Deterministic Texture Analysis and Synthesis using Tree Structure Vector Quantization

LI-YI WEI

Room 386, Gates Computer Science Building
Stanford, CA 94309, U.S.A.
liyiwei@graphics.stanford.edu

Abstract. Texture analysis and synthesis is very important for computer graphics, vision, and image processing. This paper describes an algorithm which can produce new textures with a matching visual appearance from a given example image. Our algorithm is based on a model that characterizes textures using a nonlinear deterministic function. During analysis, an example texture is summarized into this function using tree structure vector quantization. An output texture, initially random noise, is then synthesized from this estimated function. Compared to existing approaches, our algorithm can efficiently generate a wide variety of textures. The effectiveness of our approach is demonstrated using standard test images from the Brodatz texture album.

Keywords: Texture Analysis, Texture Synthesis, Image Processing

1 Introduction

Many applications in computer graphics, vision, and image processing can benefit from a texture analysis and synthesis algorithm. For example, textures have long been used to decorate object surfaces in computer rendered images. However, natural textures are often difficult to generate manually; therefore an algorithm to synthesize a large texture from a small scanned patch will be desirable. In machine vision, the ability to discriminate, segment, or classify textures is essential for tasks such as automatic inspection. Natural texture images are also notoriously hard to compress due to their high frequency contents, and compact representation may be achieved via a good texture analysis algorithm. Despite these wide applicability, however, an algorithm that is both efficient and capable of representing all the texture properties has yet to come.

In this paper, a new texture analysis and synthesis algorithm is proposed (Figure 1). We base this algorithm on a model that represents textures using a nonlinear deterministic characteristic function ([25]). In a two-phase process, the parameters of this function are first analyzed from an example texture using tree structure vector quantization. In the second phase, a new texture is synthesized based on the estimated parameters. The key advantage of this approach is that it can efficiently generate high quality textures. In addition, it can be easily implemented; the most complex component is tree structure VQ.

The paper is organized as follows. We introduce our texture model in Section 2 . Section 3 presents the texture generation algorithm built upon this model. Section 4 shows some example textures generated by our algorithm. We compare our algorithm with previous techniques in Section 5, and conclude this paper in Section 6.

2 Texture Model

Texture is a common visual experience. However, a precise definition of texture is difficult to formulate ([22]). Traditionally, textures have been categorized as either structural or stochastic. A structural texture is characterized by a set of primitives (texons) and placement rules. For example, a brick wall texture is generated by tiling up bricks (primitives) in a layered fashion (the placement rule). A stochastic texture, on the other hand, do not contain explicit primitives (such as grass and sand). Since most natural textures are neither complete stochastic nor structural, we attempt to characterize both kinds of textures with a single model.

Our texture model is based on the Markov Random Field (MRF, [3]). That is, the probability distribution of the intensity value for a single pixel is completely determined by the intensity values of pixels in its spatial neighborhood. More formally, let $N(x, y)$ be the collection of pixel values in the neighborhood of a pixel located at (x, y). Then the value of pixel (x, y), $I(x, y)$, can be represented as follows:

$$I(x, y) = F(N(x, y)) \tag{1}$$

where F is a probability distribution that determines the pixel value of $I(x, y)$ from its neighborhoods $N(x, y)$. The size of the neighborhood determines the "randomness" of the texture. Textures with regular patterns such as tiled floor require large neighborhoods, while small neighborhoods are sufficient for stochastic textures as such sand beach.

