www.icgst.com

A Brush Tool for Interactive Texture Synthesis

Colas Schretter
Université Libre de Bruxelles (ULB),
Boulevard du Triomphe, 1050 Brussels, Belgium.
cschrett@ulb.ac.be

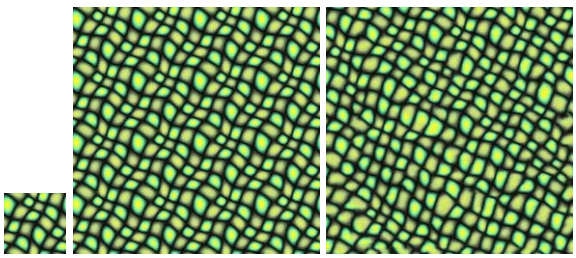
Abstract

We present a user-centered approach to image-based texture synthesis. The synthesis uses a user-defined brush as generation primitive. This allows synthesis of textures at the texel level and further integration of the method into most of the existing digital image edition softwares. We treat texture images as probability density estimators from which new images with perceptually similar appearance and structure can be sampled. We model the input texture sample as a Markov Random Field and propose several algorithms based on the *locality* and *stationarity* principles assumed from the theory.

Keywords: *Markov Random Field, Texture Synthesis, Interactive Painting.*

1 Introduction

The main goal of texture synthesis, shown in Figure 1, is to develop a transformation function $I_o = F(I_i)$, which takes an input texture image I_i and produce an output image I_o, such that their perceived visual difference $D_p(I_i, I_o)$ is above a threshold of visual difference from the original, while matching the same underlying structure.



(a) (b) (c)

Figure 1: Goal of texture synthesis. Starting from a 64×64 example texture (a), our method generated the 256×256 output (c) using a smoothed round brush of 9 pixels diameter. Note that we avoided the verbatim repetition produced by tiling the input pattern (b).

Our aim is to propose an intrinsically interactive and user-centered texture synthesis technique that fulfill the requirements of artists. The main characteristic of a usable texture synthesis technique remain its ability to produce tillable textures from arbitrary inputs and its ability to constrain itself to existing image regions of any shape.

This paper present an algorithm that can be directly linked in most of the image editing framework, as an additional brush tool, similar to the *clone tool*. The user's needs and constraints where our guideline in choices to meet both performances, usability and generality. The user will be able paint custom seamless textures with a simple brush.

This work is based on validated existing texture synthesis techniques. However, we adapted the ideas to allow brush-driven texture generation and interactive performances. We propose a speed-up algorithm that allow fast neighborhood comparison based on early discarding of non-similar sites. Moreover, our perceptual similarity metric take into account an optional *feature map* that provide a measure of the perceptual weight of any *site*.

Users want to edit images in real-time. Hence, methods based on preprocessing of the input sample are not applicable. Our method trades speed with quality by limiting the search region. Thus, the running time becomes independent of the size of the input texture sample, and only dependent on the size of the biggest visual feature. We exploit the principles of *locality* and *stationarity* brought by the Markov Random Field (MRF) theory to justify our algorithms.

1.1 Previous Work

The first synthesis technique proposed in [8] was based on histogram matching the synthesized texture with the input sample at various resolution levels. Later, another multiresolution approach was proposed in [3]. This method synthesized a broad range of unstructured textures.

The latest works in statistical-based texture synthesis use operations on wavelets to transform input samples. [16] provides impressive results at the expense of slow computing time. [2] extended the texture synthesis principle to the generation of tillable Time Varying Textures

animations.

A few MRF-based techniques exist. Their strongest disadvantage is the running time needed to reach an acceptable level of quality. The primary bottleneck to the running time is the great number of nearest neighborhood searches to process.

[15] synthesize textures on massively parallel architecture using a *stochastic relaxation* method [7]. [6] perform an expensive exhaustive search in the input sample to elect each pixel's color. [17] propose to use a Vector Quantization technique to accelerate the nearest neighborhood search. Latest approaches require preprocessing to classify the sites sharing similar neighborhood [12, 18].

Several image-based rendering techniques [5, 11] yield real-time acceleration of the synthesis process by stitching cropped patches sampled in the input texture. A hybrid method [13] combines two approaches by choosing a per-pixel synthesis only if the distances between two overlapping pixels exceed a given threshold.

