Isotropic Stochastic Procedural Textures by Example

Ares Lagae
Peter Vangorp
Toon Lenaerts
Philip Dutré

Report CW 546, May 2009

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Isotropic Stochastic Procedural Textures by Example

Ares Lagae
Peter Vangorp
Toon Lenaerts
Philip Dutré

Report CW 546, May 2009

Department of Computer Science, K.U.Leuven

Abstract

Procedural textures have significant advantages over image textures. Procedural textures are compact, are resolution and size independent, often remove the need for a texture parameterization, can easily be parameterized and edited, and allow high quality anti-aliasing. However, creating procedural textures is more difficult than creating image textures. Creating procedural textures typically involves some sort of programming language or an interactive visual interface, while image textures can be created by simply taking a digital photograph. In this paper we present a method for creating procedural textures by example, designed for isotropic stochastic textures. From a single uncalibrated photograph of a texture we compute a small set of parameters that defines a procedural texture similar to the texture in the photograph. Our method allows us to replace image textures with similar procedural textures, combining the advantages of procedural textures and image textures. Our method for creating isotropic stochastic procedural textures by example therefore has the potential to dramatically improve the texturing and modeling process.

CR Subject Classification : I.3.3, I.3.7.

Report CW 546, May 2009, Department of Computer Science, Katholieke Universiteit Leuven.

Isotropic Stochastic Procedural Textures by Example

Ares Lagae Peter Vangorp Toon Lenaerts Philip Dutré

Department of Computer Science
Katholieke Universiteit Leuven[*]

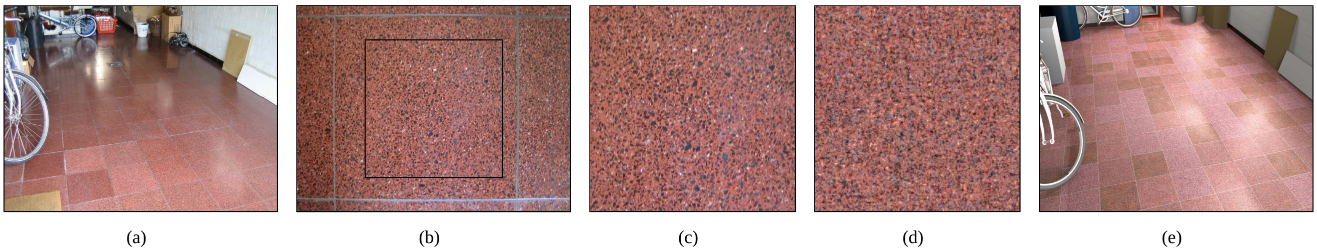


Figure 1: *Isotropic stochastic procedural textures by example. (a) A photograph of a real-world scene. (b) A photograph of a texture in the scene. (c) A cropped version of the photograph of the texture. (d) A procedural texture automatically generated from the cropped photograph. (e) A rendering of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

Abstract

Procedural textures have significant advantages over image textures. Procedural textures are compact, are resolution and size independent, often remove the need for a texture parameterization, can easily be parameterized and edited, and allow high quality anti-aliasing. However, creating procedural textures is more difficult than creating image textures. Creating procedural textures typically involves some sort of programming language or an interactive visual interface, while image textures can be created by simply taking a digital photograph. In this paper we present a method for creating procedural textures by example, designed for isotropic stochastic textures. From a single uncalibrated photograph of a texture we compute a small set of parameters that defines a procedural texture similar to the texture in the photograph. Our method allows us to replace image textures with similar procedural textures, combining the advantages of procedural textures and image textures. Our method for creating isotropic stochastic procedural textures by example therefore has the potential to dramatically improve the texturing and modeling process.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: procedural texture, solid texture, texture synthesis, wavelet noise

1 Introduction

Texturing was introduced by Catmull [1974] as a method for increasing the visual complexity of computer-generated images without adding geometric detail. Texturing quickly became a fundamental tool in computer graphics. There are two important types of textures: image textures consisting of raster data and procedural textures.

Procedural textures [Ebert et al. 2002] have significant advantages over image textures. Procedural textures are compact, are resolution and size independent, and can easily be parameterized; procedural textures are often solid textures, which remove the need for a texture parameterization; procedural textures built with band-limited noise functions such as wavelet noise [Cook and DeRose 2005] allow high quality anti-aliasing; and procedural textures leave room for an artist to tweak. Because of these advantages, procedural textures are increasingly popular in production rendering. For example, wavelet noise was recently developed at *Pixar*, and *Blue Sky*'s *CGIStudio* uses a completely procedural approach to texturing for animated feature films [Eringis 2006].

Creating procedural textures in an intuitive way is one of the biggest challenges in procedural texturing. Creating procedural textures 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 2008], accompanied by a deep understanding of procedural texturing. In contrast, image textures can be created by simply taking a photograph.

In this paper we present a perceptually motivated method for creating procedural textures by example, designed for isotropic stochastic textures. From a single uncalibrated photograph of a texture we compute a small set of parameters that defines a procedural texture similar to the texture in the photograph. Figure 1 shows an example. Our method consists of an analysis phase and a synthesis phase. During the analysis phase, the photograph of a texture is analyzed and a small set of parameters for a similar procedural texture is computed. After the analysis phase, the photograph is discarded. During the synthesis phase, the procedural texture is evaluated using the parameters computed during the analysis phase, for example as part of a rendering program or a GPU shader.

Our method allows us to replace image textures with similar procedural textures, combining the advantages of procedural textures and image textures. Our method for creating isotropic stochastic procedural textures by example therefore has the potential to dramatically improve the texturing and modeling process. For example, when an artist imports an image texture into a 3D modeling software package, the software package could automatically convert the image texture into a procedural texture. The artist would not have to worry about the size and resolution of the texture, the solid version of the procedural texture removes the need for a texture parameterization, the software package can produce high quality anti-aliased renderings, and the artist can tweak the procedural texture.

[*]e-mail: {ares,peter,toon,phil}@cs.kuleuven.be

Overview

Section 2 discusses related work, and section 3 introduces multiresolution wavelet noise. We present our method for creating isotropic stochastic procedural textures by example in sections 4, 5 and 6. Our method consist of three building blocks:

1. a spectral method to match the weights in a multiresolution wavelet noise equation to a photograph (section 4);
2. a method to match the intensity distribution of the photograph to the procedural texture based on histogram matching (section 5); and
3. a method to handle color based on principal component analysis (section 6).

We summarize our method in section 7. In section 8 we present results, in section 9 we compare with related work, and in section 10 we conclude.

2 Related Work

Our work is related to procedural texturing, procedural textures by example, texture synthesis by example, and solid texture modeling.

2.1 Procedural Texturing

Procedural texturing and solid texturing are closely related. Solid texturing was introduced by Perlin [1985] and Peachy [1985]. At the same time, Perlin introduced his famous noise function, the basis for a large variety of procedural textures. Although solid texturing and procedural texturing are closely related, procedural texturing was already known before the introduction of solid texturing [Fournier et al. 1982; Gardner 1984]. Perlin's noise function quickly became the most popular noise function, and probably still is today.
One of the desirable properties of a noise function identified by Perlin is that it is band limited [Perlin 1985]. However, although Perlin's noise function is simple, efficient and elegant, it is not band limited, and therefore prone to problems with aliasing and detail loss [Cook and DeRose 2005]. Lewis [1989] proposed two noise functions similar to Perlin's noise function with better control over the noise power spectrum, but they did not gain widespread use. Perlin [2002] revised his noise function but did not address the fact that it was not band limited. Inspired by the work of Lewis, Cook and DeRose [2005] introduced a noise function similar to Perlin's noise function but with a better band limited behavior. Goldberg et al. [2008] recently presented a technique that provides high quality anisotropic filtering for noise textures. Cook and DeRose and Goldberg et al. showed that procedural textures can be antialiased significantly better than raster data without sacrificing detail. The method for creating procedural textures by example we present in this work builds on the work of Cook and DeRose.
Next to noise functions similar to Perlin's noise function, several other procedural texture basis functions have been proposed, such as the cellular texture basis function of Worley [1996] and the procedural object distribution function of Lagae and Dutré [2005], but these are geared towards very specific classes of procedural textures.
For an extensive overview of procedural texturing and modeling we refer to Ebert et al. [2002].

2.2 Procedural Textures by Example

Procedural textures by example is a collection of methods that recovers parameters for a procedural texture from an example texture. Ghazanfarpour and Dischler [1995; 1996] introduced a spectral method for automatic solid procedural texture generation from a single or multiple 2D example textures. However, their method is designed for textures with only a few dominant frequencies. Lefebvre and Poulin [2000] presented a system to extract values for parameters of structural textures from photographs. Nevertheless, their method is limited to rectangular tilings and wood. Qin and Yang [2002] introduced a genetic-based multiresolution parameter estimation approach to recover the parameter values for a given procedural texture. However, they did not demonstrate results on real-world textures. Bourque and Dudek [2004] presented a very general system that performs a two-phase search over a library of procedural shaders. Their system can handle several texture classes, but determining the parameters of a procedural texture using a local search is less efficient and less accurate than a direct computation.
Procedural textures by example is an important unsolved problem in texturing.

