Image-Driven Procedural Texture Specification

Eric Bourque Gregory Dudek

Centre for Intelligent Machines
McGill University
Montréal, Canada

Abstract

In this paper we describe an approach to the automated specification of procedural textures to be used in rendering, based on representative samples. Procedural textures exhibit many advantages over traditional surface texturing techniques, but unfortunately finding the correct procedural texture and appropriate parameters to create the desired texture can be a daunting task for even the most experienced computer graphics artists. The method we propose here, which we refer to as *image-based procedural texturing*, allows the specification of the desired texture in the form of a digital input image. From this sample texture, a corresponding procedural texture is found which produces a texture which is perceptually similar to the input sample.

Key words: Texture analysis, texture synthesis, texture perception, procedural textures, parameter estimation.

1 Introduction

Much of the apparent realism of computer generated imagery is due to the appropriate use of texture. This allows us to add richness to a scene that is critical to its visual appeal, as exemplified by the fact that many commercial rendering projects employ hundreds or even thousands of different texture maps or procedural textures. Fortunately, numerous shaders have been developed and many of these have very rich ranges of textures that they can synthesize. This, in turn, leads to the issue of *how* one can select the appropriate settings for a procedural texture in a particular application; that is the subject of this paper.

Procedural texturing allows an algorithm to describe how a textured surface should appear. There are several advantages to procedural texturing, including abstraction, parameterization, resolution independence, and a compact representation. Moreover, procedural textures do not suffer from many of the shortcomings of traditional texture mapping using texture images. Procedural textures also have the advantage that a wide range of related textures can be specified by a single procedure.

In this paper, we will describe a technique we refer to as *image-based procedural texturing*; a technique which combines the ease of static texture mapping[1] with the power of procedural texturing. In image-based procedural texturing, the user specifies a digital input image, and a procedural texture which generates a similar[2] texture is found. This gives the graphic artist much more freedom to be creative, minimizing the arduous time spent tweaking various aspects of the procedural framework. This technique has many promising applications as well as future directions.

2 Motivation

Despite the numerous advantages of using procedural techniques for texturing, there are, unfortunately, some shortcomings to the approach. The most notable is that once one has a procedural shader, it can be quite difficult to obtain the correct parameters to synthesize the desired texture. Procedural textures often exhibit localized instability, i.e., a small change in a parameter can lead to a large change in the synthesized texture for isolated subdomains of the parameter domain. Even when the procedural texture is stable, some shaders have an enormous number of parameters which can be exceedingly cumbersome to specify manually[3]. These factors can make it difficult for the end user to obtain the desired results, even though the procedural texture is capable *in principle* of achieving them.

The creation of procedural shaders is not a task suited to a non-specialist since it requires an algorithmic formulation of how the texture should appear. This is obviously more difficult than using a typical paint program to create a single texture to be used as a texture map. Moreover, the implementers of procedural shaders must worry about technical problems such as anti-aliasing. In gen-

[1] We refer to the concept of using a static image to be mapped directly to a surface as *static texture mapping*, to avoid confusion with the term *procedural texturing*.

[2] The notion of similarity will be explored further in Sec. 5.2.

[3] The water surface shader in the film *The Perfect Storm* had 299 parameters! Apparently, there was no single individual at ILM who knew what each parameter controlled [8].

eral, the author of a procedural shader does not know at which points the texture will be sampled, and must therefore minimize the high frequency content through the use of smooth edges. This is an example of one of the many issues which must be resolved during the specification of a procedural shader.

Image synthesis is usually slower when using procedural textures rather than static textures since a (sometimes complex) procedure must be evaluated for every sample (pixel) in addition to the illumination model. There has been recent work by Peercy *et al.* [12] which addresses this issue and their preliminary results look promising. Rendering speed is, however, perhaps the smallest concern when weighed against the advantages of using procedural textures, especially considering the dizzying pace at which graphics hardware improves.

The approach we outline here for the automatic specification of a procedural texture from an initial sample attempts to address these shortcomings in a principled and general-purpose way.