An attempt to bring some limited interactive control to the synthesis process by extending the Wei & Levoy technique was proposed in [1]. Image analogies [9] allow a kind of interactivity by exploiting a full-screen user-drawn sketch to steer the synthesis process.

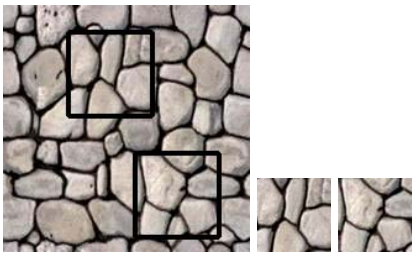
2 Theoretical Considerations

Across the various references in computer graphics, computer vision and data analysis, we did not find a suitable formal definition of digital texture. [10] defines texture as:

Texture The term *texture* generally refers to repetition of basic texture elements called *texels*.

Texel The *texel* contains several *pixels*, whose placement could be periodic, quasi-periodic or random.

Our definition will be based on two key principles of *stationarity* and *locality*, as shown by Figure 3.



(a) (b) (c)

Figure 3: Texture ensuring the *stationarity* and *locality* principles. Two square view windows (b) and (c) are randomly chosen in the input texture (a). The windows are large enough to capture the visual feature of the texture sample. With respect to the *stationarity* principle, (b) and (c) look similar. In addition, by the *locality* principle, each pixel of the texture is only related to a small set of neighboring pixels.

Stationarity Given a finite view window, two different regions of a texture defined by two different location of the view window always look similar.

Locality The color of a pixel is only related to the colors of a finite set of neighboring pixels.

Texture A *texture* is a infinite array of *pixels* that obey the principles of *stationarity* and *locality*.

The former definition allow us to use the Markov Random Field model as a valid statistical representation of textures. This means that we can consider brightness value of a pixel as only being dependent on the brightness values of its direct neighboring pixels.

2.1 Markov Random Field Model

A variable X_s located at a *site* s on a lattice

$$S \equiv \{s = (x,y) : 0 \leq x < N, 0 \leq y < M\}$$

can be equal to any value $x_s \in \Lambda_s$, where the *state space* Λ_s is usually the RGB or greyscale color space. But the probability $P[X_s = x_s]$ only depends on values x_r located in the *neighborhood* of the *site* s.

Neighborhood The *neighboring sites* are defined as those sites $r \in N_s \subset S$ where the *neighborhood* N_s contains usually a constant number of the closest *sites* around s.

Markov Random Field The MRF is defined by the local conditional probability density function (LCPDF) with respect to the *neighborhood system*

$$N : P[x_s | x_r, r \in N_s] \, s \in S, x_s \in \Lambda$$

To model a greyscale digital image I as a MRF, we consider the coordinates of each pixel in I as a *site* s on a lattice S. The luminosity value of the pixel is equal to x_s. If the pixel value is coded with 8 bits then the luminosity value is contained in the *state space* $\Lambda = \{0, 1, \ldots, 255\}$.

2.2 Non-Parametric Model

Explicitly defining the LCPDF using the MRF model and then sampling from it, is a natural application of the theory. However, in practice, we must learn the MRF from realistic and finite inputs. The explicit building of the model can be achieved by common statistical learning techniques such as Gibbs sampling [7]. Unfortunately, Gibbs sampling is notoriously slow, and it's difficult to assess when the process has converged.

In this work, we do not explicitly build the MRF. Instead, we start directly with the input image I_i and search for the most likely neighborhood $I_{best} \subset I_i$, given an arbitrary query image $I_{query} \subset I_o$ where I_o is the current output image. The process is repeated for each query.