2.3 Texture Synthesis by Example

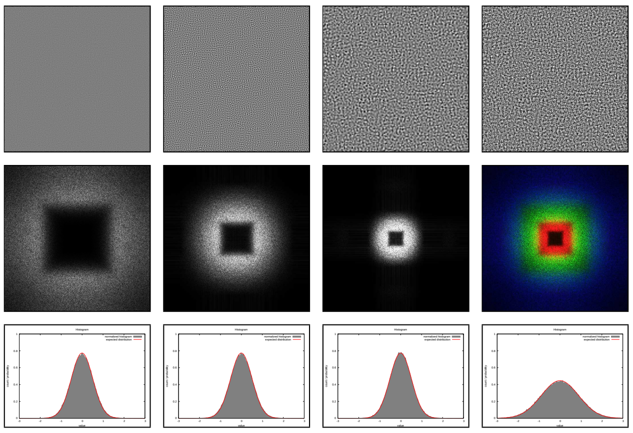
Texture synthesis by example is a collection of methods that synthesizes a new texture from an example texture.
Texture synthesis by example includes parametric methods [Heeger and Bergen 1995; Portilla and Simoncelli 2000], non-parametric methods [Bonet 1997], including pixel-based methods [Efros and Leung 1999; Wei and Levoy 2000] and patch-based methods [Efros and Freeman 2001; Kwatra et al. 2003], and optimization-based methods [Kwatra et al. 2005].
Texture synthesis by example is usually geared towards 2D image textures, but some methods also consider solid textures. Heeger and Bergen [1995] proposed a parametric method for texture synthesis by example that can be used to generate solid textures, and Kopf et al. [2007] introduced an optimization-based texture synthesis method for solid texture synthesis from 2D examples.
The major difference between methods for texture synthesis by example and the method presented in this paper is that our method constructs a procedural texture while methods for texture synthesis synthesize raster data.
Our method bears some similarity to parametric methods for texture synthesis, such as the methods by Heeger and Bergen [1995] and Portilla and Simoncelli [2000]. However, in contrast to our method, these methods are not randomly accessible and can therefore not be formulated as procedural textures.

2.4 Solid Texture Modeling

Solid texture modeling is a collection of several methods for generating solid textures.
Dischler and Ghazanfarpour presented a method based on hybrid analysis for automatic solid texture synthesis [Dischler et al. 1998], and an interactive system for image based modeling of macrostructured textures [Dischler and Ghazanfarpour 1999]. Jagnow et al. [2004] use stereological methods to synthesize 3D solid textures of aggregate materials of particles from 2D images of existing materials. However, neither method produces procedural textures.
Several previously mentioned methods for procedural textures by example [Ghazanfarpour and Dischler 1995; Ghazanfarpour and Dischler 1996] and texture synthesis by example [Heeger and Bergen 1995; Kopf et al. 2007] can be used for solid texture modeling.
For an extensive overview of solid texturing, including several approaches to solid texture modeling, we refer to Dischler and Ghazanfarpour [2001].

3 Multiresolution Wavelet Noise

Procedural textures are constructed starting from noise functions using a process similar to function composition. Therefore, pro-



(a) $N\left(2^{i+2}x\right)$ (b) $N\left(2^{i+1}x\right)$ (c) $N\left(2^{i+0}x\right)$ (d) $M\left(x\right)$

Figure 2: *Multiresolution wavelet noise. (a, b, c) Three consecutive wavelet noise bands. (d) Multiresolution wavelet noise composed from the three noise bands using unit weights. Each figure shows the noise, the power spectrum of the noise, and the histogram of the noise (gray) with the expected distribution (red).*

cedural textures can be arbitrarily complex. Accordingly, we must select a specific class of procedural textures. We chose multiresolution noise. This class of procedural textures roughly corresponds to the class of isotropic stochastic textures. This is one of the simplest but also one of the most important texture classes. Isotropic stochastic textures occur frequently in the real world, and at the appropriate scale, a lot of real world textures can accurately be approximated by isotropic stochastic textures.

Multiresolution noise $M\left(x\right)$ is constructed by summing scaled and weighted versions of a noise band $N\left(x\right)$

$$M\left(x\right) = \sum_i w_i N\left(2^i\left(x + o_i\right)\right). \tag{1}$$

A random offset o_i is added to each noise band to decorrelate the different noise bands. The weights w_i determine the look of the procedural texture $M\left(x\right)$.

Mutiresolution noise works with any type of noise. A desirable property of the noise function is that it has a narrow bandpass limit in frequency. We use wavelet noise [Cook and DeRose 2005] instead of Perlin's noise function [Perlin 2002]. Wavelet noise looks very similar to Perlin's noise function but in contrast with Perlin's noise function, wavelet noise is band-limited and has a known Gaussian intensity distribution. Note that multiresolution noise based on a band-limited noise function can easily be antialiased using clamping [Norton et al. 1982].

A wavelet noise band $N\left(x\right)$ is a quadratic B-spline surface. The coefficients for the surface are constructed by creating an image filled with random Gaussian noise, and removing the part that can be represented at half the resolution. This is done by subtracting from the image a downsampled and upsampled version of the image, using appropriate filters.
The expected variance σ_N^2 of a wavelet noise band is approximately 0.265. The expected variance of multiresolution wavelet noise $M\left(x\right)$ is

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

For more details we refer to Cook and DeRose [2005].

Figure 2 shows three wavelet noise bands and multiresolution

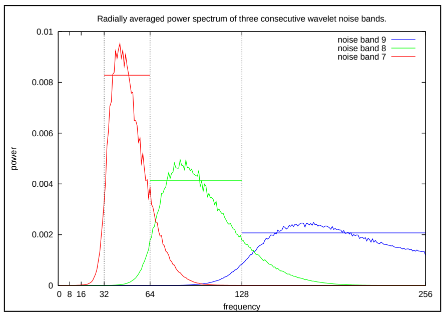


Figure 3: *The radially averaged power spectrum of three consecutive wavelet noise bands.*

wavelet noise composed from these noise bands using unit weights. The histograms of the noise match the corresponding expected Gaussian intensity distributions well. The color coded power spectrum of the multiresolution wavelet noise shows that the wavelet noise bands are band limited and have a limited overlap in frequency space. This is also illustrated in figure 3 which shows the radially averaged power spectrum of three consecutive wavelet noise bands.

4 Matching The Power Spectrum

We construct a procedural texture from a photograph of a texture starting from the multiresolution noise equation (equation 1) using wavelet noise bands. The goal is to compute the weights w_i of the multiresolution wavelet noise $M\left(x\right)$ such that the multiresolution noise matches the texture in the photograph. In this analysis we assume that the photograph consists of a luminance channel only.

4.1 Theoretical Analysis

We apply the Fourier transform $\mathcal{F}$ to both sides of the multiresolution noise equation. Without loss of generality we can ignore the offsets o_i. Since the Fourier transform is a linear operator it can be distributed inside the summation.

$$\mathcal{F}\left(M\left(x\right)\right) = \sum_i w_i \mathcal{F}\left(N\left(2^i x\right)\right). \tag{3}$$

In order to compute the weight w_b we define an operator S_b that selects the frequency band b in the Fourier domain

$$S_b\left(\mathcal{F}\left(N\left(2^i x\right)\right)\right) = \begin{cases} 0 & b \neq i \\ \mathcal{F}\left(N\left(2^b x\right)\right) & b = i \end{cases} \tag{4}$$

The operator S_b assumes that the noise band is perfectly band-limited. We apply S_b to both sides. Since S_b is a linear operator it can be distributed inside the summation. Using the definition of S_b we arrive at

$$S_b\left(\mathcal{F}\left(M\left(x\right)\right)\right) = w_b \mathcal{F}\left(N\left(2^b x\right)\right). \tag{5}$$

At this point it might seem as if the weight w_b can be determined by dividing $S_b\left(\mathcal{F}\left(M\left(x\right)\right)\right)$ by $\mathcal{F}\left(N\left(2^b x\right)\right)$. However, this is not the case, since only the statistical properties of $N\left(2^b x\right)$ are known. This approach would work if $M\left(x\right)$ was a multiresolution noise generated with known noise bands instead of an arbitrary photograph.
Although we cannot assume that the photograph $M\left(x\right)$ was constructed with a known noise band $N\left(2^b x\right)$, we can exploit the statistical properties of a wavelet noise band to approximate the weight