3 Background

A procedural texture is a function of a set of input parameters $\mathbf{x} = (x_1, \ldots, x_n)$ which returns the value of a local surface property (typically reflectance) at the point (u, v) queried:

$$P(u, v, \mathbf{x}) = f(u, v, \mathbf{L}, \mathbf{N}, O_d, O_s, \ldots, \mathbf{x}) \qquad (1)$$

where P is a procedural texture having a parameter vector $\mathbf{x}$ and like a static texture map is indexed by $(u, v) \in [0, 1]^2$. The input parameters thus determine the appearance of the surface over the region being textured by the procedural texture. Internally P is a function not only of the parameters of the texture itself, but also of the light direction (L), surface normal (N), object color (O_d, O_s), etc., as illustrated with the function f above. Each procedural texture (often called a shader) will be represented by a unique function requiring a distinct set of parameters relative to that texture.

4 Previous Work

Texture synthesis and analysis have been an active research topic and several methods for texture generation from pictorial samples have been formulated [9, 5, 7, 13, 16]. What all of these techniques have in common is that they produce an arbitrary amount of texture which resembles an input texture sample. They differ only in their methodology and results.

Most notable for use in computer graphics is perhaps [16], since the results reported appear to be able to reproduce almost-arbitrary texture samples more convincingly than previous methods with added computational efficiency. Unfortunately, all existing techniques suffer from two drawbacks: 1) they produce fixed resolution textures and as such fall prey to all of the problems inherent with static texture mapping and 2) the resulting textures cannot be parametrically modified to achieve an appearance *similar to*, but different from, a specified sample. Another shortcoming of the local neighborhood search and fill techniques [7, 16] is not only that textures are produced at a fixed resolution, but that the output resolution can never be higher than the resolution of the sample texture. While increasing the size of the desired synthesized texture will produce more texture, the resolution will remain the same. Other advantages of procedural textures, namely compact representation, unlimited resolution, arbitrary complexity, as well as the potential for higher dimensional textures are simply not available in traditional sample-based texture synthesis.

Dana *et al.* [4] have a somewhat different approach to the texture synthesis problem, and propose a method which is similar to the techniques used for measuring the bi-directional reflectance distribution function (BRDF). They define a bi-directional texture function (BTF) analogously to the BRDF: for each possible viewing and illumination direction, the BTF of a particular texture returns an image. This model accommodates both isotropic and anisotropic textures very well. Actual textured materials (carpet, velvet, stucco, etc.) are imaged for each possible viewing and lighting angle, and the resulting texture images are stored for later use. At rendering time, the correct texture samples are retrieved and blended together. Although this approach has produced some nice demonstration images, the time and machinery necessary to accurately sample each desired texture for all orientations, and the space needed to store these samples is quite prohibitive. Suen and Healey [14] have proposed a method for calculating a basis set of minimal dimension when using a BTF to avoid storing entire texture databases.

One method for automating parameter estimation for procedural texturing involves the explicit modeling of features present in certain types of highly structured textures [10]. While this work provides some interesting examples for brick and wood, their method is dependent on determining measurable features on a per-texture basis, which unfortunately cannot be generalized. Moreover, it is not clear that this method can be used for textures which are not structured; a class which is perhaps more frequently used in procedural texturing.

5 Approach

We wish to eliminate the difficulties in using procedural textures by allowing the end user to specify one or more desired texture samples using pictures (i.e. photographs), and provide them with a procedural texture which is sim-

ilar to the input sample. We call this technique *image-based procedural texturing*.

Consider a set $\mathcal{S}$ of tuples $\langle P_i, D_i \rangle$, where P_i is a procedural texture of arbitrary dimension. Given a texture sample T, we wish to find a procedural texture $P \in \mathcal{S}$, and an associated parameter vector $\mathbf{x}$ such that $P(\mathbf{x})$ produces a texture perceptually similar to T. The process for finding P and $\mathbf{x}$ are outlined below.

Achieving the above is critically dependent on a measurement function D_i that serves to estimate the perceptual similarity of two textures. We will refer to this as the *similarity function* for that texture (for details refer to Sec. 5.2).

