Procedural Isotropic Stochastic Textures by Example

Ares Lagae[1], Peter Vangorp, Toon Lenaerts, Philip Dutré

Department of Computer Science, Katholieke Universiteit Leuven

Abstract

Image textures can easily be created using texture synthesis by example. However, creating procedural textures is much more difficult. This is unfortunate, since procedural textures have significant advantages over image textures. In this paper we address the problem of texture synthesis by example for procedural textures. We introduce a method for procedural multiresolution noise by example. Our method computes the weights of a procedural multiresolution noise, a simple but common class of procedural textures, from an example. We illustrate this method by using it as a key component in a method for texture synthesis by example for isotropic stochastic procedural textures. Our method significantly facilitates the creation of these procedural textures.

Keywords: procedural texture, isotropic stochastic texture, texture synthesis by example, texture synthesis/analysis, noise, procedural noise, wavelet noise, stochastic modeling, solid texture

1. Introduction

Texturing is a fundamental tool in computer graphics to increase the visual complexity of computer-generated imagery. There are two important classes of textures: image textures consisting of raster data and procedural textures. Procedural textures have several potential advantages over image textures [Ebert et al., 2002]:

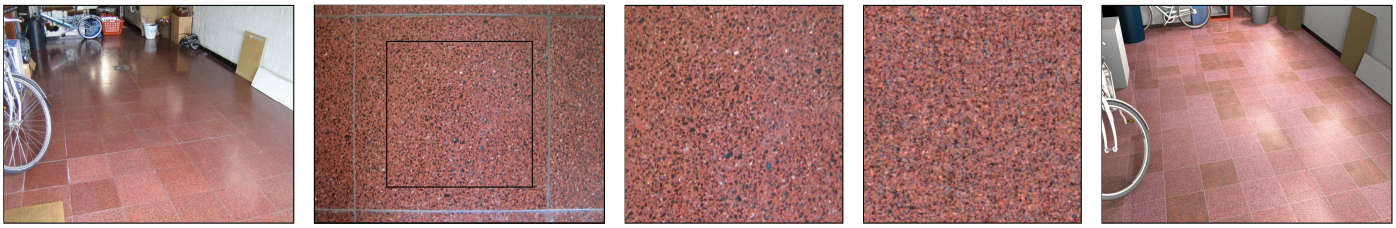
- Procedural textures are compact, their storage requirements are orders of magnitude smaller than those of image textures.
- Procedural textures have no fixed resolution and size, they have an infinite extent and are not subject to undesired seams and repetitions.
- Procedural textures are parametrized, they represent a class of related textures rather than a single texture, and leave room for an artist to tweak.
- Solid procedural textures do not require texture coordinates and are not subject to undesired seams and distortions.
- Procedural textures often allow high-quality anti-aliasing [Ebert et al., 2002; Cook and DeRose, 2005].

Procedural textures can therefore be an attractive alternative to image textures. This is witnessed by the fact that procedural textures are increasingly popular in production rendering. For example, *Pixar* recently developed wavelet noise [Cook and DeRose, 2005], and *Blue Sky Studios* uses a completely procedural approach to texturing for animated feature films in their renderer *CGIStudio* [Eringis, 2006]. However, creating procedural textures can be difficult. This typically involves either some sort of programming language, such as the *RenderMan* shading language [Pixar, 2005], or an interactive visual interface, such as *MaPZone* [Allegorithmic], accompanied by a deep understanding of procedural textures. In contrast, creating image textures is easy, and can be done by simply taking a digital photograph, or by using texture synthesis by example [Wei et al., 2009].

In this paper we address this problem by exploring texture synthesis by example for procedural textures. We believe that investigating automatic methods for creating procedural textures is definitely worthwhile, given the previously mentioned difficulties with existing manual methods. Because procedural textures are typically created starting from a procedural noise function, such as Perlin noise, using a process similar to function composition [Perlin, 1985], they can be arbitrarily complex. Therefore, we limit to a specific class of procedural textures, multiresolution noise [Perlin, 1985; Ebert et al., 2002; Cook and DeRose, 2005], a simple but common class of procedural textures, and present a method for procedural multiresolution noise by example. Multiresolution noise roughly corresponds to the class of isotropic stochastic textures. We present a method for procedural isotropic stochastic textures by example, by combining our method for procedural multiresolution noise by example with existing techniques from texture synthesis. This is illustrated in figure 1. For isotropic stochastic textures, our method is similar to the one of Heeger and Bergen [1995] but produces a procedural texture rather than an image texture, that is randomly accessible and can be evaluated on-the-fly, for example in a GPU shader.

Email address: ares.lagae@cs.kuleuven.be (Ares Lagae)

[1]Ares Lagae is visiting at REVES/INRIA Sophia-Antipolis in 2009-2010.



(a) (b) (c) (d) (e)

Figure 1: Our method for texture synthesis by example for isotropic stochastic procedural textures. (a) A photograph of a real-world scene. (b) A photograph of a texture in the scene. (c) A cropped version of the photograph of the texture. (d) A procedural texture automatically generated from the cropped version of the photograph using our method. (e) A rendering of a virtual scene textured using several of these procedural textures.

2. Related Work

Our work is related to procedural multiresolution noise, and to texture synthesis by example for image textures and for procedural textures.

Procedural multiresolution noise was introduced by Perlin [1985] in the same work that introduced solid modeling and Perlin noise. Cook and DeRose [2005] introduced a more band-limited procedural noise function that improves modeling with and filtering of multiresolution noise. Methods related to multiresolution noise were introduced by Lewis [1989], and recently by Goldberg et al. [2008] and Lagae et al. [2009]. Goldberg et al. use an automatic method to compute the parameters of an anisotropic multiresolution noise. However, their application is compensating for parametric distortion, and their noise function is not procedural. In all of the other methods, the parameters of the multiresolution noise are manipulated manually, and none of the methods addresses the problem of multiresolution noise by example. For an overview of procedural noise functions, see Lagae et al. [2010].