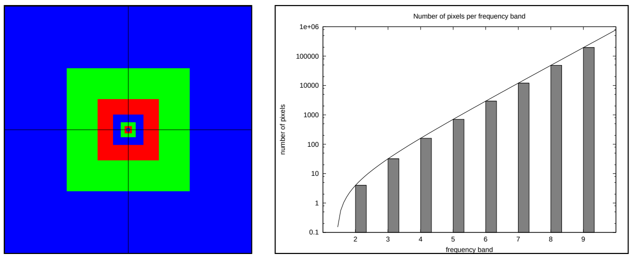


Figure 4: *Wavelet noise frequency bands. (a) The color-coded support of each wavelet noise frequency band in the frequency domain. (b) The number of pixels in each band. The resolution is* 512×512.

w_b. We proceed by taking the power of both sides

$$\left|S_b\left(\mathcal{F}\left(M\left(x\right)\right)\right)\right|^2 = w_b^2\left|\mathcal{F}\left(N\left(2^b x\right)\right)\right|^2. \tag{6}$$

To determine the power in the noise band $N\left(2^b x\right)$ we use Parseval's theorem, which states that the power is the same in the spatial domain and in the frequency domain

$$\left|\mathcal{F}\left(N\left(2^b x\right)\right)\right|^2 = \left|N\left(2^b x\right)\right|^2. \tag{7}$$

Since a wavelet noise band is a random Gaussian variable with zero mean and known variance σ_N^2, we can use the computational formula for the variance to determine its expected average power

$$\left\langle\left|N\left(x\right)\right|^2\right\rangle = \sigma_N^2 \approx 0.265. \tag{8}$$

Note that the factor 2^b can be dropped, since the power in a wavelet noise band is independent of its scale. The weight w_b can now be approximated by

$$\left|w_b\right| \approx \sqrt{\frac{\left|S_b\left(\mathcal{F}\left(M\left(x\right)\right)\right)\right|^2}{\sigma_N^2}}. \tag{9}$$

In this analysis we make two important assumptions. The first is that the multiresolution wavelet noise is a good model for the photograph. This is the case for isotropic stochastic textures. The second is that expected average power in a wavelet noise band is a good estimate for the actual power in the wavelet noise band. This assumption is necessary because we must make abstraction of the actual noise bands.

Figure 3 shows the radially averaged power spectrum of three consecutive wavelet noise bands. This figure also illustrates our model for the expected power in each noise band. The power in each noise band, which corresponds to the area below the corresponding curve, is equal to σ_N^2, the area below the corresponding square wave.

4.2 Implementation

We compute the weights w_b for a photograph M with a power of two resolution of $N \times N$ according to equation 9. We compute the power in each frequency band by interpreting the photograph M as an $N \times N$ matrix, computing the Fast Fourier Transform, iterating over all elements, sorting the elements into their corresponding frequency band, and accumulating the power in each frequency band.

We number the wavelet noise bands according to their frequency band. The wavelet noise band with index b corresponds to a frequency band with discrete frequencies with an absolute horizontal and vertical frequency component smaller than 2^{b-1} and greater than or equal to 2^{b-2}, and is available if the resolution of the photograph is at least $2^b \times 2^b$. The band index b for an element at position (i, j) in the Fast Fourier Transform of the photograph M is given by

$$b = \max\left(\log_2\left|f_i\right|, \log_2\left|f_j\right|\right) + 2, \tag{10}$$

where f_i and f_j are the horizontal and vertical frequency corresponding to element (i, j). For a resolution of 512×512, the noise bands in figure 2 have indices 9, 8 and 7. A photograph M with a power of two resolution of $N \times N$ contains information for $\log_2 N - 1$ wavelet noise bands. For a resolution of 512×512, information for 8 frequency bands is available.

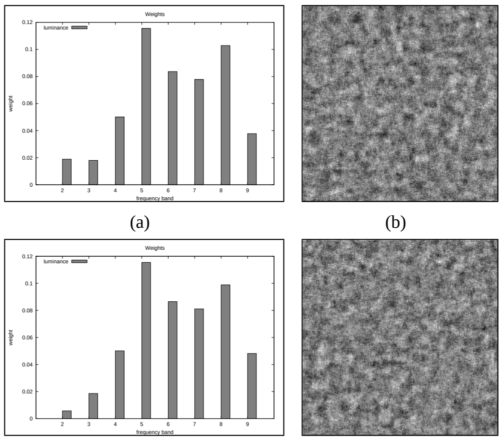


Figure 5: *Matching power spectrum. (a) A set of randomly generated weights. (b) An image generated using the weights in (a). (c) The weights recovered from the image in (b). (d) An image generated using the weights in (c).*

We also compute the average luminance during the analysis phase. Note that the average luminance corresponds to the DC component of the Fast Fourier Transform of the photograph M. During the synthesis phase, an approximation of the photograph M can be constructed by evaluating the multiresolution noise equation using wavelet noise bands and the computed weights, and adding back the average luminance.

Figure 4 shows the support of the frequency bands and the number of pixels in each frequency band. The number of pixels for lower frequency bands becomes increasingly smaller. This implies that the expected average power becomes a less reliable estimate for the actual power in the frequency band. Therefore, the weight of lower frequency bands also becomes increasingly less reliable.

Figure 5 illustrates our method for matching weights. We have generated a set of random weights, generated an image using these weights, computed the weights from the generated image using the method presented in this section, and generated a new image using the computed weights. Both the weights and the images are similar, and there is a slightly larger error on weights corresponding to lower less reliable frequency bands.

5 Matching The Intensity Distribution

Our method for matching the weights in the multiresolution noise equation to a photograph always constructs a procedural texture with a Gaussian intensity distribution. This is because the wavelet noise bands are independent Gaussian random variables (see section 3). Therefore it is reasonable to assume that our method for matching the weights will work better if the intensity distribution of the photograph is more Gaussian. Many of our photographs of stochastic textures already have more or less a Gaussian intensity distribution, but for photographs for which this is not the case, we match the intensity distribution of the generated procedural texture to the photograph using histogram matching. This ensures that our method also works with photographs with a non-Gaussian intensity distribution.

Histogram matching is a method to coerce the histogram of an image into a desired histogram shape. This is done by transforming the pixels in the image with the cumulative histogram of the image, which results in an equalized histogram, and then transforming the pixels again with the inverse cumulative desired histogram. Histogram matching is similar to the transformation method in statistics, that generates a random variable with a known probability density function by transforming a uniform random variable with the corresponding inverse cumulative probability density function. For more information about histogram matching we refer to Gonzalez and Woods [2008].

In the analysis phase, we coerce the histogram of the photograph into a Gaussian intensity distribution, with a variance that best fits the histogram of the photograph. We then apply the weight matching method to the Gaussianized version of the photograph. We store the histogram that transforms the photograph back to its original intensity distribution as an additional parameter. In the synthesis phase, evaluating the multiresolution wavelet noise results in a Gaussian intensity distribution, which we transform back to the original intensity distribution of the photograph with the stored histogram.
Image histograms are traditionally represented using 256 bins. However, 256 parameters is a lot for a procedural texture. We reduce the number of parameters using histogram fitting, which approximates the discrete histogram with a more compact parameterized representation. There are several possibilities, such as polynomials, exponentials and Gaussians. We use a monotone piecewise cubic hermite spline [Fritsch and Carlson 1980] to fit the subsampled inverse cumulative histogram. We choose the subsampled number of bins roughly the same as the number of weights. For a photograph with a resolution of 512×512 we use 10 bins, which provides good results.
Note that our method for matching the intensity distribution does not compromise the random accessibility of the procedural texture. This is because the histogram of the synthesized procedural texture is derived from the statistical properties of wavelet noise (see section 3) rather than computed directly from the intensity values.

Figure 6 illustrates the method for matching the intensity distribution of a photograph to the generated procedural texture. Figure 6(a) shows a photograph and its histogram. Figure 6(b) and figure 6(c) show the procedural texture without and with histogram matching. Note that figure 6(b) has a Gaussian histogram. The procedural texture constructed with histogram matching is more similar to the photograph than the procedural texture constructed without histogram matching. Figure 6(a) also shows the fitted histogram.