Given such a model, the goal of a texture analysis and synthesis algorithm is to estimate the parameters of F, and generate new textures by sampling from it. However, because sampling from a MRF is usually computationally demanding, we restrict our F to be a deterministic function,

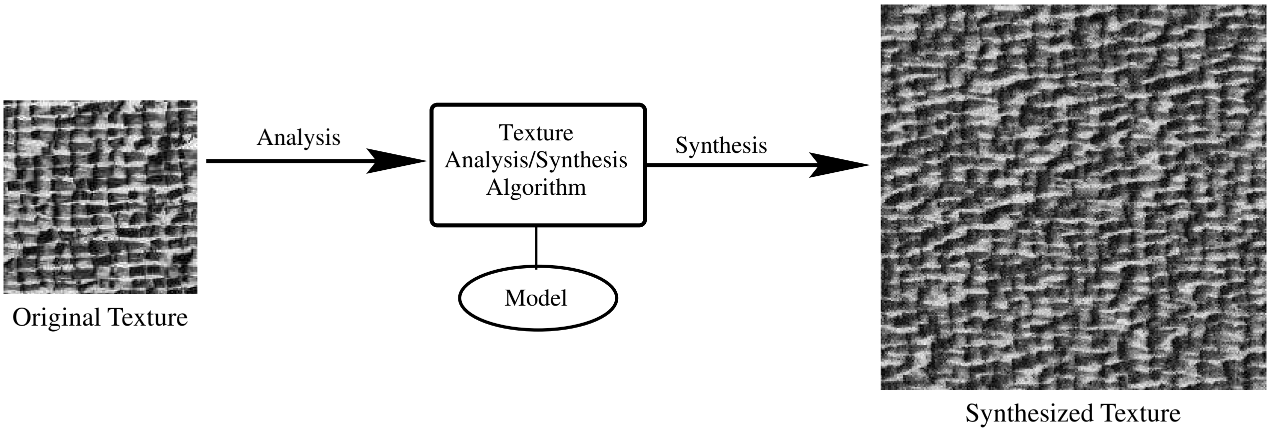


Figure 1: Given an example texture image, our algorithm first analyzes its parameters based on a prior model. A new texture is then synthesized, based on this model along with the estimated parameters.

hereafter referred to as the *Texture Characteristic Function*. As will be demonstrated, restricting F to be deterministic can speed up the computations without sacrificing the quality of the synthesis results.

3 Texture Generation

Our texture generation process starts with an example texture and analyzes the parameters of the corresponding characteristic function (Section 3.1). An output texture, initially random, is then synthesized from this function (Section 3.2). The representation of F is carefully chosen so that both the analysis and synthesis phases can be efficiently executed. For clarity, we summarize the algorithm in Section 3.5.

3.1 Texture Analysis

In the analysis phase, the parameters of F for the input texture are estimated. However, a general representation of F must first be determined. One option is the multilayer neural network, often used to represent complex high dimensional functions. However, because general neural networks may require excessive training time, we use the radial basis function, which can be trained noniteratively.

Radial Basis Function. Given a function $f(x)$, the goal of the radial basis function is to approximate it as a linear combination of some elementary functions $\{\phi_i(x)\}$:

$$f(x) = \sum_{i=1}^{K} w_i \phi_i(x) \tag{2}$$

where $\{\phi_i\}$ are certain simple functions (e.g., the Gaussian function); $\{w_i\}$ are the weight factors for these functions and K is the number of basis functions. Many methods can be used to esimate these parameters. A common way

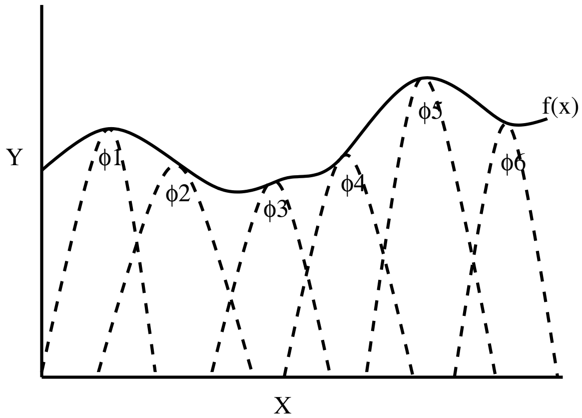


Figure 2: Example of the radial basis function. An arbitrary 1D function, $f(x)$, is approximated as a linear combination of several simple functions, $\{\phi_i\}$. These functions $\{\phi_i\}$ are translated version of a simple 1D Gaussian function.

is to use a clustering algorithm (e.g., vector quantization) to determine the $\{\phi_i\}$ first, and then to use linear equations to solve $\{w_i\}$. The advantage of this approach is that the training is non-iterative and fast, but the parameters may be suboptimal. Better results can be achieved via more rigorous techniques, such as EM (expectation-maxization), with the disadvantage of longer computation time. More information about radial basis functions and the relevant optimization algorithms can be found in [1].