There is a large amount of work that addresses texture synthesis by example for image textures. This includes parametric methods [Heeger and Bergen, 1995; Portilla and Simoncelli, 2000], non-parametric methods [Bonet, 1997], including pixel-based methods [Efros and Leung, 1999; Wei and Levoy, 2000] as well as patch-based methods [Efros and Freeman, 2001; Kwatra et al., 2003], and optimization-based methods [Kwatra et al., 2005]. Some methods also consider solid textures [Heeger and Bergen, 1995; Dischler et al., 1998; Kopf et al., 2007; Dong et al., 2008]. For a recent overview, refer to Wei et al. [2009]. Our method is similar to parametric methods for texture synthesis, such as the work by Heeger and Bergen [1995], Portilla et al. [1996], and Portilla and Simoncelli [2000]. However, all of these methods produce image textures, while our method produces procedural textures.

Work that addresses texture synthesis by example for procedural textures is much less common. Ghazanfarpour and Dischler [1995; 1996] presented a spectral method for automatic solid procedural texture generation from a single or multiple 2D exemplars. However, their method is designed for textures with only a few dominant frequencies, which is a different texture class than the one we address in this work. Dischler and Ghazanfarpour [1997] also presented a method for automatically generating procedural geometric textures similar to the one described in this paper. We discuss the differences between both methods in section 6. Lefebvre and Poulin [2000] introduced a system to extract values for parameters of structural textures from photographs. However, their method is limited to rectangular tiling and wood textures. Qin and Yang [2002] introduced a genetic-based multiresolution parameter estimation approach to recover the parameter values for a given procedural texture, and Bourque and Dudek [2004] introduced a system that performs a two-phase search over a library of procedural shaders. However, these systems are limited to textures that are represented in the library, and determining the parameters of a procedural texture using a search method is less efficient and less accurate than a direct computation, such as the one presented in this paper.

Recently, there is some convergence between texture synthesis by example for image textures and for procedural textures. There is some work that tries to port advantages of procedural textures, such as random accessibility [Lefebvre and Hoppe, 2005; Dong et al., 2008], resolution independence [Han et al., 2008] and compactness [Wei et al., 2008], to texture synthesis by example for image textures. In this work we try to port one of the advantages of image textures, easy creation using texture synthesis by example, to procedural textures.

3. Procedural Multiresolution Noise

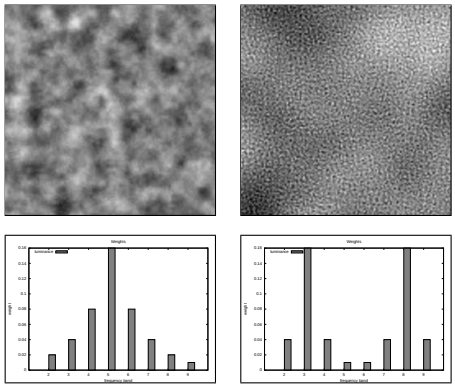
Multiresolution noise M is defined as a weighted sum of noise bands N_i,

$$M(x,y) = \sum_i w_i N_i(x,y). \tag{1}$$

The appearance of the noise M is determined by the weights w_i. The noise bands N_i are scaled versions of a single noise band N, typically a procedural noise function, where consecutive noise bands are scaled by a factor of two, and translated using a random offset o_i,

$$N_i(x,y) = N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right]. \tag{2}$$

The random offsets decorrelate the noise bands by randomly shifting them with respect to each other. The noise bands should ideally be orthogonal [Cook and DeRose, 2005]. In that case, the weights accurately control the spectral shape of the multiresolution noise, and the noise can be filtered using frequency



(a) (b)

Figure 2: Procedural multiresolution noise. (a) A noise with mainly mid-frequency content. (b) A noise with mainly low and high-frequency content. Each example shows the noise and the weights.

clamping [Norton et al., 1982]. We therefore use wavelet noise [Cook and DeRose, 2005] rather than Perlin noise [Perlin, 1985, 2002]. Procedural multiresolution noise is illustrated in figure 2. For more information on multiresolution noise, refer to Perlin [1985], Ebert et al. [2002] and Cook and DeRose [2005].

4. Procedural Multiresolution Noise by Example

In this section we present our method for procedural multiresolution noise by example. Given an exemplar E, for example a grayscale photograph of a texture, our method computes the weights w_i of a multiresolution noise M, such that the noise M appears similar to the exemplar E,

$$M(x,y) = \sum_i w_i \, N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right] \approx E(x,y). \quad (3)$$

We assume that the reader is familiar with Fourier analysis (see e.g. Bracewell [1999]).

4.1. Derivation

In this subsection, we derive a direct formula to compute the weight w_i from the exemplar E. First, we apply the Fourier transform $\mathcal{F}$ to both sides of equation 3,

$$\sum_i w_i \mathcal{F}\left\{N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right]\right\} \approx \mathcal{F}\left[E(x,y)\right]. \quad (4)$$