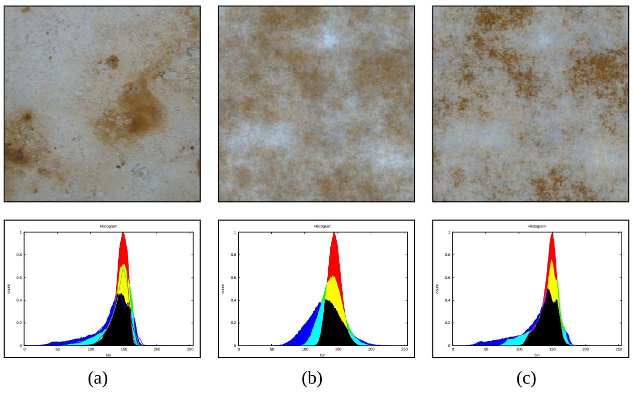


Figure 6: *Matching intensity distribution. (a) A photograph of a texture. (b,c) A procedural texture generated from the photograph (b) without histogram matching and (c) with histogram matching. Histograms are also shown. The fitted histogram is shown in (a) with lines.*

6 Matching The Color

Our methods for matching the power spectrum and matching the intensity distribution assume that the photograph consists of only a luminance channel. In this section we show how to match the color of the computed procedural texture to the color of the photograph using principal component analysis. For more information on principal component analysis, we refer to Jolliffe [2002].

6.1 RGB Color Space

The most straightforward method to match the color of the computed procedural texture to the color of the photograph is to apply the analysis of the photograph and the synthesis of the procedural texture to the red, green and blue channel independently.

Figure 7 illustrates the RGB color space method. Figure 7(a) shows a photograph and figure 7(b) shows the corresponding procedural texture with an independently generated red, green and blue channel. However, this method results in unwanted color shifts not present in the photograph, due to the random offsets o_i in the multiresolution noise equation (equation 1) that decorrelate the different noise bands. For example, in order to produce a yellow color, the red and the green channel must be correlated, producing large values simultaneously, but the random offsets prevent this from happening, producing red and green color shifts instead.
The unwanted color shifts can be eliminated by using the same random offsets o_i for the red, green and blue channel. This is illustrated in figure 7(c). However, the red, green and blue channel are now perfectly correlated. This means that this method cannot reproduce variation in color that is present in the photograph. Figure 7 shows that the RGB color space method is not able to reproduce the green-brown colors in the photograph.

6.2 Decorrelated Color Space

The red, green and blue channel in natural images are highly correlated [Ruderman et al. 1998; Reinhard et al. 2001]. This means that a pixel with a large value for the red channel is likely to have a large value for the green and the blue channel. This is the reason why treating the red, green and blue channel independently did not generate the expected result in the previous subsection. However, treating channels independently is possible in a decorrelated color space obtained by principal component analysis.

We construct a decorrelated color space for a photograph M by computing the singular value decomposition of the covariance matrix C of the photograph. This is a 3×3 matrix that measures the correlation between the random variables corresponding to the red, green and blue channel. The covariance matrix C is obtained by multiplying the photograph M with its transpose, where M is a matrix with a row for the red, green and blue channel and a column for each pixel, in which the average red, green and blue value was subtracted from each element. The singular value decomposition of the covariance matrix C is a matrix decomposition of the form

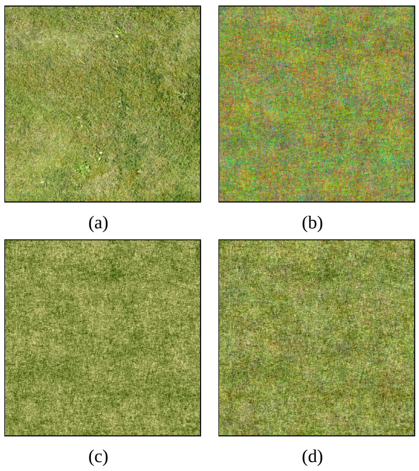


Figure 7: *Matching color. (a) A photograph of a texture. (b-d) Procedural textures created from the photograph using (b) an RGB color space, (c) an RGB color space with the same random offsets for the red, green and blue channel, and (d) a decorrelated color space.*

$C = UDU^T$, where U is a 3×3 orthogonal matrix and D is a 3×3 diagonal matrix whose elements are the singular values of C. The 3×3 transformation matrix that transforms M into a decorrelated color space is U^T. The covariance matrix of $U^T M$ is a diagonal matrix, indicating perfectly decorrelated random variables.

A similar technique was used by Heeger and Bergen [1995] in the context of texture synthesis and by Ruderman et al. [1998] to construct a decorrelated color space for natural images.

In the analysis phase, we compute the decorrelated color space, and store the transformation matrix U^T as an additional parameter. We apply the methods for matching the intensity distribution and matching the power spectrum independently to each of the decorrelated channels. During the synthesis phase, the procedural texture is evaluated, yielding a value for each of the decorrelated channels. We convert these values back to the RGB color space using the transformation matrix.

Figure 7 illustrates the decorrelated color space method. Figure 7(a) shows a photograph and figure 7(d) shows the corresponding procedural texture in which the channels were independently generated in the decorrelated color space, using different random offsets o_i for each of the decorrelated color channels. The decorrelated color space method is able to reproduce the green-brown colors in the photograph.

7 Summary

The analysis phase of our method can be summarized as follows:

1. compute the average value of the red, green and blue channel and subtract it from the corresponding channel;
2. decorrelate the red, green and blue channel (see section 6);
3. (optionally) apply histogram matching to each of the decorrelated color channels (see section 5); and
4. compute the weights for each of the decorrelated color channels (section 4).

The analysis phase is fast, robust and fully automatic. In less than a second, a 512×512 photograph is reduced to a handful of parameters: the average value of the red, green and blue channel (3 parameters), the transformation matrix (9 parameters), the weights (3×8 parameters) and the histogram (3×10 parameters). After the analysis phase, the photograph is discarded.

The synthesis phase of our method can be summarized as follows:

1. evaluate the multiresolution wavelet noise equation (equation 1) for each of the decorrelated color channels;
2. (optionally) apply histogram matching to each of the decorrelated color channels (see section 5);
3. recorrelate the red, green and blue channel (see section 6); and
4. add the average value of the red, green and blue channel back to the corresponding channel.

The synthesis phase uses the parameters generated during the analysis phase, and corresponds to a true procedural texture with all the previously mentioned advantages.
The procedural texture can efficiently be evaluated. A single evaluation of the procedural texture corresponds to a single multiresolution wavelet noise evaluation for each of the three channels, a single spline lookup for each of the three channels to match the intensity distribution, and a single matrix multiplication to recorrelate the three channels.

8 Results

We have constructed a wide variety of procedural textures using our method. Figure 1 and figure 8 show several successful examples of procedural textures created with our method. 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 photograph, and a rendering of a virtual scene textured using the procedural texture. Figure 9 and figure 10 show more examples. The photograph with context and the photograph of the texture are uncalibrated photographs taken with an inexpensive digital camera. Note that the color of the texture in the context photograph and in the photograph of the texture can be different due to automatic white balancing.

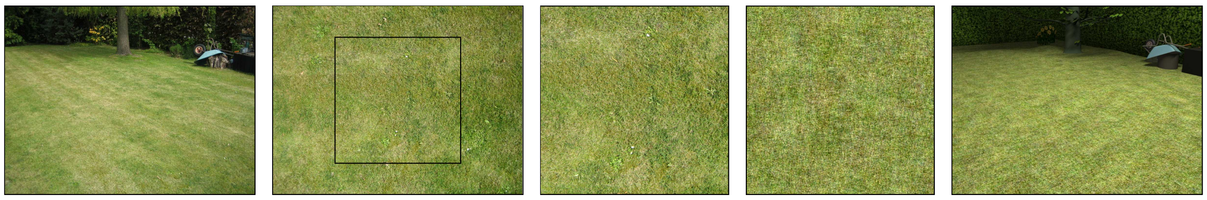
Our method constructs a procedural texture rather than an image texture. In contrast to an image texture, the corresponding procedural texture is not limited in size and resolution, the procedural texture is parameterized and can be edited, and the solid version of the procedural texture does not require a texture parameterization. This is illustrated in figure 11. Our method strictly decouples the analysis phase and the synthesis phase, discarding the photograph after the analysis phase. This allows us to integrate the synthesis phase into for example a 3D modeling software package, a renderer or a GPU shader. We have implemented our method as a Maya plugin and as a GPU shader.

Our method is designed for isotropic stochastic textures. Figure 12 shows some unsuccessful examples of procedural textures created with our method. Our method cannot reproduce marble veins and wood grain. These are typically created procedurally by combining turbulence with a sine function [Perlin 1985], and are not isotropic stochastic textures.

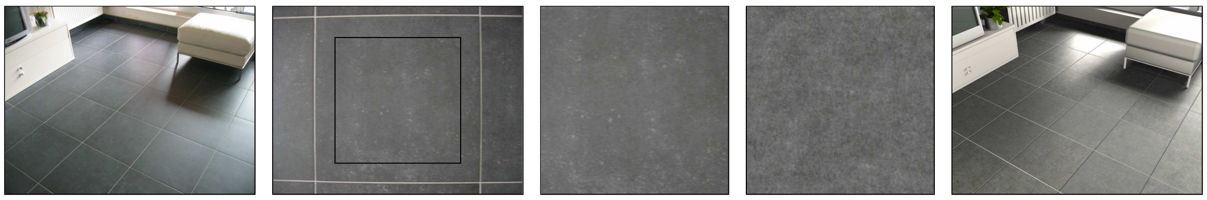
Procedural textures constructed using our method can be extrapolated to solid procedural textures simply by evaluating them using 3D wavelet noise bands. This is illustrated in figures 11 and 13.

