M., KUEHNE B., SIMMONS M.: Automatic shader level of detail. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (San Diego, CA, USA, 2003), pp. 7–14.

[Ola05] OLANO M.: Modified noise for evaluation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2005), pp. 105–110.

[Pap71] PAPOULIS A.: High density shot noise and Gaussianity. *Journal of Applied Probability*, *8*, 1 (1971), 118–127.

[Pea85] PEACHY D. R.: Solid texturing of complex surfaces. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)* (1985), vol. 19, pp. 279–286.

[Per85] PERLIN K.: An image synthesizer. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)* (1985), vol. 19, pp. 287–296.

[Per02] PERLIN K.: Improving noise. In *Proceedings of ACM SIGGRAPH 2002* (San Antonio, TX, USA, 2002), pp. 681–682.

[PH89] PERLIN K., HOFFERT E. M.: Hypertexture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)* (1989), vol. 23, pp. 253–262.

[PN01] PERLIN K., NEYRET F.: Flow noise. In *Proceedings of the ACM SIGGRAPH Technical Sketches and Applications* (Los Angeles, CA, USA, 2001), p. 187.

[PNNT96] PORTILLA J., NAVARRO R., NESTARES O., TABERNERO A.: Texture synthesis-by-analysis method based on a multiscale early-vision model. *Optical Engineering*, *35*, 8 (1996), 2403–2417.

[PP02] PAPOULIS A., PILLAI U.: *Probability, Random Variables and Stochastic Processes* (4th edition). McGraw-Hill, New York, USA, 2002.

[PS00] PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, *40*, 1 (2000), 49–70.

[PVTF02] PRESS W. H., VETTERLING W. T., TEUKOLSKY S. A., FLANNERY B. P.: *Numerical Recipes in C++: The Art of Scientific Computing* (2nd edition). Cambridge University Press, Cambridge, UK, 2002.

[QY02] QIN X., YANG Y.-H.: Estimating parameters for procedural texturing by genetic algorithms. *Graphical Models*, *64*, 1 (2002), 19–39.

[Ros06] ROST R. J.: *OpenGL Shading Language* (2nd edition). Addison-Wesley, Reading, MA, USA, 2006.

[RTB*92] RHOADES J., TURK G., BELL A., STATE A., NEUMANN U., VARSHNEY A.: Real-time procedural textures. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics* (Cambridge, MA, USA, 1992), pp. 95–100.

[Sau88] SAUPE D.: Algorithms for random fractals. In *The Science of Fractal Images*. Heinz-Otto Peitgen and Dietmar Saupe (Eds.). Springer-Verlag, New York, Inc., 1988, pp. 71–113.

[SF95] SIMONCELLI E. P., FREEMAN W. T.: The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Proceedings of the 1995 International Conference on Image Processing* (Vancouver, BC, Canada, 1995), p. 3444.

[SKB09] SPJUT J. B., KENSLER A. E., BRUNVAND E. L.: Hardware-accelerated gradient noise for graphics. In *Proceedings of the 19th ACM Great Lakes Symposium on VLSI* (Boston Area, MA, USA, 2009), pp. 457–462.

[ST89] SZELISKI R., TERZOPOULOS D.: From splines to fractals. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)* (1989), vol. 23, pp. 51–60.

[TW08] TZENG S., WEI L.-Y.: Parallel white noise generation on a GPU via cryptographic hash. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (Redwood City, CA, USA, 2008), pp. 79–87.

[Uli88] ULICHNEY R.: Dithering with blue noise. *Proceedings of the IEEE*, *76*, 1 (1988), 56–79.

[Vos88] VOSS R. F.: Fractals in nature: from characterization to simulation. In *The Science of Fractal Images*. Springer-Verlag, New York, 1988, pp. 21–70.

[vW91] VAN WIJK J. J.: Spot noise texture synthesis for data visualization. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)* (1991), vol. 25, pp. 309–318.

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Proceeding of the EG 2009 — State of the Art Reports* (Munich, Germany, 2009), pp. 93–117.

[WN99] WYVILL G., NOVINS K.: Filtered noise and the fourth dimension. In *Proceedings of the ACM SIGGRAPH 99 Conference Abstracts and Applications* (Los Angeles, CA, USA, 1999), p. 242.

[Wor96] WORLEY S.: A cellular texture basis function. In *Proceedings of ACM SIGGRAPH 1996* (New Orleans, LA, USA, 1996), pp. 291–294.

[YL08] YOON J.-C., LEE I.-K.: Stable and controllable noise. *Graphical Models*, *70*, 5 (2008), 105–115.

[YLC04] YOON J.-C., LEE I.-K., CHOI J.-J.: Editing noise. *Computer Animation and Virtual Worlds*, *15*, 3–4 (2004), 277–287.

Appendix. Background on Texture Filtering

This section explains the basic concepts for filtering signals mapped to surfaces. We describe here the general approach, which applies both to bitmap textures, procedural textures or any other signal applied to a surface. The intent is only to introduce the basics concepts needed for Section 5.2. Please refer to Heckbert [Hec89] for a more detailed account of texture filtering.

The filter to be applied to the texture varies in each screen pixel, as it depends on the viewing condition. In screen space, each pixel is typically modelled as a small Gaussian filter (but this could be any other filter). The colour of the pixel should be obtained as the integral of the filter multiplied by the visible portion of the texture. In other words, this evaluates at the pixel the convolution between the filter and the projected texture. To simplify computations, the filter is back-projected in the texture domain through the view transformation and through the mapping used to apply the texture onto the visible surface. This results in a distorted Gaussian in the texture domain (Figure 10).

More precisely, the filter for pixel (x, y) in image space is the Gaussian:

$$f(x, y) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2\sigma^2}(x^2+y^2)}, \tag{A.1}$$

where σ is the width of the Gaussian in pixels (typically $\sigma = 1.0$). The corresponding filter in the texture space is the Gaussian

$$f(J^{-1}[u\ v]^{\mathrm{T}}) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2\sigma^2}[u\ v]J^{-1^{\mathrm{T}}}J^{-1}[u\ v]^{\mathrm{T}}}, \tag{A.2}$$

where J is a local affine approximation (Jacobian) of the mapping from image to texture coordinates. It is easily estimated in every pixel using the screen space derivatives of the texture coordinates (rasterization) or ray differentials (ray-tracing):

$$J = \begin{bmatrix} \dfrac{\mathrm{d}u}{\mathrm{d}x} & \dfrac{\mathrm{d}u}{\mathrm{d}y} \\ \dfrac{\mathrm{d}v}{\mathrm{d}x} & \dfrac{\mathrm{d}v}{\mathrm{d}y} \end{bmatrix}. \tag{A.3}$$

Note that this Gaussian filter remains a Gaussian in the spectral domain. The corresponding *frequency domain* filter is the Gaussian:

$$F\left(J^{\mathrm{T}}[f_u\ f_v]^{\mathrm{T}}\right) = \mathrm{e}^{-2\pi^2\sigma^2[f_u\ f_v]JJ^{\mathrm{T}}[f_u\ f_v]^{\mathrm{T}}}. \tag{A.4}$$

We thus can easily relate the filter to the spectrum of the texture. A properly filtered texture will have for spectrum the multiplication between its unfiltered spectrum and the filter spectrum. This follows from the convolution theorem: The spatial domain convolution simply becomes a multiplication in the spectral domain.