Based on this radial basis function representation, we express our texture characteristic function F as follows:

$$I(x, y) = F(N_{x,y}) = \sum_{i=1}^{K} w_i \phi_i(N_{x,y}) \tag{3}$$

where $N_{x,y}$ is the concatenation of the set of neighborhood pixels $N(x, y)$ of (x, y) into a single vector; $\{\phi_i\}$ are chosen to be Gaussian functions with diagonal covariance matrices; and $\{w_i\}$ are weight factors.

The free parameters of the equation, which include the mean and covariance matrices of the Gaussian functions $\{\phi_i\}$ and the weight factors $\{w_i\}$, are trained from the input texture image. Before the training, the number of basis functions K is decided by the user, and the training neighborhood vectors are gathered from the input texture image. Given these training vectors, K codewords are calculated using standard tree structure VQ (TSVQ, [11]), and these codewords are used as the means of the functions $\{\phi_i\}$. The covariance matrix of the jth basis function, ϕ_j, is then determined from the componentwise variances of the set of vectors mapped to codeword j, and w_j is the average of $I(x, y)$ from the same set of vectors.

Using TSVQ as the training algorithm presents several advantages. First, the training can be efficiently executed. If the number of training vectors is T and the number of basis functions is K, then the average case time complexity for TSVQ is $O(T * log(K))$, which is very fast compared to other algorithms such as EM. Second, $F(N_{x,y})$ can be efficiently evaluated using the tree data structure built during the training process. This is crucial for the synthesis phase, since F needs to be calculated for each output pixel, as shown in the next section.

3.2 Texture Synthesis

In the synthesis process, the output texture will be transformed from a random noise to a new image based on the estimated F. The synthesis process can be described by the following pseudo code:

1. Loop through all pixels (x, y) in the output texture in raster scan order.

2. Collect the neighborhood vector, $N_{x,y}$, of pixel (x, y).

3. Assign $F(N_{x,y})$ to be the synthesized color of pixel (x, y).

Most of the computation during synthesis is devoted to the evaluation of $F(N_{x,y})$. Normally, this evaluation has time complexity $O(K)$, since there are K terms in Equation 3 and each ϕ_i needs to be computed. However, because these Gaussian functions $\{\phi_i\}$ are localized, only a few of them contribute significantly to F. For further acceleration, we only compute one ϕ_k with close mean to the query vector $N_{x,y}$, and use $w_k\phi_k$ as the value of $F(N_{x,y})$. This ϕ_k can be determined quickly using the tree data structure built for F. In other words, evaluating $F(N_{x,y})$ is equivalent to encoding $N_{x,y}$ using tree structure VQ. Because of this structure, the search can be executed with time complexity $O(log(K))$ instead of $O(K)$.

3.3 Neighborhood

Because the output texture is synthesized in a raster scan ordering, we restrict our neighborhood system, $N(x, y)$, to a causal neighborhood, which means that $N(x, y)$ depends only on the previous pixels in the raster scan ordering. A noncausal neighborhood $N(x, y)$ will lead to an iterative synthesis algorithm, which will take longer computation time.

Though we model texture images based on local characteristic functions, the required size of the neighborhood, $N(x, y)$, can vary over different kinds of textures. Small neighborhoods are sufficient for random or micro-structured textures, but textures with extended patterns require larger neighborhoods. Although one may attempt to make the neighborhood as large as possible, two problems can occur: the computation will be slower, and F will be estimated less accurately due to the limited amount of training data. These problems can be alleviated using the multiresolution pyramid.