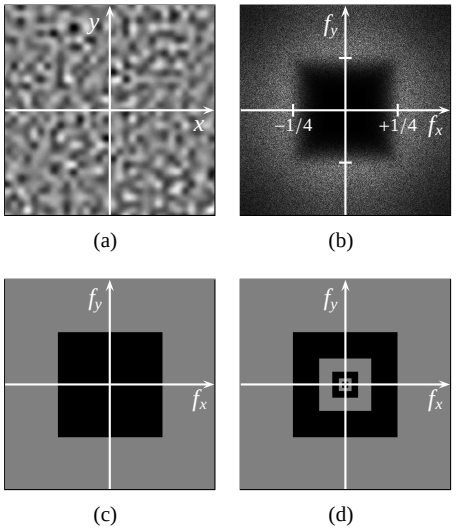
Note that $\mathcal{F}$ can be distributed inside the summation since it is a linear operator. Then, we define a decomposition of frequency space D, where D_i indicates the support of the noise band N_i in the frequency domain. For wavelet noise, we define D by

$$D_0\left(f_x, f_y\right) = \begin{cases} 1 & |f_x|, \left|f_y\right| < 1/2 \text{ and } |f_x|, \left|f_y\right| > 1/4 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

$$D_i\left(f_x, f_y\right) = D_0\left(2^i f_x, 2^i f_y\right), \quad (6)$$

where D_0 is the support of wavelet noise in the frequency domain. This definition follows from the construction of wavelet noise [Cook and DeRose, 2005], $D_0\left(f_x, f_y\right)$ equals 0 for $|f_x|, \left|f_y\right| < 1/4$ since wavelet noise is band-limited. Note that

$$D_i \mathcal{F}\left(N_j\right) = \begin{cases} \mathcal{F}\left(N_j\right) & i = j \\ 0 & \text{otherwise} \end{cases}. \quad (7)$$



(a) (b) (c) (d)

Figure 3: Decomposition of frequency space for wavelet noise. (a) Wavelet noise. (b) The power spectrum of wavelet noise (f_x, $f_y \in [-1/2, +1/2]$). (c) The support of wavelet noise in the frequency domain (D_0). (d) The decomposition of frequency space (D_i) (the support of consecutive noise bands is shown alternating in gray and black).

This decomposition of frequency space is illustrated in figure 3. Next, we multiply both sides of equation 4 by D_i in order to isolate the weight w_i,

$$w_i \mathcal{F}\left\{N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right]\right\} \approx D_i \mathcal{F}\left[E(x,y)\right]. \quad (8)$$

Note that D_i can be distributed inside the summation since it is a linear operator, and that all terms in the summation, except the term involving w_i, vanish due to equation 7. At this point, it might seem as if the weight w_i can be obtained by a simple division. However, this is not the case, since the noise band N is stochastic, and its value is not known. Therefore, we exploit the statistical properties of the noise band N. We proceed by taking the power of both sides of equation 8,

$$w_i^2 \int \left|\mathcal{F}\left\{N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right]\right\}\right|^2 \approx \int \left|D_i \mathcal{F}\left[E(x,y)\right]\right|^2. \quad (9)$$

Then, we use Rayleigh's theorem, which states that the power is the same in the spatial domain and in the frequency domain,

$$\int \left|\mathcal{F}\left\{N\left[2^i (x + o_{i,x}), 2^i \left(y + o_{i,y}\right)\right]\right\}\right|^2 = \int \left|N(x,y)\right|^2. \quad (10)$$

Note that the scale 2^i and offset o_i can be dropped, since the variance, and therefore the power, of all noise bands N_i is the same. Next, we approximate the power in the noise band N by the expected power, noting that wavelet noise has a Gaussian intensity distribution with zero mean and variance $\sigma_N^2 \approx 0.265$ [Cook and DeRose, 2005], and using the computational formula for the variance,

$$\int \left|N(x,y)\right|^2 \approx \left\langle \left|N(x)\right|^2 \right\rangle = \sigma_N^2 \approx 0.265, \quad (11)$$

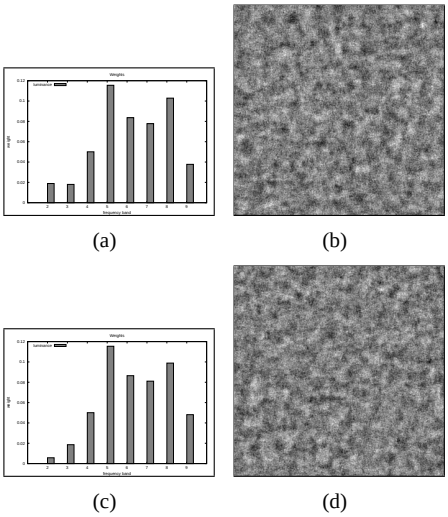


Figure 4: Validation of our method for procedural multiresolution noise by example. (a) A set of random weights. (b) A noise generated using the random weights. (c) A set of weights computed from the noise using our method. (d) A noise generated using the computed weights.

where $\langle\rangle$ denotes expectation value. Finally, we substitute equations 10 and 11 into equation 9 and obtain a direct formula for the weight w_i,

$$w_i \approx \sqrt{\frac{\int \left|D_i \mathcal{F}\left[E\left(x,y\right)\right]\right|^2}{\sigma_N^2}}. \tag{12}$$

4.2. Implementation

We obtain the term $\int \left|D_i \mathcal{F}\left[E\left(x,y\right)\right]\right|^2$ of equation 12 for all i by interpreting the exemplar E as a matrix, computing the Fourier transform, iterating over all elements, sorting the elements into their corresponding frequency band i, and accumulating the power in each frequency band. Note that this is a simple method for power spectrum estimation. We obtain the frequency band i from an element with frequency $\left(f_y, f_x\right)$ as

$$i = \mathrm{int}\left\{\min[\log_2\left(1/|f_x|\right), \log_2\left(1/|f_y|\right)]\right\} + 1, \tag{13}$$

which follows from equations 5 and 6.

4.3. Validation

We present a validation of our method for procedural multiresolution noise by example in figure 4. We construct an exemplar E with known weights by generating a set of random weights (figure 4(a)), and generate a noise using the random weights (figure 4(b)). We obtain a noise M by computing a set of weights from the exemplar using our method (figure 4(c)), and generating a noise using the computed weights (figure 4(d)). The exemplar E and the noise M appear very similar, they could be different samples from the same procedural noise function. The two sets of weights are also very similar. Note that, even though the exemplar is constructed using multiresolution noise, the two sets of weights are not exactly the same. This is because of the stochastic nature of noise. More specifically, the power in each noise band of a finite sample of noise is not exactly equal to its expected power determined by the weights, although the power converges to the expected power with increasing sample size.