We assume at the outset that the context of our input sample texture T can be approximated using a procedural texture. We then need to find procedural textures in the set $\mathcal{S}$ which have some similarity with T. We will call this set of candidate procedural textures $\mathcal{Q}$ ($\mathcal{Q} \subset \mathcal{S}$). Depending on how we design our classification algorithm, the set $\mathcal{Q}$ could be empty, or we can force it to always have at least one element.

Each candidate procedural texture $Q_i \in \mathcal{Q}$ will have an associated *parameter space*, that is, the space corresponding to the domain of all valid parameters, and an output *texture space* corresponding to the range of all textures which can be synthesized using this procedural texture. The dimensionality of the parameter space associated with Q_i is generally given by the number of input parameters, and that of the texture space is equal to the number of pixels in the synthesized texture multiplied by the dimensionality of each pixel. For each candidate procedural texture Q_i we must find a parameter vector $\mathbf{x}_i$ such that $D_i(T, Q_i(\mathbf{x}_i))$ is minimized. We then choose the $Q_i(\mathbf{x}_i)$ from the set $\mathcal{Q}$ which has the lowest value for D. This is the $P(\mathbf{x})$ which was sought.

5.1 Searching in Texture Space

In general, the (high-dimensional) surface produced by evaluating under D_i is unlikely to be convex. As a result, the process of finding the correct parameters entails the use of non-convex optimization. While exhaustive search would, in theory, eventually produce a correct result, our desire for a solution in interactive time suggests a two stage approach: a preliminary search using precomputed data and an on-line refinement stage.

For each $Q \in \mathcal{Q}$ we first search globally over the parameter space considering samples which are uniformly distributed for each parameter. Synthesizing the corresponding texture for each sample point in the parameter space to compare against the target texture T is a local, independent operation, and therefore can be easily parallelized.

Once the best sample point from the sparse search has been found, we perform a local search to find the parameter vector $\mathbf{x}_i$ which produces the texture $Q_i(\mathbf{x}_i)$ closest to T. The notion here is that the global search has brought us close enough to the actual target so that a local optimization method will converge without getting caught in local minima.

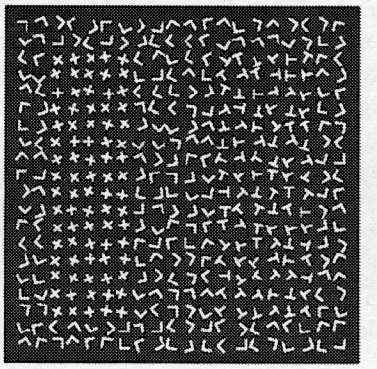


Figure 1: An example of texture segregation (left) and two textures which do not segregate (right). This figure originally appeared as Fig. 17.1 in [3, p. 254]. Courtesy of Jim Bergen.

5.2 Evaluating Texture Similarity

The specification of the texture similarity function D_i is important to the performance of our method. There are, of course, many possible functions which could be used to calculate the difference between two images, but here we are interested in the *perceptual* difference between two textures. A simple sum of squares distance (SSD) between respective pixels, for example, is not appropriate in this case: consider two images of the same texture which are slightly offset from one another. Clearly, these two images contain the same texture, but their SSD measures could be quite large.

Formally, we can think of an ideal measure of the perceptual difference between textural characteristics in two images:

$$D^*(T_1, T_2) \in [0, 1] \tag{2}$$

where a value of 0 indicates that T_1 and T_2 are indiscernible, and a value of 1 implies that the two textures are maximally distinct perceptually.

While the understanding of human texture recognition and discrimination remains an area of ongoing research and some disagreement, several consistent results have been developed formally [1]. Current theories of texture discrimination maintain that when two textures produce a similar response to frequency-selective oriented linear

filters they are often indistinguishable [2, 1, 9]. Textures which do not segregate can be considered to be similar perceptually, while textures which do segregate are dissimilar (Fig. 1).

We wish to determine a computable similarity function D, to approximate D^*. Different types of textures may lend themselves to different computable approximations to D^* which is why our framework allows each procedural texture P_i to have an associated similarity function D_i in the set $\mathcal{S}$.

In practice thus far we have used a single D to approximate D^*:

$$D(T_1, T_2) = \sum_x \sum_y \Big(F_m(T_1)[x, y] - F_m(T_2)[x, y] \Big)^2 \tag{3}$$

where $F_m(T)[x, y]$ is the intensity of the pixel at location (x, y) in the magnitude of the amplitude image of the Fourier transform of T.