3.4 Multiresolution Pyramid

The texture analysis and synthesis algorithm introduced above is based on single resolution. To capture extended texture structures while avoiding large neighborhoods, the Gaussian pyramid is used. In a Gaussian pyramid, each successive level is a low pass filtered image of the higher resolution level, so large scale texture structures, which may span several pixels in the highest resolution, will be close together within a small neighborhood in some low resolution level. In this setting, the neighborhood of a pixel (x, y) at resolution level L, $N(x, y, L)$, can contain pixels in the same level L, as well as pixels in lower resolution levels.

Based on the Gaussian pyramid, the texture generation process is modified as follows. In the analysis process, a pyramid is first built from the example texture. Because different levels can have different texture characteristics, we estimate F_L separately for each level L. During the synthesis process, the Gaussian pyramid of the output texture is generated, from lower to higher frequency levels, using the corresponding F_L. The highest resolution will contain the final synthesized texture. Figure 3 shows an example of texture generation using a Gaussian pyramid with 6 levels.

Because the Gaussian pyramid contains at most $1/3$ more pixels of the original image, the time complexity of the analysis phase and synthesis phase are $O(A * log(K))$ and $O(S * log(K))$, respectively, where A is the number of pixels of the original image; S the number of pixels of the output image; and K the number of basis functions.

3.5 Summary

Our texture generation algorithm can be summarized as follows:

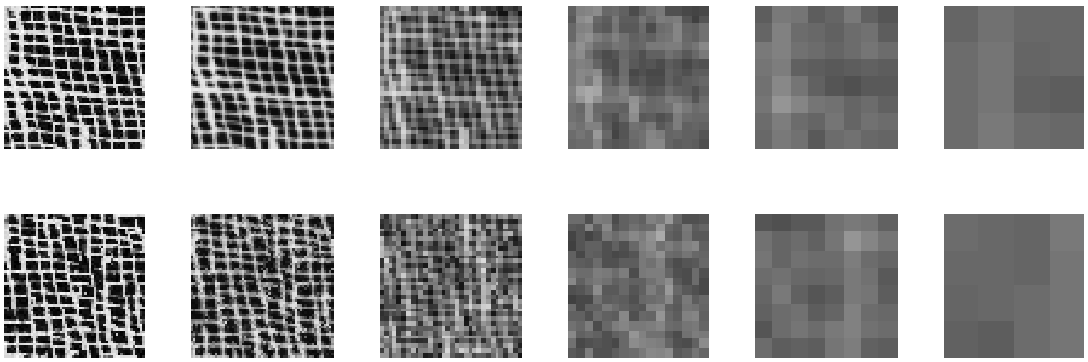


Figure 3: The top row shows the Gaussian pyramid of the input texture, with successive lower resolution images toward the right side. During the synthesis process, the output texture, shown in the bottom row, is sampled from the estimated texture characteristic function, from lower to higher resolutions.

Input:

I_a The example texture image

I_s The output texture image of desired size and shape, initialized to be a white random noise

K The number of basis functions used to approximate the characteristic functions

N The neighborhood system

Analysis:

Step 1 Build a Gaussian pyramid G_a with L levels from the input image I_a.

Step 2 Use TSVQ to train the texture characteristic functions, F_i, for each level i of G_a, using the training data gathered from the neighborhood vectors, $N(x, y, i)$, of all pixels (x, y, i) at the ith level of G_a. This set of functions F_i will be used during the synthesis phase.

Synthesis:

Step 1 Build a Gaussian pyramid G_s with L levels from the output image I_s.

Step 2 Loop through all pixels (x, y, i) in G_s in raster scan ordering, from the lowest resolution to the highest resolution. Collect the neighborhood vector, $N(x, y, i)$, of pixel (x, y, i). Then assign the value of $F_i(N(x, y, i))$ to $G_s(x, y, i)$.

Step 3 The final synthesized texture will be level 1 of G_s.

4 Results and Discussion

Our algorithm is able to synthesize a broad range of textures. Examples using images from the Brodatz texture album ([2]) are shown in Figure 4. Structured textures (D101, D103), textures with dominating orientations (D11, D15), random textures with microscopic structures (D57, D84) or large scale structures (D86, D71) are all successfully synthesized. Our algorithm is also efficient: each texture shown in Figure 4 can be generated in minutes. In comparison, many existing techniques will take hours to generate textures of similar sizes ([28], [18], [19]).