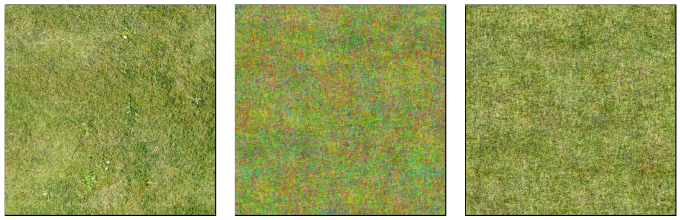


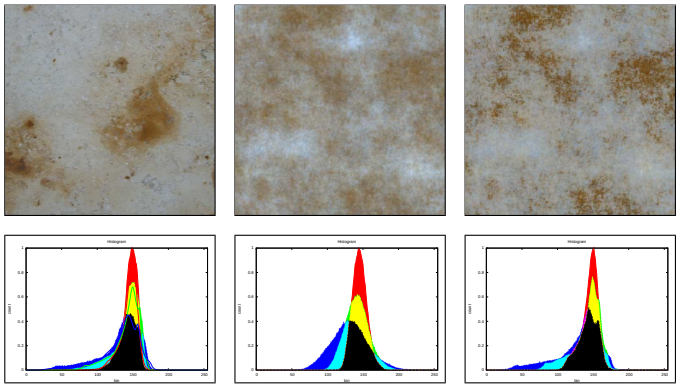
Figure 5: Multi-channel synthesis. (a) A photograph of a texture. (b) A synthesis result generated using multi-channel synthesis in an RGB color space. (c) A synthesis result generated using multi-channel synthesis in a decorrelated color space.

5. Texture Synthesis by Example for Isotropic Stochastic Procedural Textures

Multiresolution noise roughly corresponds to the class of isotropic stochastic textures. In this section, we therefore present a method for texture synthesis by example for isotropic stochastic procedural textures, based on our method for multiresolution noise by example. Like many other methods [Heeger and Bergen, 1995; Portilla et al., 1996; Dischler and Ghazanfarpour, 1997; Portilla and Simoncelli, 2000], our method is motivated by the seminal work of Julesz [1962], which states that textures with similar first and second order statistics, or histogram and power spectrum, are difficult to discriminate. Therefore, we combine our method for multiresolution noise by example, which matches the power spectrum, with a method for matching the histogram. We also extend our method for multiresolution noise by example to color textures. We use existing techniques from texture synthesis, without compromising the compactness and random accessibility of the procedural texture. We summarize our method as a texture-synthesis-by-analysis method.

5.1. Multi-Channel Synthesis

Multiresolution noise is limited to single-channel textures, because it is a scalar function. The most straightforward extension of a single-channel synthesis method to multiple channels is independent channel synthesis, where the synthesis method is applied to each of the channels independently. However, independent channel synthesis in an RGB color space results in undesired color shifts. This is illustrated in figure 5(b). This is because the channels of natural images are not independent. In fact, they are highly correlated [Ruderman et al., 1998; Reinhard et al., 2001]. Instead, we use independent channel synthesis in a decorrelated color space, obtained using principal component analysis (PCA), a fairly established technique in texture synthesis [Ruderman et al., 1998; Reinhard et al., 2001; Heeger and Bergen, 1995; Qin and Yang, 2002]. This is illustrated in



(a) (b) (c)

Figure 6: Arbitrary intensity distributions. (a) A photograph of a texture with a non-Gaussian intensity distribution. (b) A synthesis result without histogram matching. (c) A synthesis result with histogram matching. Each example shows the texture and the RGB histogram.

figure 5(c). Note that the undesired color shifts have disappeared. This technique can easily be applied without compromising the compactness and random accessibility of the procedural texture, since it boils down to a simple color space transformation represented by a 3×3 transformation matrix.

There seems to be no consensus about the use of independent channel synthesis in a decorrelated color space: Heeger and Bergen [1995] report good results for stochastic textures, while Kopf et al. report bad results for structured textures [Kopf et al., 2007, figure 2]. We believe this can be explained as follows. Decorrelation minimizes the correlation between the channels. However, this does not necessarily imply that the channels are also statistically independent. For stochastic textures, the decorrelated channels are often statistically independent, and if they are not, decorrelation minimizes the visible artifacts, while for structured textures, the decorrelated channels are usually not statistically independent, because of the structure. We believe this is why independent channel synthesis in a decorrelated color space works much better for stochastic textures than for structured textures.

5.2. Arbitrary Intensity Distributions

Multiresolution noise is limited to textures with a Gaussian histogram, because it always has a Gaussian intensity distribution. We address this limitation using histogram matching, a popular technique in texture synthesis [Heeger and Bergen, 1995; Dischler et al., 1998; Kopf et al., 2007]. Histogram matching is a technique to coerce the histogram of an image into a desired shape. Before we apply our method for multiresolution noise by example, we impose a Gaussian histogram on the exemplar using histogram matching, since the resulting noise will also have a Gaussian intensity distribution. After evaluating the resulting noise, we impose the original intensity distribution of the exemplar on the noise, again using histogram matching. This improves the synthesis results for textures that do not already have a Gaussian intensity distribution. This is illustrated in figure 6. It is worthwhile to note that applying histogram matching can affect the power spectrum. Therefore, some methods try to match both statistics jointly or iteratively. However, Portilla et al. [1996] showed that matching the histogram after matching the power spectrum works well in practice.