Our method allows to morph between textures by interpolating the parameters of the generated procedural textures. Figure 15 shows an example. All parameters are linearly interpolated, except the transformation matrix, which is interpolated using quaternion spherical linear interpolation.

Our method is perceptually motivated, similar to the method for tex-



(a) Grass.



(b) Ceramic tile. *

Figure 8: *Successful examples of procedural textures created with our method. 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 photograph, and a rendering of a virtual scene textured using the procedural texture. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.*

ture synthesis by example of Heeger and Bergen [1995]. Textures with similar first and second order statistics are difficult to discriminate [Julesz 1962; Malik and Perona 1990; Heeger and Bergen 1995]. Our method for matching the power spectrum matches second order statistics, and our method for histogram matching matches the first order statistics.

Although our method fails on textures that are not isotropic stochastic, it still constructs a procedural texture with matching image statistics. These procedural textures are often still usable. For example, structure is often found in higher frequencies, which are filtered out with distance. This means that our method also works for such textures as long as they are not inspected from nearby.

This is illustrated in figure 14 with the texture of figure 12(a). The three dresses on the left use the image texture and the three dresses on the right use the procedural texture. The texture appears more similar with increasing distance. This is also illustrated in figure 8(a) with the texture of figure 12(d).

Because of this, our method might still be a valuable tool for modeling textures that are not strictly isotropic stochastic. Since our method is fast and fully automatic, our method might also be usefull for bulk modeling of textures, for example when digitizing a complete scene.

9 Comparisons

Heeger and Bergen [1995] and Portilla and Simoncelli [2000] presented parametric methods for texture synthesis by example. Our method bears some similarity to these methods. The textures synthesized with the method of Heeger and Bergen are similar to the procedural textures constructed with our method. Both methods also use a decorrelated color space and histogram matching.

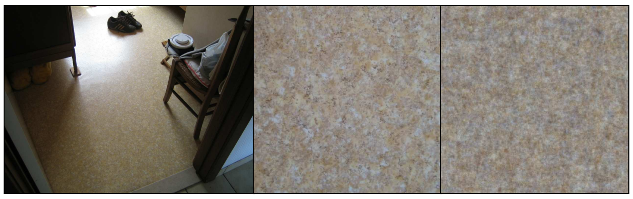
Figure 16 compares the results of our method with the method of Heeger and Bergen. The comparison is based on the publicly available steerable pyramid implementation of Simoncelli. The results of both methods are similar. The major difference however is that our method constructs a procedural texture, while the method of Heeger and Bergen cannot be formulated as a procedural texture. Heeger and Bergen synthesize a texture by iteratively matching the histograms of the levels of the Laplacian or steerable pyramid of the example texture and the synthesized texture. However, computing the histograms of the synthesized texture is a global operation. This means that the method of Heeger and Bergen does not result in a texture that is randomly accessible and cannot be formulated as a procedural texture. Our method avoids this problem by deriving the histogram of the procedural texture from the statistical properties of wavelet noise rather than computing it directly from the intensity values.

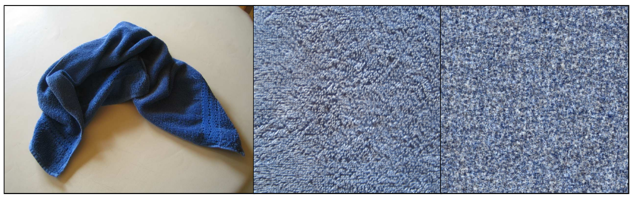
The method of Heeger and Bergen and our method both use histogram matching, but in a different way. Our method uses histogram matching to compensate for the fact that multiresolution wavelet noise always produces a texture with a Gaussian intensity distribution, and uses a direct spectral method to match the textures. In contrast, the method of Heeger and Bergen iteratively matches the histograms of the levels of the pyramids of the textures to match the textures.

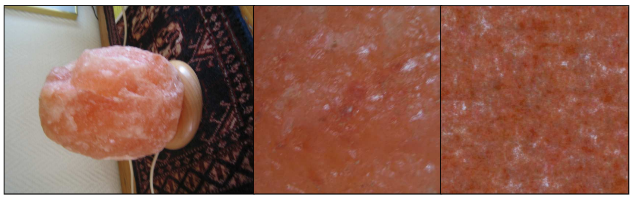
Ghazanfarpour and Dischler [1995; 1996] introduced a spectral method for automatic solid procedural texture generation from a single or multiple 2D example textures. Their method constructs a procedural texture from an example texture, consisting of a summation of a small number of weighted cosines, determined by the frequencies with the highest amplitude in the Fourier transform. Their method is designed for textures with only a few dominant frequencies, such as checkerboards or wood, and cannot handle textures with content on a large number of frequencies. In contrast, the method presented in this paper is designed for isotropic stochastic textures, and can handle content on a large number of frequencies.

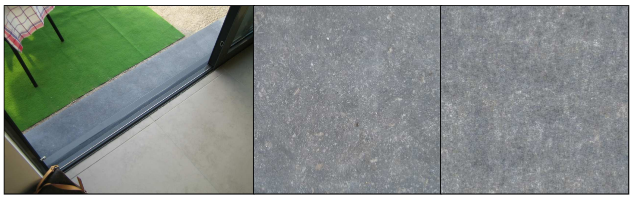
Bourque and Dudek [2004] presented a very general system that performs a two-phase search over a library of procedural shaders. The first global search determines the texture class, and the second local search determines the parameters of the procedural texture. In contrast, our method directly computes the parameters of the procedural texture, but is designed for isotropic stochastic textures. A local search has significant disadvantages compared to a direct computation. A local search is less efficient than a direct computation, since evaluating a set of parameters requires rendering the texture and evaluating the similarity to the example texture, and less accurate than a direct computation, since the best parameters have to be discovered in the potentially high dimensional space of parameters.

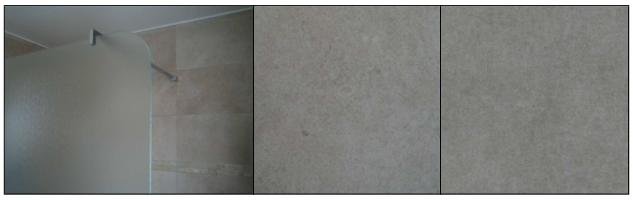
Methods for texture synthesis by example, including parametric methods such as the method of Heeger and Bergen [1995], can usually not be formulated as a procedural texture. Non-parametric methods for texture synthesis by example, such as the method of Wei and Levoy [2000], generally need the example texture in order to synthesize a new texture, and do not allow to strictly decouple the

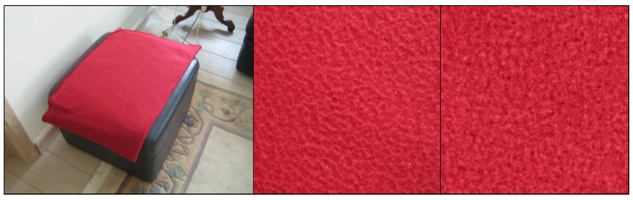

(a) Vinyl flooring.

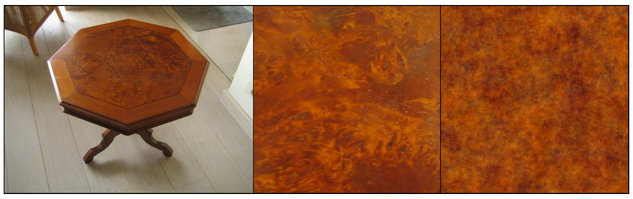

(b) Blue towel. *

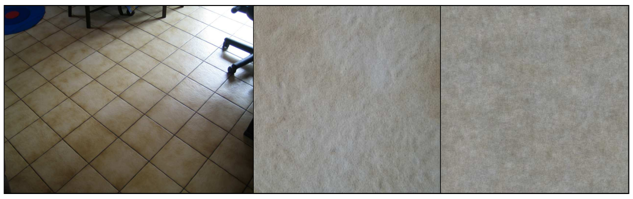

(c) Rock salt lamp. *

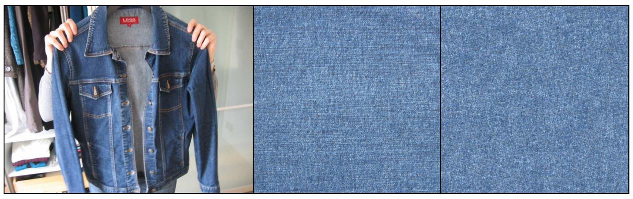

(d) Bluestone doorstep. *

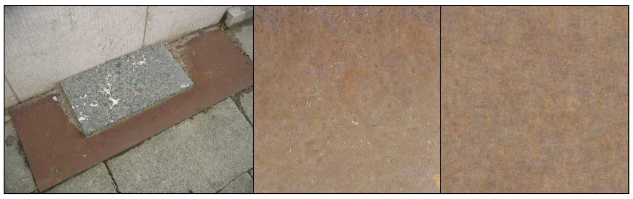

(e) Ceramic shower tile.