Based on the locality and stationarity assumptions, our algorithm is less effective in modeling textures which violate these conditions. For example, the brick structure in D95 (Figure 5) extends over a wide range of the image area and violates the locality assumption, while D87 (Figure 5), which contains a tree branching structure, can not be described with a stationary process; rather a larger neighborhood or a model capable of representing inhomogeneous structures will be required. Also, a formal theoretical treatment should be justified for several aspects of our algorithm. For example, when starting from an arbitrary white noise, it is not known if the output image will always converge to look like the input texture. Though no failures have been observed, we have no formal proof yet. Finally, parameters such as K and N are currently decided manually, and algorithms that can automatically determine them will be beneficial.

5 Previous Work

Numerous approaches have been proposed for the analysis and synthesis of visual textures. Because an exhaustive survey is out of the scope of this paper, we only compare our

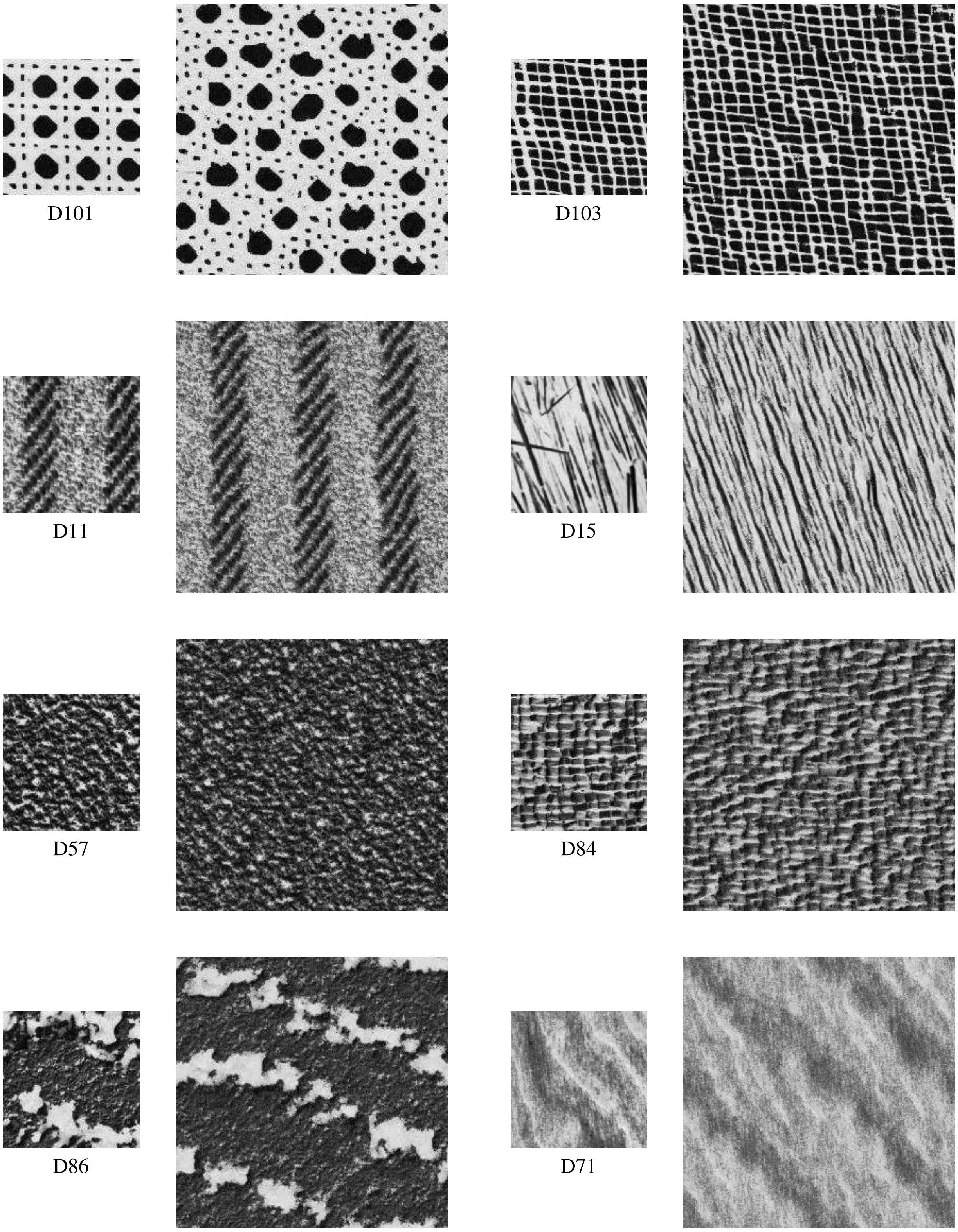
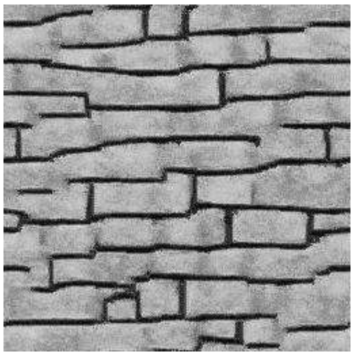
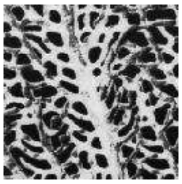