This measure is reasonable since it captures similarities which are located in the frequency domain and is thus comparable to the image pyramid analysis proposed by Heeger and Bergen [9] with the advantage that it can be efficiently computed. Our preliminary results suggest that this measure works well in practice.

6 Experimental Results

Textures have often been described as being either deterministic or stochastic. Deterministic textures exhibit some form of underlying pattern, and can be thought of as having rules which govern the placement of the textural primitives. Stochastic textures, on the other hand, do not have any easily identifiable primitives. Many textures are in fact some combination of both deterministic and stochastic textures. Most texture synthesis techniques to date work well only for stochastic textures which are both stationary and local [9, 5, 16]. In this section, we will demonstrate the power of our system using both deterministic and stochastic procedural textures. To validate our method, we first illustrate our technique by replicating a texture which has itself been generated using a procedural texture whose parameters we are attempting to find. This allows us to clearly observe the extent of the mismatch in the final result that is due to imperfect parameter fitting, as opposed to errors due to shortcomings in the expressive ability of the procedural texture itself. As a second example, we apply our framework to a texture sample which has not been created procedurally.

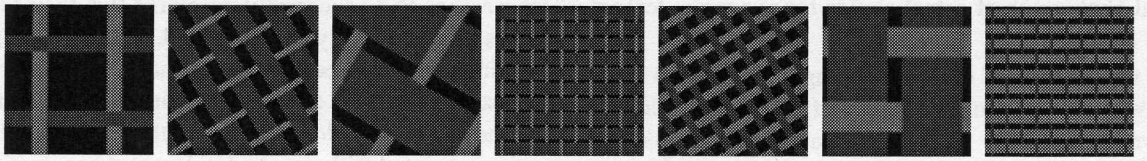
Our system uses procedural shaders which are written in the RenderMan Shading Language [15], although it can easily support other procedural shading and texturing languages. In addition to standard use of the Shading Language in the graphics community, we can take advantage of some recent techniques described in [12] to accelerate the rendering of procedural textures when using this language.

In our first example, we have chosen a weave texture based on an example in Steve May's RManNotes [11] (see Fig. 2). This shader has four parameters: the frequency of the weave, the orientation of the weave, the width of the horizontal stripes, and the width of the vertical stripes. There are many possible textures associated with random samples of the parameter domain as is shown in Fig. 2(a). Figure 2(b) shows the target texture T we are trying to synthesize using our system for parameter estimation. The candidates from the global search as outlined in Sec. 5.1 are shown in Fig. 2(c). For each of these candidates, we performed a local search and the resulting texture with the lowest distance value is shown in Fig. 2(d). Because the target texture originated from the same procedural texture, we can verify the accuracy of the parameters found by our method. The parameters for Fig. 2(b) were $\langle 23, 2.5, 0.22, 0.78 \rangle$ for orientation, frequency, horizontal width, and vertical width respectively. The parameters recovered by our method were $\langle 22.9639, 2.50109, 0.213725, 0.785673 \rangle$. These results are satisfying both visually and numerically.

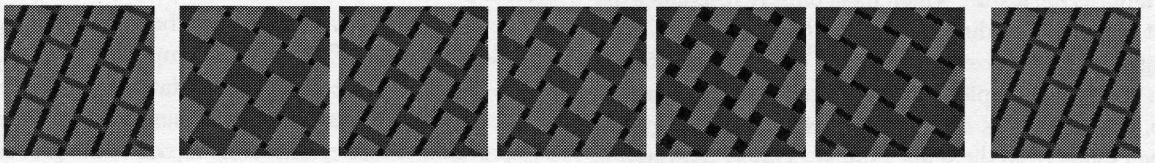
To verify the behavior of our system using a more stochastic procedural texture, we used Musgrave's `puffyclouds` shader [6, p. 295]. This procedural texture makes use of fractional Brownian motion (fBm), and as such has parameters for the number of octaves making up the fBm, the lacunarity, the fractal increment, parameters for the color of the sky and clouds, as well as a threshold for choosing between the sky and cloud color depending on the value of the fBm. This number of parameters, as well the trial and error guesswork involved in the specification of their values for the desired results is common when using procedural textures. These parameters also do not have intuitive meanings for non-specialists as compared to the parameters which control the weave shader discussed above. As such, this particular shader serves well to demonstrate the usefulness of this technique for graphic artists.