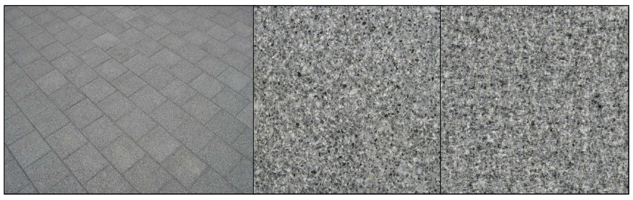

(f) Blanket. *

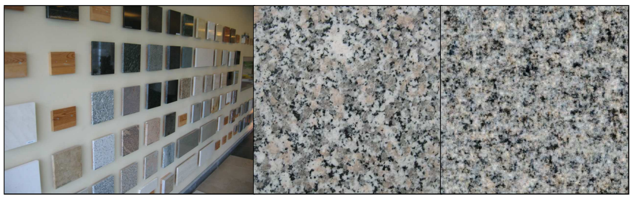

(g) Octagonal table.

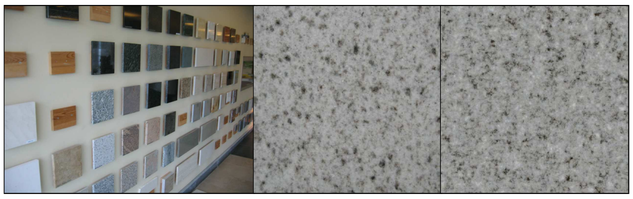

(h) Living room tile.


(i) Jeans vest. *


(j) Rust.


(k) Clinker. *


(l) Granite (Rose Sarde). *


(m) Granite (Bethel White). *

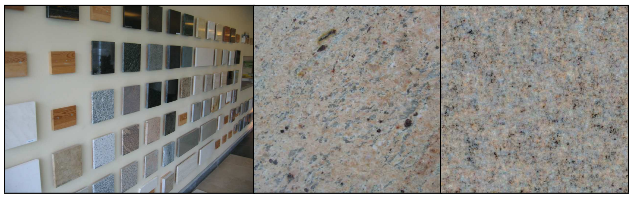
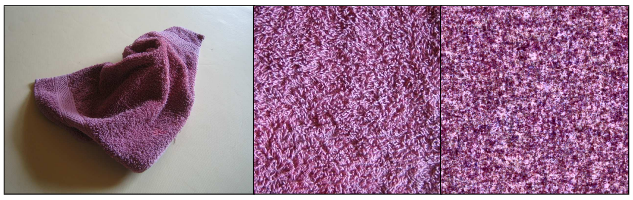
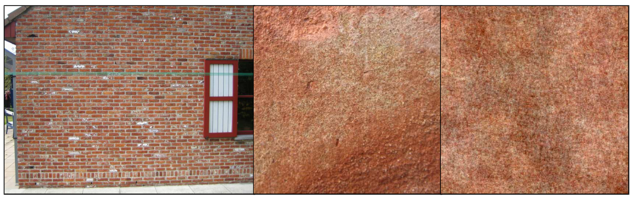
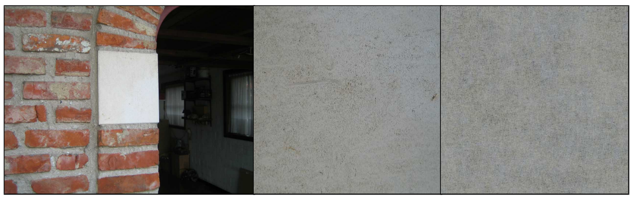
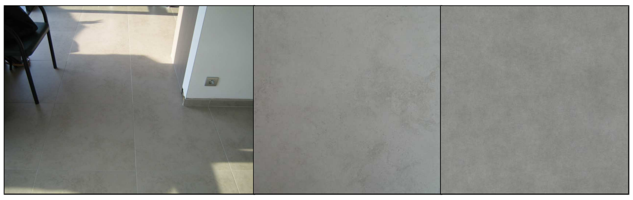

(n) Granite (Shivakasi Yellow). *

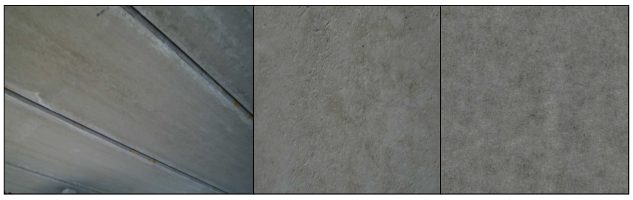
Figure 9: *Several successful examples of procedural textures created with our method. Each example shows a photograph of a real-world scene, a cropped version of a photograph of a texture, and a procedural texture automatically generated from the cropped photograph. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.*

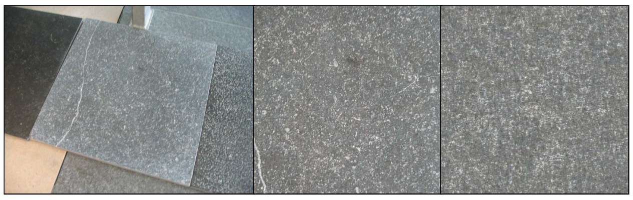

(a) Purple towel. *

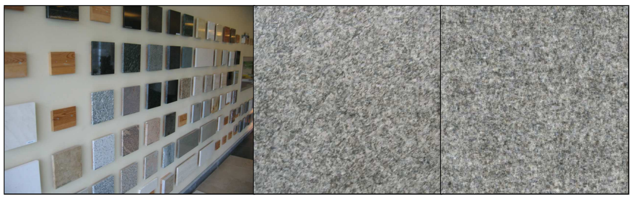

(b) Brick wall.

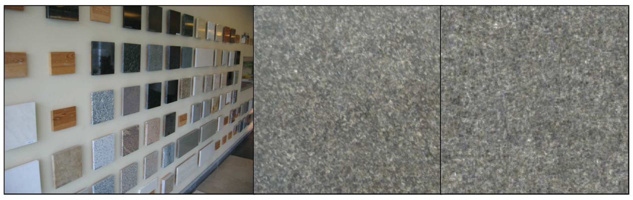

(c) Limestone brick. *

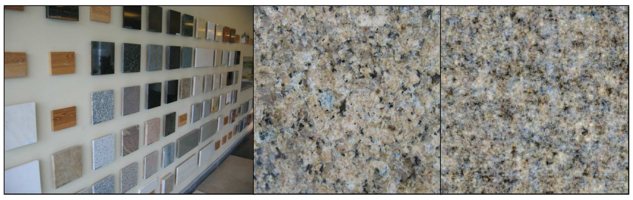

(d) Ceramic tile.

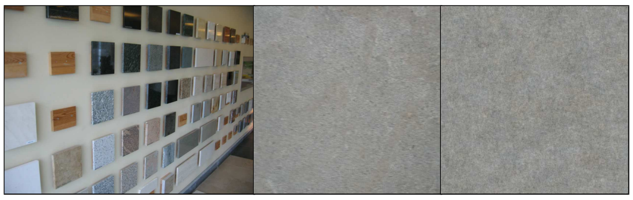

(e) Concrete ceiling. *

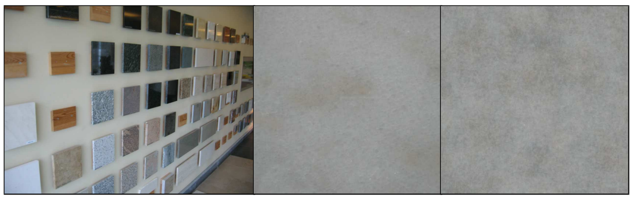

(f) Stone tile. *

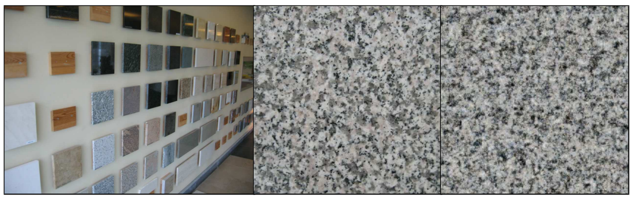

(g) Granite (Jasberg). *

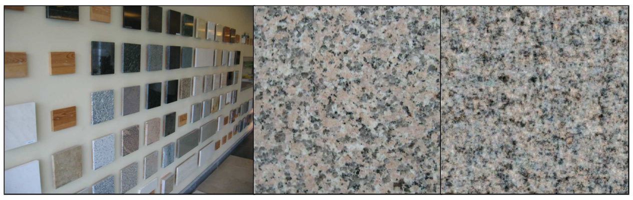

(h) Granite (Rustenbuild). *


(i) Granite (Giallo Veneziano). *


(j) Marble (Azul Valverde). *


(k) Marble (Bianco Royal). *


(l) Granite (Mountain Pink). *


(m) Granite (Porrino). *

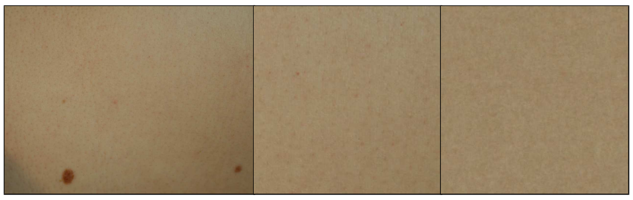

(n) Human skin.