The challenge of using histogram matching in the context of procedural textures is doing so without compromising the random accessibility and compactness of the procedural texture. In order to perform histogram matching, we need to know all involved histograms. However, computing the histogram of the multiresolution noise in the conventional way would compromise random accessibility. Therefore, we approximate this histogram using the expected intensity distribution of multiresolution noise. This is a Gaussian distribution with zero mean and variance σ_M^2,

$$\sigma_M^2 = \sigma_N^2 \sum_i w_i, \tag{14}$$

where $\sigma_N^2 \approx 0.265$ is the variance of wavelet noise, and w_i are the multiresolution noise weights [Cook and DeRose, 2005]. During the analysis phase we compute a single cumulative histogram that transforms the Gaussian intensity distribution of the noise, obtained using this approximation, into the histogram of the exemplar. In order to ensure compactness, we fit this discrete histogram with a more compact parametrized representation. We subsample the histogram using a small number of samples, typically 10, and store only these samples. During the analysis phase, we evaluate the cumulative histogram using a monotone piecewise cubic Hermite spline [Fritsch and Carlson, 1980]. This technique thus boils down to applying a single spline lookup to the values obtained from equation 1. Note that the resulting histograms are also shown in figure 6(a).

5.3. Texture Synthesis By Analysis

We summarize our method as a texture-synthesis-by-analysis method. The key component of the analysis phase is our method for multiresolution noise by example, and the synthesis phase corresponds to evaluating the procedural texture.

In the analysis phase, the parameters for a procedural texture are computed. First, the channels of the exemplar are decorrelated (subsection 5.1). This results in a 3×3 transformation matrix. Then, for each of the channels, the cumulative histogram is computed, and a Gaussian histogram is imposed (subsection 5.2). This results in a 10-sample subsampled cumulative histogram per channel. Finally, for each of the channels, the weights of a multiresolution noise are computed (section 4). This results in 9 weights per channel for a resolution of 512×512. The total number of parameters is thus 66.

In the synthesis phase, the procedural texture is evaluated. First, for each of the channels, the multiresolution noise is evaluated (section 4). This corresponds to evaluating equation 3. Then, for each of the channels, the original histogram is imposed (subsection 5.2). This corresponds to a spline lookup. Finally, the channels are recorrelated. This corresponds to a matrix multiplication. Evaluating the procedural texture is fast, and can therefore be done for example per pixel in a GPU shader. The performance of the GPU shader roughly equals the

performance of the Wavelet noise GPU implementation scaled by the number of noise bands.

Although we summarize our method as a texture-synthesis-by-analysis method, the synthesized texture is a true procedural texture, with all advantages mentioned in the introduction.

6. Results and Discussion

Our method for texture synthesis by example for isotropic stochastic procedural textures is illustrated in figure 7 and figure 8. We chose not to include a comparison with state-of-the-art methods for texture synthesis by example, because that would only be of limited use. On the one hand, these methods are better than our method in terms of synthesis quality and texture classes. On the other hand, these methods produce image textures, while our method produces procedural textures, which have several advantages over image textures:

- Our procedural textures are compact. For example, a 512×512 image texture requires 768 kB of storage, while the corresponding procedural texture only requires 66 parameters of storage.
- Our procedural textures are not limited in size and resolution, and have an infinite extent. These advantages are inherited from wavelet noise [Cook and DeRose, 2005].
- Our procedural textures allow high-quality anti-aliasing using frequency clamping [Norton et al., 1982; Cook and DeRose, 2005], even when histogram matching is used [Hart et al., 1999; Lagae et al., 2009].
- Our procedural textures support frequency-based texture editing, by manipulating the weights. This is illustrated in figure 10(a,b,c).
- Our procedural textures support texture morphing, by interpolating the parameters. This is illustrated in figure 12.
- Our procedural textures support solid texture extrapolation, by replacing 2D wavelet noise by 3D wavelet noise. This is illustrated in figure 10(a,b,d) and figure 11.
- Our procedural textures can easily be integrated into production rendering software and can easily be implemented as a GPU shader[2].

Therefore, our method can be a viable alternative to state-of-the-art methods for texture synthesis by example.

For isotropic stochastic textures, our method can be seen as a procedural variant of the method of Heeger and Bergen [1995], a milestone in texture synthesis by example. Both methods achieve a similar synthesis quality[3] and use similar techniques, although in a different way. Especially note the connection between multiresolution wavelet noise and the Laplacian pyramid of white noise. The most important difference is that the textures produced by our method are procedural, they are randomly accessible and can be evaluated on-the-fly, for example in a GPU shader. We believe this is a significant advantage over the method of Heeger and Bergen, which is not procedural, and must generate the entire texture at once, in a pre-allocated memory buffer, prior to rendering. Note that although a GPU implementation of the method of Heeger and Bergen might be possible, by exploiting recent developments on noise generation on the GPU [Tzeng and Wei, 2008], the iterative pyramid construction and collapse seems problematic with respect to randomly accessibility and generation on-the-fly. The method of Heeger and Bergen [1995] however can handle a slightly larger range of textures, including anisotropic stochastic textures. More recent noise functions that support anisotropy, such as Gabor noise [Lagae et al., 2009], might enable an extension of our method to anisotropic textures.

Our method is also similar to the method of Dischler and Ghazanfarpour [1997], which automatically generates procedural geometric textures. Both methods match the power spectrum and the histogram, but again using a different methodology. For example, their spectral analysis is based on an analysis of 1D profiles and the classification of frequencies in categories, is iterative, and is subject to convergence problems, and their histogram matching is very coarse. The most important difference, however, is that the method of Dischler and Ghazanfarpour is geared towards geometric textures. They do present an example involving color textures [Dischler and Ghazanfarpour, 1997, plate 3, lower part], but only demonstrate 1D color variation. In contrast, our method can model more complex color variation.

Our method is designed for isotropic stochastic textures, and obviously cannot successfully synthesize textures that do not belong to this texture class. This limitation is inherited from wavelet noise. This is illustrated in figure 9, but to a certain degree also in figure 8(b,f,i).