For the purposes of these illustrative experiments, we have fixed the number of octaves in the fBm since it is a function of the resolution of the rendered image, and also fixed the colors for the sky and clouds to their defaults in order to reduce the dimensionality of the search space.

It will not always be the case that we can find a close match if the target texture is not contained in the texture range of the procedural texture which we are searching. This will frequently be the case when using real photographic images as target textures (which is our intended

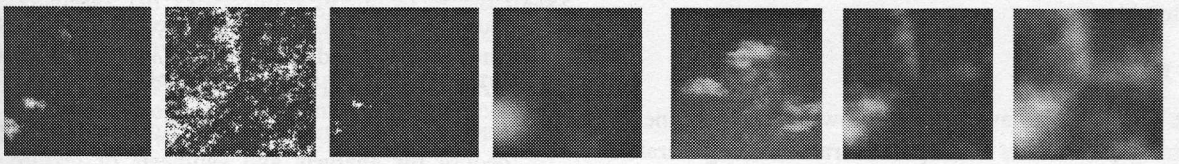


(a) A few samples from the texture range of the weave shader.



(b) (c) Top candidates matching target texture found during the global search. (d)

Figure 2: Results from using our framework on a procedural texture of a weave (a deterministic texture). (a) shows a sampling of the various weave textures which can result from using different parameter vectors. The texture whose parameters we wish to estimate in shown in (b). The sparse global search over the texture range returns the candidates shown in (c), which are deemed to be the closest to the target texture. A local search is completed starting from each of the candidates in (c) to find the parameter vector which will give us a resulting texture as close to the target as possible. This resulting texture is shown in (d).



(a) Samples from the texture range of the cloud shader. (b) (c) (d)

Figure 3: An example of how our system performs for a stochastic texture. As in Fig. 2, a few samples of the range of the procedural texture are shown in (a). The target texture (an actual picture of clouds) is shown in (b). The closest match from the global search is shown in (c), and the final texture resulting from the local search in shown in (d).

application). In this situation, we can only hope to find a parameter vector $\mathbf{x}$ such that $P(\mathbf{x})$ is as perceptually similar as possible to the target texture T. In order to demonstrate this principle, we have chosen to use a traditional photograph of actual clouds for our target texture (Fig. 3(b)).

A few images of clouds resulting from various samples within the parameter domain of the cloud shader are shown in Fig. 3(a). The global search over the parameter domain chose the image shown in Fig. 3(c) as the closest match. From this texture, a local search was performed which resulted in the image shown in Fig. 3(d). The latter is the texture which most closely resembles the input sample according to our similarity function. Although the synthesized images are noticeably different from the input samples (unlike those of the weave texture), the search has captured the critical characteristics of the clouds from the input sample. This is reasonable, given that it is unlikely that there is a parameter vector which will result in an image which is exactly the same as the input sample. Because we have chosen to use a target texture from an actual photograph, we cannot verify our results numerically as we could with the weave shader, and must rely exclusively on the visual appearance.

For each texture, the global search was performed using a low density (less than 100 samples) of uniformly distributed sample points within the parameter domain of the respective procedural textures. The local refining searches from the top candidates found in the global search converged quite quickly as is illustrated in Fig. 4.

The results reported above suggest that this technique for parameter estimation does in fact work well for both deterministic and stochastic procedural textures. While we have shown only a limited number of procedural textures, our preliminary results are promising, and suggest that this technique can be used for a wide variety of procedural textures.

7 Discussion

The approach we have outlined allows us to determine the suitable parameters for a procedural texture generation function to allow it to approximate a given input texture, assuming that the input texture sample is in the domain of the procedural texture generator. This allows for the automated specification of procedural parameters that might otherwise be difficult to discover. One issue, however, is that for a texture that is well outside the domain of the procedural texture generator's output, it may be difficult to detect that the texture cannot be replicated. While this problem appears intuitive, its rigorous solution in fact relates to a range of difficult computer vision problems and is the subject of ongoing research.