Figure 10: *Several successful examples of procedural textures created with our method. Each example shows a photograph of a real-world scene, a cropped version of a photograph of a texture, and a procedural texture automatically generated from the cropped photograph. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.*

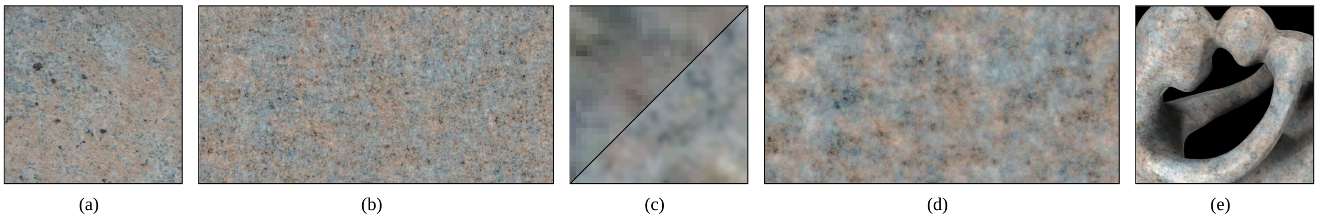


Figure 11: *Advantages of procedural textures by example. (a) An image texture. (b) A larger procedural texture automatically generated from the image texture. (c) A close-up of the top left corner of the image texture and the procedural texture. (d) An edited version of the procedural texture. (e) A rendering of a virtual scene textured using the solid version of the procedural texture. In contrast to the image texture, the procedural texture is compact ($768\ kB$ versus $1\ kB$), is not limited in size and resolution, can be edited by manipulating the weights, and removes the need for a texture parameterization.*

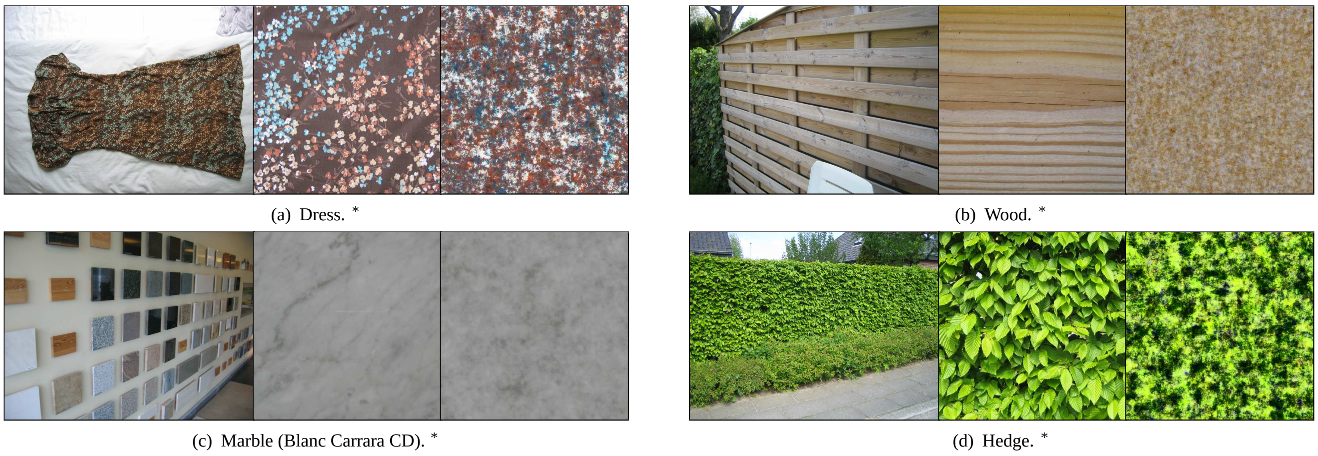


Figure 12: *Some unsuccessful examples of procedural textures created with our method. Each example shows a photograph of a real-world scene, a cropped version of a photograph of a texture, and a procedural texture automatically generated from the cropped photograph. The procedural textures in the examples marked with an asterisk * were constructed using histogram matching.*

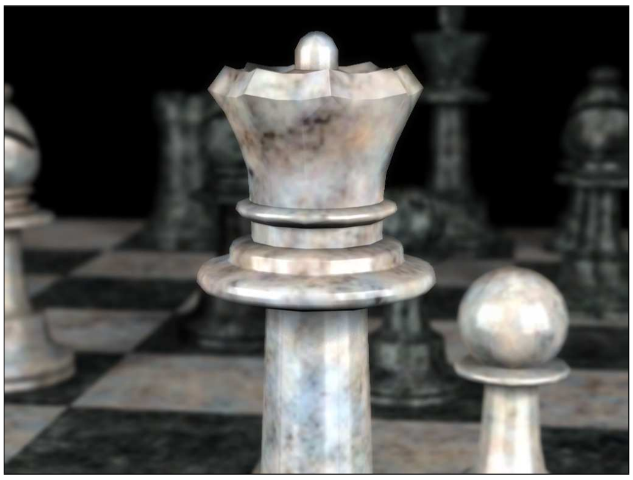


Figure 13: *A virtual scene textured using solid procedural textures created with our method. Our method allows us to extrapolate 2D procedural textures to solid procedural textures simply by evaluating the procedural textures using 3D wavelet noise bands.*

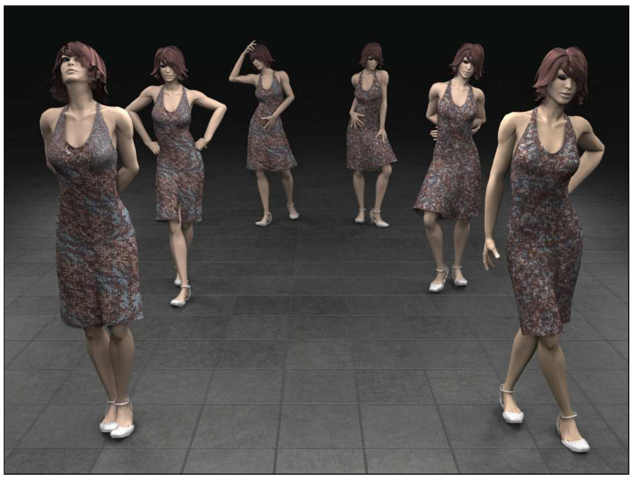


Figure 14: *Perceptual motivation for using procedural textures created with our method. The three dresses on the left use the image texture and the three dresses on the right use the procedural texture. The textures appear more similar with increasing distance.*

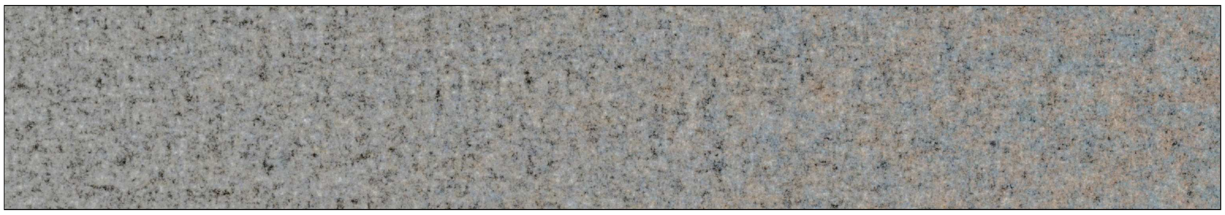


Figure 15: *Texture morphing. The texture of figure 9(m) is morphed into the texture of figure 11 by interpolating the parameters of the corresponding procedural textures.*