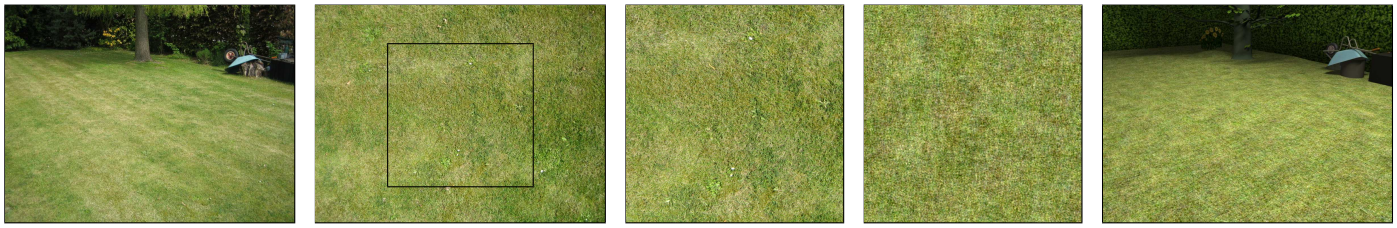
Interestingly, even unsuccessfully synthesized textures are often still usable, for example at a low level of detail, or in a pre-attentive context. We believe this is because our method always synthesizes a texture with first and second order image statistics similar to those of the exemplar, and that textures with similar image statistics are difficult to discriminate [Julesz, 1962; Malik and Perona, 1990; Heeger and Bergen, 1995]. This is illustrated in figure 13. This might be exploitable in the context of level of detail.

One application we have in mind for our method for texture synthesis by example for isotropic stochastic procedural textures is bulk modeling of textures in applications for rapid modeling such as *Google SketchUp* [Google]. A typical use case is an artist who is asked produce a textured model of a building in a short time. The artist could use our method to rapidly and fully automatically obtain an initial approximation for all textures in the scene, and concentrate manual modeling on important textures that will appear in close-up.

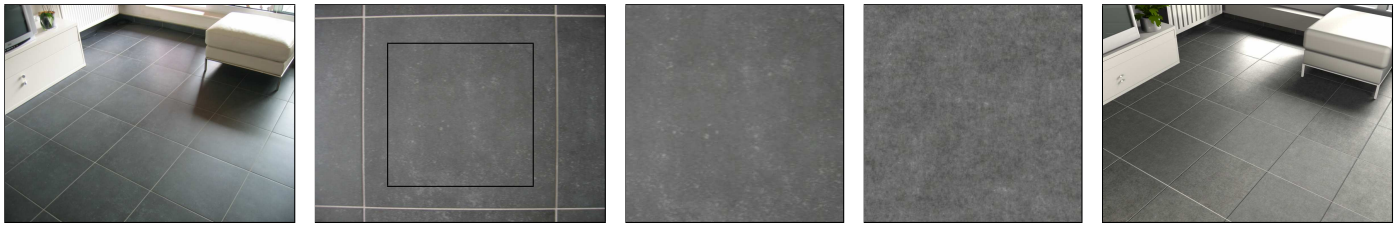
We have illustrated our method for multiresolution noise by example using texture synthesis by example for isotropic stochastic procedural textures. However, our method can also be used in any context where weights are manipulated, for ex-

[2] See supplemental material for examples.

[3] See supplemental material for a comparison.



(a) Grass.



(b) Ceramic tile. *

Figure 7: Examples of our method for texture synthesis by example for isotropic stochastic procedural textures. Each example shows a photograph of a real-world scene, a photograph of a texture in the scene, a cropped version of the photograph of the texture, a procedural texture automatically generated from the cropped version of the photograph using our method, and a rendering of a virtual scene textured using several of these procedural textures. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.

Figure 13: Preservation of image statistics. The three dresses on the left use the image texture of figure 9(a), while the three dresses on the right use the procedural texture of figure 9(a). Although the textures appear dissimilar from close by, they appear more similar from further away, because our method preserves image statistics.

ample in terrain modeling [Ebert et al., 2002].

7. Conclusion

Although procedural textures have several advantages over image textures, there is a huge discrepancy between the current state-of-the-art in texture synthesis by example for image textures and for procedural textures. We believe we have taken a step to address this discrepancy, by introducing a method for procedural multiresolution noise by example, and by illustrating this method by using it as a key component in a method for texture synthesis by example for isotropic stochastic procedural textures. Although a lot of work remains to be done, the results are encouraging. We believe that texture synthesis by example for procedural textures is a very promising research direction. Recent developments in procedural texturing are encouraging: anisotropic noise [Goldberg et al., 2008] and Gabor noise [Lagae et al., 2009] might enable an extension of our method to anisotropic textures, and lift the limitation to isotropic textures inherited from wavelet noise.

Acknowledgments

We would like to thank the anonymous reviewers, and George Drettakis, David J. Heeger, Sylvain Lefebvre, Tom Mertens, Eero P. Simoncelli and Celine Vens. Ares Lagae is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO), and acknowledges K.U.Leuven CREA funding (CREA/08/017). Toon Lenaerts acknowledges the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

Allegorithmic, . Mapzone. `http://www.mapzoneeditor.com/`.