A more technical issue is the selection of appropriate sampling parameters for the space of procedural textures that can be generated. The textures explored in this paper did not require a dense sampling in order for the local searches to be successful, implying that their texture spaces were relatively well behaved. Specifically, given a setting of the procedural texture that was a very rough approximation of the target texture, it was possible to refine the match using local search methods. As the texture space associated with a procedural texture becomes more unstable (and has more local minima), a more dense sampling will be required in order to correctly seed the local search.

The techniques we have described have a few applications outside of traditional computer generated imagery. Because procedural representations of textures are very compact, there are obvious connections with image compression. Moreover, current trends show processing power increasing much more rapidly than network bandwidth. As such, implicit representations are well suited to this scenario where the representations are transferred across networks to be rendered in real-time on the client computer.

8 References

[1] James R. Bergen. *Spatial Vision*, volume 10, chapter 5: Theories of visual texture perception, pages 114–134. CRC Press, 1991.

[2] James R. Bergen and Edward H. Adelson. Early vision and texture perception. *Nature*, 333:363–364, 1988.

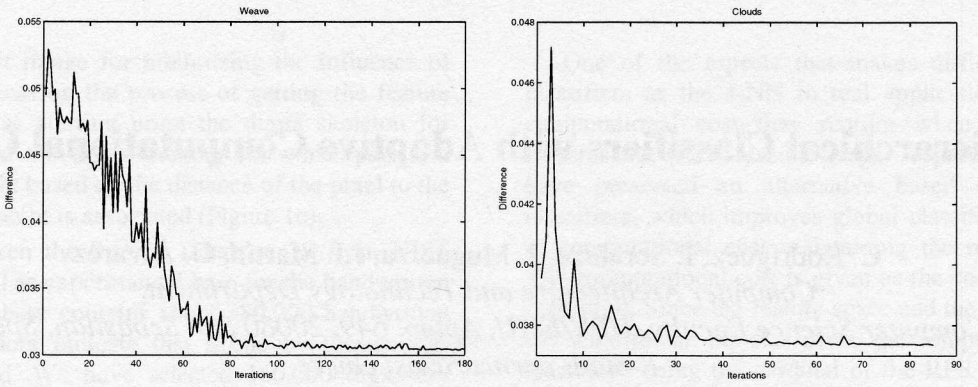
[3] James R. Bergen and Michael S. Landy. *Computation Models of Visual Processing*, chapter 17: Computational Modeling of Visual Texture Segregation, pages 253–271. MIT Press, 1991.

[4] Kristin J. Dana, Bram Van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[5] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of the ACM Computer Graphics (SIGGRAPH)*, pages 361–368, 1997.

[6] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: A Procedural Approach*. Academic Press, 2nd edition, 1998.

[7] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 1033–1038, September 1999.



(a) Weave texture convergence.

(b) Cloud texture convergence.

Figure 4: Graphs illustrating the convergence of the local search for both the weave and cloud textures.

[8] Stefen Fangmeier. Industrial Light & Magic : The making of "The Perfect Storm". Special Session at SIGGRAPH 2000, July 2000.

[9] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the ACM Computer Graphics (SIGGRAPH)*, pages 229–238, 1995.

[10] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Graphics Interface*, pages 77–86, May 2000.

[11] Steve May. RManNotes. http://www.cgrg.ohio-state.edu/%7Esmay/RManNotes/rmannotes.html, 1995.

[12] Mark S. Peercy, Marc Olano, John Airey, and P. Jeffrey Ungar. Interactive multi-pass programmable shading. In *Proceedings of the ACM Computer Graphics (SIGGRAPH)*, pages 425–432, 2000.

[13] Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision (IJCV)*, 40(1), 2000.

[14] Pei-hsiu Suen and Glenn Healey. The analysis and recognition of real-world textures in three dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(5):491–503, May 2000.

[15] Steve Upstill. *The RenderMan Companion*. Addison-Wesley, 1990.

[16] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the ACM Computer Graphics (SIGGRAPH)*, pages 479–488, 2000.