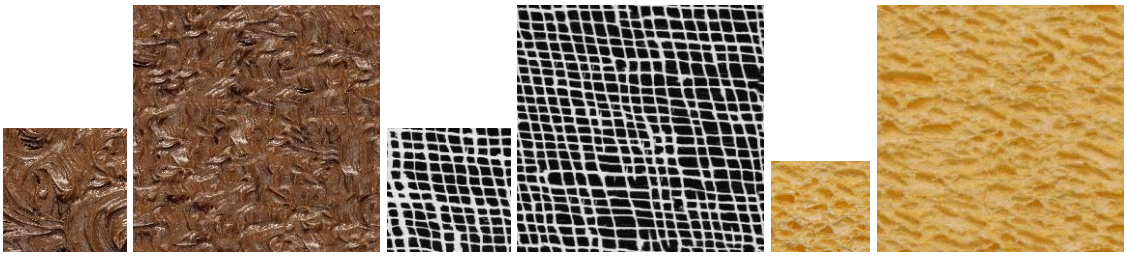


Figure 2: Texture synthesis results. All synthesized outputs are 256×256 tillable images. A round brush of 9 pixels height was used to paint the textures, starting from an empty template.

2.3 Perceptual Similarity Metric

Given two digital images I_1 and I_2 of identical size $N \times M$, it's not easy to quantify their degree of resemblance. Almost all the texture synthesis techniques found in literature use the L_2 norm. While the later metric has mathematical sense, it does not take into account the psycho-visual parameters of human vision. How the brain extracts the main visual features of the image is not a fully understood process. For instance, empirical experiments demonstrate that surface highlights and edges are discriminating features of images.

At the beginning of the process, some pixels of the output image are masked to indicate that they must be synthesized. Often this binary masking is implemented by using a conventional key color. During the texture synthesis, the neighborhood comparisons should only involve valid non-color keyed pixels of the output image I_o. Therefore we will use an approximation of the distance of two neighborhoods by using the causal D_c function.

Let $p_1 = I_1(x,y)$ and $p_2 = I_2(x,y)$ then the expression of D_c can be written as

$$D_c(I_1,I_2) = \frac{\sum_{y<N}\sum_{x<M} K_c(x,y)\, L_{lum}[p_1,p_2]}{\sum_{y<N}\sum_{x<M} K_c(x,y)}$$

where the squared luminosity distance of two pixels p_1 and p_2 is given by:

$$\begin{aligned} L_{lum}(p_1,p_2) &= [0.30\cdot(R(p_1)-R(p_2))]^2 \\ &+ [0.59\cdot(G(p_1)-G(p_2))]^2 \\ &+ [0.11\cdot(B(p_1)-B(p_2))]^2 \end{aligned}$$

The R, G and B functions extract respectively the red, green and blue components of the pixel.

A Gaussian kernel K is used to preserve the local structure of the texture part by increasing the error of nearby pixels from the center of the considered neighborhood. Moreover, we take into account a *feature map* F that contain, for each site of I_1, a scalar weight relative to the perceptual importance of the site. Hence,

$$K_c(x,y) = A(x,y)\,F(x,y)\,K(x,y)$$

with $A(x,y) = 0$ if the corresponding output pixel is equal to the key color and $A(x,y) = 1$ otherwise.

3 Our Method

This section presents, with a bottom-up approach, our synthesis method. The algorithm needs two true color images I_i and I_o. Some of their pixels are masked. The masked sites of the output image I_o will be filled by a sequence of brush strokes.

Each stroke copies a patch found in the input image I_i while using a nearest neighborhood search strategy to elect the best causal patch. The neighborhood comparison used by the search rely on our perceptual similarity presented at the previous section.

3.1 Nearest Neighborhood Search

Our method is adaptive to input region of arbitrary (even non-contiguous) shape. Therefore we will only consider the input sites that allows a brush stroke to transfer only non-masked pixels. We define $S_i \equiv \{s_1, s_2, \ldots, s_k\}$ where the brush, centered on $s_i \in S_i$, only contain sites $s \in S_i$.