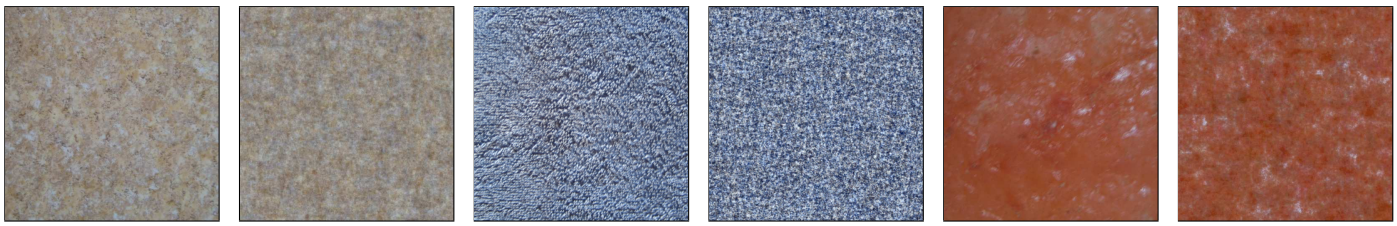
Bonet, J.S.D., 1997. Multiresolution sampling procedure for analysis and synthesis of texture images, in: Proceedings of ACM SIGGRAPH 1997, pp. 361–368.

Bourque, E., Dudek, G., 2004. Procedural texture matching and transformation. Computer Graphics Forum 23, 461–468.

Bracewell, R.N., 1999. The Fourier Transform and its Applications. McGraw-Hill. 3rd edition.

Cook, R.L., DeRose, T., 2005. Wavelet noise. ACM Transactions on Graphics 24, 803–811.

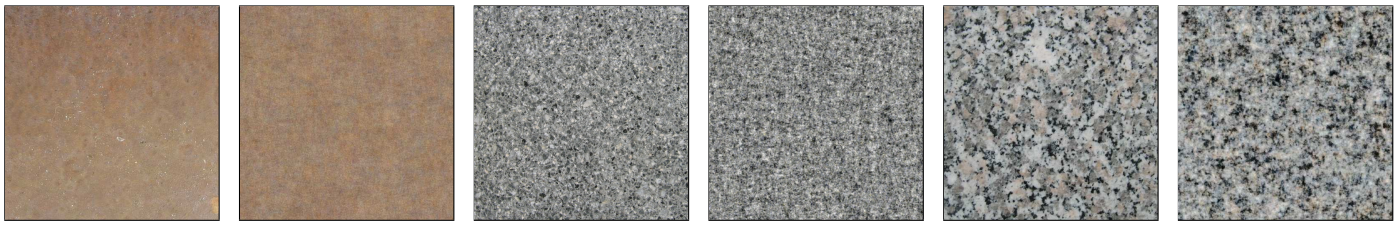
Dischler, J.M., Ghazanfarpour, D., 1997. A procedural description of geometric textures by spectral and spatial analysis of profiles. Computer Graphics Forum 16, 1997.



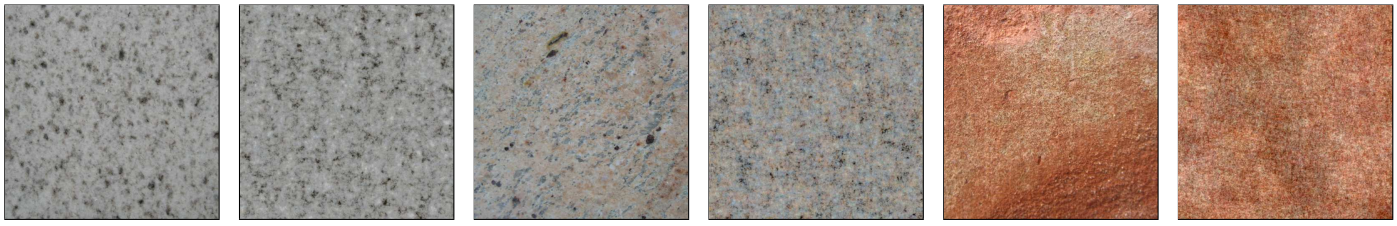
(a) Vinyl flooring. (b) Blue towel. * (c) Rock salt lamp. *

(d) Bluestone doorstep. * (e) Ceramic shower tile. (f) Blanket. *

(g) Octagonal table. (h) Living room tile. (i) Jeans vest. *



(j) Rust. (k) Clinker. * (l) Granite (Rose Sarde). *



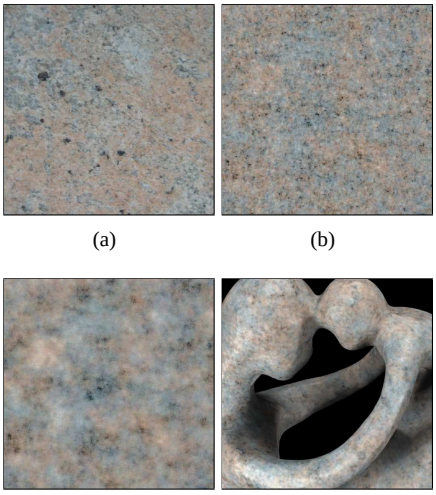
(m) Granite (Bethel White). * (n) Granite (Shivakasi Yellow). * (o) Brick wall.

(p) Stone tile. * (q) Marble (Bianco Royal). * (r) Human skin.

Figure 8: More examples of our method for texture synthesis by example for isotropic stochastic procedural textures. Each example shows a photograph of a texture and a procedural texture automatically generated from the photograph using our method. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.

(a) Dress. * (b) Wood. * (c) Hedge. *

Figure 9: Unsuccessful examples of our method for texture synthesis by example for isotropic stochastic procedural textures. Each example shows a photograph of a texture and a procedural texture automatically generated from the photograph using our method. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.



(a) (b)

(c) (d)

Figure 10: Frequency-based texture editing and solid texture extrapolation. (a) A photograph of a texture. (b) A procedural texture automatically generated from the photograph using our method. (c) The high frequencies of the procedural texture are attenuated using frequency-based texture editing. (d) The procedural texture is extrapolated to a solid procedural texture.

Figure 11: Solid texture extrapolation. A virtual scene textured using solid procedural textures obtained by applying solid texture extrapolation to procedural textures generated from photographs of textures using our method.