Figure 4: The smaller patches are the original textures with size 128x128 pixels, and to their right are synthesized ones which are 4 times larger. Each original texture patch is labeled with the index in the Brodatz album. The number of basis functions used is 4096. The neighborhood contains the current level and one lower resolution level, with sizes 7x7 and 3x3, respectively.

D95





D87

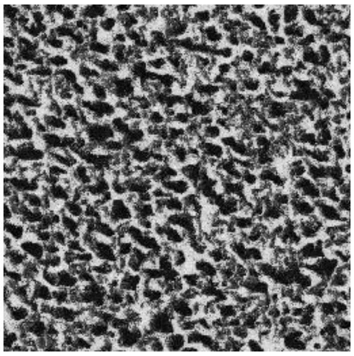


Figure 5: Some synthesis failures.

algorithm with several recent works. The reader is referred to [12], [16], [24], [20], and [22] for more complete surveys of previous texture analysis/synthesis algorithms.

One way to synthesize natural textures is to develop specialized procedures that simulate the underlying physical texture generation process. For example, reaction diffusion ([23], [26]) and cellular texturing ([7], [27]) have been used to generate biological patterns. The advantage of those approaches is that the textures can be developed directly onto the 3D surfaces; thus the distortion problem during texture mapping is avoided. Nevertheless, these algorithms are usually slow, hard to use (require hand-crafted parameters), and only suitable for specific textures (such as wood, marble, or animal skin). In comparison, our algorithm is easy to use and can efficiently generate a broad range of textures.

Instead of adopting directly simulation, statistical algorithms treat textures as realizations of probability distributions, and generate new textures by sampling from such distributions. For example, Markov random field and Gibbs sampling are widely employeed to model textures ([3], [5], [13], [10], [28], [19], [18]). These techniques are either very time consuming, or incapable of generating high quality textures due to the restrictions of the employeed models. To reduce the amount of computation, other algorithms model textures by a set of features, and generate new images by matching these features with an example texture ([14], [4], [21], [9]). The success of these algorithms depends primarily on the kind of features chosen. For example, Heeger and Bergen ([14]) model textures by marginal histograms of the image pyramids. Their technique is capable of generating highly stochastic patterns but fails on more structured textures. De Bonet ([4]) synthesizes new images by randomizing an input texture sample while preserving the cross-scale dependencies (the "parent structure" in his paper). This method works better on structured textures, but can produce visible artifacts if the input texture is not tileable. Simoncelli and Portilla ([21]) generate textures by matching the joint statistics of the image pyramids. It can successfully capture global textural structures but fails to preserve local patterns. Compared with these techniques, our algorithm performs well on both random and structured textures. In addition, it can produce smooth and tileable results even if the input is not tileable (Note that none of the original patches in Figure 4 are tileable).