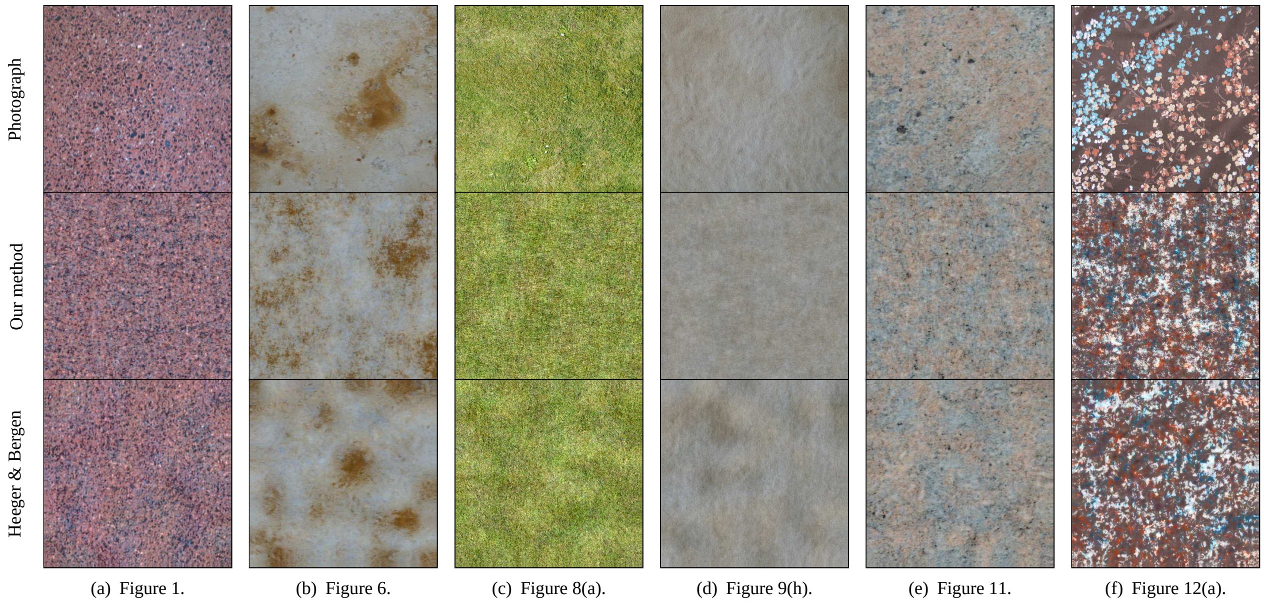


Figure 16: *A comparison of our method to the method of Heeger and Bergen. Each example shows a cropped version of a photograph of a texture, a procedural texture automatically generated from the cropped photograph using our method, and an image texture synthesized from the cropped photograph using the method of Heeger and Bergen. The results of both methods are similar, but the method of Heeger and Bergen cannot be formulated as a procedural texture.*

analysis and the synthesis phase. There have been efforts however to port some of the 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.

10 Conclusion

We have presented a perceptually motivated method for creating procedural textures by example, designed for isotropic stochastic textures. From a single uncalibrated photograph of a texture we compute a small set of parameters that defines a procedural texture similar to the texture in the photograph. Our method allows us to replace image textures with similar procedural textures, combining the advantages of procedural textures and image textures. Creating procedural textures by example has the potential to dramatically improve the texturing and modeling process. Our method is fast, robust and fully automatic. The method presented in this paper is highly practical and is ready to be integrated into production rendering software.

In future work, we would like to extend procedural textures by example to more texture classes. Our method could probably be extended to anisotropic stochastic textures using anisotropic noise functions, such as the recently proposed anisotropic noise of Goldberg et al. [2008]. Parametric methods for texture synthesis by example are most likely also a useful source of inspiration. We believe that a general system for procedural textures by example could be constructed by combining a global search method to determine the texture class, like the one by Bourque and Dudek [2004], with a method for determining the parameters of a procedural texture, like the one presented in this work, for each texture class.

Acknowledgments

Ares Lagae is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO). Toon Lenaerts acknowledges the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). We are grateful to Eero P. Simoncelli for his implementation of the steerable pyramid. We are grateful to Tom Mertens for his implementation of the method of Heeger and Bergen.

References

ALLEGORITHMIC, 2008. Mapzone 2.6. `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*, 361–368.

BOURQUE, E., AND DUDEK, G. 2004. Procedural texture matching and transformation. *Computer Graphics Forum 23*, 3, 461–468.

CATMULL, E. E. 1974. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of Computer Science, University of Utah.

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

DISCHLER, J.-M., AND GHAZANFARPOUR, D. 1999. Interactive image-based modeling of macrostructured textures. *IEEE Computer Graphics and Applications 19*, 1, 66–74.

DISCHLER, J.-M., AND GHAZANFARPOUR, D. 2001. A survey of 3d texturing. *Computers and Graphics 25*, 1, 135–151.

DISCHLER, J.-M., GHAZANFARPOUR, D., AND FREYDIER, R. 1998. Anisotropic solid texture synthesis using orthogonal 2d views. *Computer Graphics Forum 17*, 3.

DONG, Y., LEFEBVRE, S., TONG, X., AND DRETTAKIS, G. 2008. Lazy solid texture synthesis. *Computer Graphics Forum 27*, 4, 1165–1174.

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

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, 1033–1038.

ERINGIS, M. 2006. A completely procedural approach to materials.

FOURNIER, A., FUSSELL, D., AND CARPENTER, L. 1982. Computer rendering of stochastic models. *Communications of the ACM 25*, 6, 371–384.

FRITSCH, F. N., AND CARLSON, R. E. 1980. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis 17*, 2, 238–246.

GARDNER, G. Y. 1984. Simulation of natural scenes using textured quadric surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 84) 18*, 3, 11–20.

GHAZANFARPOUR, D., AND DISCHLER, J.-M. 1995. Spectral analysis for automatic 3-d texture generation. *Computers and Graphics 19*, 3, 413–422.

GHAZANFARPOUR, D., AND DISCHLER, J.-M. 1996. Generation of 3d texture using multiple 2d models analysis. *Computer Graphics Forum 15*, 3, 311–323.

GOLDBERG, A., ZWICKER, M., AND DURAND, F. 2008. Anisotropic noise. *ACM Transactions on Graphics 27*, 3.

GONZALEZ, R. C., AND WOODS, R. E. 2008. *Digital Image Processing*, 3rd ed. Prentice Hall.

HAN, C., RISSER, E., RAMAMOORTHI, R., AND GRINSPUN, E. 2008. Multiscale texture synthesis. *ACM Transactions on Graphics 27*, 3, 1–8.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. 229–238.

JAGNOW, R., DORSEY, J., AND RUSHMEIER, H. 2004. Stereological techniques for solid textures. *ACM Transactions on Graphics 23*, 3, 329–335.

JOLLIFFE, I. 2002. *Principal component analysis*, 2nd ed. Series in Statistics. Springer.

JULESZ, B. 1962. Visual pattern discrimination. *IEEE Transactions on Information Theory 8*, 2, 84–92.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2D exemplars. *ACM Transactions on Graphics 26*, 3, 2.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics 24*, 3, 795–802.

LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Transactions on Graphics 24*, 4, 1442–1461.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Transactions on Graphics 24*, 3, 777–786.

LEFEBVRE, L., AND POULIN, P. 2000. Analysis and synthesis of structural textures. In *Graphics Interface*, 77–86.

LEWIS, J. P. 1989. Algorithms for solid noise synthesis. *Computer Graphics (Proceedings of ACM SIGGRAPH 89) 23*, 3, 263–270.

MALIK, J., AND PERONA, P. 1990. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A 7*, 5, 923.

NORTON, A., ROCKWOOD, A. P., AND SKOLMOSKI, P. T. 1982. Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space. *Computer Graphics (Proceedings of ACM SIGGRAPH 82) 16*, 3, 1–8.

PEACHY, D. R. 1985. Solid texturing of complex surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 85) 19*, 3, 279–286.

PERLIN, K. 1985. An image synthesizer. *Computer Graphics (Proceedings of ACM SIGGRAPH 85) 19*, 3, 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., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 40*, 1, 49–70.

QIN, X., AND YANG, Y.-H. 2002. Estimating parameters for procedural texturing by genetic algorithms. *Graphical Models 64*, 1, 19–39.

REINHARD, E., ASHIKHMIN, M., GOOCH, B., AND SHIRLEY, P. 2001. Color transfer between images. *IEEE Comput. Graph. Appl. 21*, 5, 34–41.

RUDERMAN, D. L., CRONIN, T. W., AND CHIAO, C.-C. 1998. Statistics of cone responses to natural images: implications for visual coding. *Journal of the Optical Society of America A 15*, 8, 2036–2045.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH 2000*, 479–488.

WEI, L.-Y., HAN, J., ZHOU, K., BAO, H., GUO, B., AND SHUM, H.-Y. 2008. Inverse texture synthesis. vol. 27, 1–8.

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