Figure 12: Texture morphing. The texture of figure 8(m) is morphed into the texture of figure 10 by interpolating the parameters of the corresponding procedural textures. All parameters are linearly interpolated, except the transformation matrix, which is interpolated using quaternion spherical linear interpolation.

Dischler, J.M., Ghazanfarpour, D., Freydier, R., 1998. Anisotropic solid texture synthesis using orthogonal 2d views. Computer Graphics Forum 17.

Dong, Y., Lefebvre, S., Tong, X., Drettakis, G., 2008. Lazy solid texture synthesis. Computer Graphics Forum 27, 1165–1174.

Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S., 2002. Texturing and Modeling: A Procedural Approach. Morgan Kaufmann Publishers, Inc.. 3rd edition.

Efros, A.A., Freeman, W.T., 2001. Image quilting for texture synthesis and transfer, in: Proceedings of ACM SIGGRAPH 2001, pp. 341–346.

Efros, A.A., Leung, T.K., 1999. Texture synthesis by non-parametric sampling, in: International Conference on Computer Vision, pp. 1033–1038.

Eringis, M., 2006. A completely procedural approach to materials. ACM SIGGRAPH 2006 Sketches.

Fritsch, F.N., Carlson, R.E., 1980. Monotone piecewise cubic interpolation. SIAM Journal on Numerical Analysis 17, 238–246.

Ghazanfarpour, D., Dischler, J.M., 1995. Spectral analysis for automatic 3-d texture generation. Computers and Graphics 19, 413–422.

Ghazanfarpour, D., Dischler, J.M., 1996. Generation of 3d texture using multiple 2d models analysis. Computer Graphics Forum 15, 311–323.

Goldberg, A., Zwicker, M., Durand, F., 2008. Anisotropic noise. ACM Transactions on Graphics 27, 54:1–54:8.

Google, . Sketchup. `http://sketchup.google.com/`.

Han, C., Risser, E., Ramamoorthi, R., Grinspun, E., 2008. Multiscale texture synthesis. ACM Transactions on Graphics 27, 51:1–51:8.

Hart, J.C., Carr, N., Kameya, M., 1999. Antialiased parameterized solid texturing simplified for consumer-level hardware implementation, in: Proc. ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, pp. 45–53.

Heeger, D.J., Bergen, J.R., 1995. Pyramid-based texture analysis/synthesis, in: Proceedings of ACM SIGGRAPH 1995, pp. 229–238.

Julesz, B., 1962. Visual pattern discrimination. IEEE Transactions on Information Theory 8, 84–92.

Kopf, J., Fu, C.W., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.T., 2007. Solid texture synthesis from 2D exemplars. ACM Transactions on Graphics 26, 2.

Kwatra, V., Essa, I., Bobick, A., Kwatra, N., 2005. Texture optimization for example-based synthesis. ACM Transactions on Graphics 24, 795–802.

Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A., 2003. Graphcut textures: image and video synthesis using graph cuts. ACM Transactions on Graphics 22, 277–286.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D.S., Lewis, J.P., Perlin, K., Zwicker, M., 2010. State of the art in procedural noise functions, in: EG 2010 - State of the Art Reports.

Lagae, A., Lefebvre, S., Drettakis, G., Dutré, P., 2009. Procedural noise using sparse Gabor convolution. ACM Transactions on Graphics 28, 54:1–54:10.

Lefebvre, L., Poulin, P., 2000. Analysis and synthesis of structural textures, in: Graphics Interface 2000: Proceedings, pp. 77–86.

Lefebvre, S., Hoppe, H., 2005. Parallel controllable texture synthesis. ACM Transactions on Graphics 24, 777–786.

Lewis, J.P., 1989. Algorithms for solid noise synthesis, in: Computer Graphics (Proceedings of ACM SIGGRAPH 89), pp. 263–270.

Malik, J., Perona, P., 1990. Preattentive texture discrimination with early vision mechanisms. Journal of the Optical Society of America A 7, 923.

Norton, A., Rockwood, A.P., Skolmoski, P.T., 1982. Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space, in: Computer Graphics (Proceedings of ACM SIGGRAPH 84), pp. 1–8.

Perlin, K., 1985. An image synthesizer, in: Computer Graphics (Proceedings of ACM SIGGRAPH 85), pp. 287–296.

Perlin, K., 2002. Improving noise. ACM Transactions on Graphics , 681–682.

Pixar, 2005. The renderman interface: Verstion 3.2.1. `http://renderman.pixar.com/products/rispec/`.

Portilla, J., Navarro, R., Nestares, O., Tabernero, A., 1996. Texture synthesis-by-analysis method based on a multiscale early-vision model. Optical Engineering 35, 2403–2417.

Portilla, J., Simoncelli, E.P., 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. International Journal of Computer Vision 40, 49–70.

Qin, X., Yang, Y.H., 2002. Estimating parameters for procedural texturing by genetic algorithms. Graphical Models 64, 19–39.

Reinhard, E., Ashikhmin, M., Gooch, B., Shirley, P., 2001. Color transfer between images. IEEE Computer Graphics Applications 21, 34–41.

Ruderman, D.L., Cronin, T.W., Chiao, C.C., 1998. Statistics of cone responses to natural images: implications for visual coding. Journal of the Optical Society of America A 15, 2036–2045.

Tzeng, S., Wei, L.Y., 2008. Parallel white noise generation on a GPU via cryptographic hash, in: Proc. 2008 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 79–87.

Wei, L.Y., Han, J., Zhou, K., Bao, H., Guo, B., Shum, H.Y., 2008. Inverse texture synthesis. ACM Transactions on Graphics 27, 52:1–52:10.

Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G., 2009. State of the art in example-based texture synthesis, in: Eurographics 2009 State of the Art Reports, pp. 93–117.

Wei, L.Y., Levoy, M., 2000. Fast texture synthesis using tree-structured vector quantization, in: Proceedings of ACM SIGGRAPH 2000, pp. 479–488.