6 Conclusion

We present a new method for generating high quality texture images from examples. The key advantage of our approach is that it can efficiently synthesize a wide variety of textures. At the same time, it is simple to implement since the most complex component is tree structure VQ. Our algorithm is also easy to use: only an example image along with a few parameters are required to generate a new texture of any size and shape.

Acknowledgement

I would like to thank professors Marc Levoy and Pat Hanrahan from Stanford University, and Kris Popat from Xerox Parc for their helpful comments and discussions. This research is supported by Intel, Interval, and Sony under the Stanford Immersive Television (IMTV) project.

References

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford, 1995.

[2] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, New York, 1966.

[3] G.R. Cross and A.K. Jain. Markov random field texture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(1):25–39, January 1983.

[4] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages

361–368. ACM SIGGRAPH, Addison Wesley, August 1997.

[5] H. Derin and W.S. Cole. Segmentation of textured images using Gibbs random fields. *Computer Vision, Graphics, and Image Processing*, 35(1):72–98, July 1986.

[6] K.B. Eom. 2-D moving average models for texture synthesis and analysis. *IEEE Transactions on Image Processing*, 7(12):1741–1746, December 1998.

[7] Kurt Fleischer, David Laidlaw, Bena Currin, and Alan Barr. Cellular texture generation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 239–248. ACM SIGGRAPH, Addison Wesley, August 1995.

[8] J.M. Francos, A.Z. Meiri, and B. Porat. A unified texture model based on a 2-D wold like decomposition. *IEEE transactions on signal processing*, 41:2665–2678, August 1993.

[9] A. Gagalowicz and S.D. Ma. Sequential synthesis of natural textures. *Computer Vision, Graphics, and Image Processing*, 30:289–315, 1985.

[10] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[11] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[12] R.M. Haralick. Statistical image texture analysis. In *Handbook of Pattern Recognition and Image Processing*, volume 86, pages 247–279. Academic Press, 1986.

[13] M. Hassner and J. Sklansky. The use of Markov random fields as models of texture. *Computer Vision, Graphics, and Image Processing*, 12:357–370, 1980.

[14] David J. Heeger and James R. Bergen. Pyramid-Based texture analysis/synthesis. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995.

[15] T.I. Hsu and R. Wilson. A two-component model of texture for analysis and synthesis. *IEEE Transactions on Image Processing*, 7(10):1466, October 1998.

[16] H. Iversen and T. Lonnestad. An evaluation of stochastic models for analysis and synthesis of gray scale texture. *Pattern Recognition Letters*, 15:575–585, 1994.

[17] D.P. Mital and G.W. Leng. An autoregressive approach to surface texture analysis. *International Journal of Pattern Recognition and Machine Intelligence*, 8:845–857, 1994.

[18] R. Paget and I.D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale Markov random field. *IEEE Transactions on Image Processing*, 7(6):925–931, June 1998.

[19] K. Popat and R.W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Visual Communications and Image Processing*, 1993.

[20] A.R. Rao. *A Taxonomy for Texture Description and Identification*. Springer-Verlag, 1990.

[21] E. Simoncelli and J. Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Fifth International Conference on Image Processing*, October 1998.

[22] G. Smith. *Image Texture Analysis using Zero Crossings Information*. PhD thesis, Department of Computer Science and Electrical Engineering, University of Queensland, St Lucia 4072, Australia, 1998.

[23] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.

[24] L. VanGool, P. Dewaele, and A. Oosterlinck. Texture analysis anno 1983. *Computer Vision, Graphics, and Image Processing*, 29(3):336–357, March 1985.

[25] L.Y. Wei. Texture analysis and synthesis using a deterministic nonlinear characteristic function. In *Second IASTED International Conference on Computer Graphics and Imaging*, October 1999.

[26] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[27] Steven P. Worley. A cellular texture basis function. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996.

[28] S. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximun entropy (FRAME) - towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.