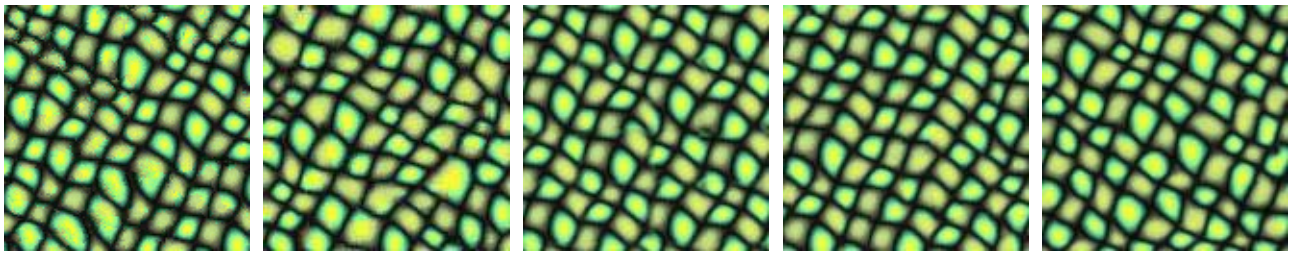
The main bottleneck of non-parametric sampling procedures is the search for the closest neighborhood of a site $s \subset I_o$ into the input domain S_i. The naive exhaustive search blows up the runtime for large input samples.

Several faster methods exist. Searching the closest image patch can be viewed as a nearest neighbor search in high dimensions vectorial spaces [14]. *KD*-tree structures [11] or Vector Quantization techniques [17] could be used to find a close match quickly. Most of the alternative approaches require preprocessing to classify the sites sharing similar neighborhood [12, 18].

However, we could rely once more on the MRF theory to justify a simpler proposal that only require precalculation dependent on the size of the kernel N.

3.1.1 Similarity Evaluation

Before each search, we build a sorted list containing every sites $s \in N$ sorted by their relative potential contribution $K_c(s)$to the total similarity scoring. If $A(s) = 0$, then s is not inserted in the list. Only the input-dependent term $F(s)$ needs to be modulate to the precomputed value $A(s)K(s)$ presents in the list. The firsts sites are likely to contribute greatly to raise the similarity scoring. Therefore, we could stop the evaluation early once the current score exceed a given proportion of the best solution found so far. In prac-



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

Figure 4: Comparison of various texture synthesis algorithms. (a) Efros & Leung. (b) Wei & Levoy. (c) Patch-based Sampling. (d) Hybrid Texture Synthesis. (e) Interactive Texture Synthesis using a round brush of 9 pixels diameter. Every output is a 128×128 image.

tice, early discarding of potential sites improve the search efficiency by a factor.

The associated *feature map* could be arbitrary. We experimented that greyscale maps produced by edges-sensitive filters such as the *glowing edges filter* improved the similarity accuracy.

3.1.2 Moving Window Search

A corollary of the principles of *locality* and *stationarity* is that any neighborhood, large enough to capture the biggest visual feature of the input texture, and centered on a given site $s_i \in S_i$, contains a pair of sites such as their neighborhoods match.

At first glance, we could limit the search area to a window N_r centered around a random pivot point $r \in S_i$. N_r should contain at least as many sites as the size of the perceptual kernel K. Moreover, we use a larger window's size to explore the input domain further at each step.

We propose to *displace* the window's pivot point instead of randomly picking a site at each time step. This step by step coherent approach allows, in practice, to find more suitable neighborhoods. Indeed, the spatial correlation between visual features is taken into account.

If the search do not yields an acceptable similar site, then we re-launch the search with a randomly chosen pivot point.

3.1.3 Introducing Variance

To add a controlled variance to the election of the nearest neighborhood, we retain a small list of the bests sites found by the neighborhood matching procedure. A tolerance parameter $t \geq 0$ is used by the algorithm to control the amount of variance.

Formally, we define a set

$$\Omega_p \equiv \left\{ s_i \in S_i : D_c\left(N_o, N_{s_i}\right) \leq D_c\left(N_o, N_{s_b}\right) + t \right\}$$

where $N_o \subset I_o$ and $N_{s_b} \subset I_i$ is the neighborhood of the site $s_b \in S_i$ that match the best N_o. To elect the best neighborhood, we sample randomly Ω_p.

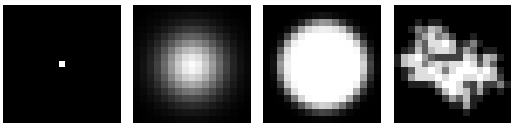
3.2 Brush Stroke

Given a pivot site p_o in the output image and the best neighbor site p_i found in the input texture, the method will synthesize several pixels in a row. A transparency value is associated to each site offset $(o_1, o_2, \ldots, o_k)$ of the brush shape. For each output site $s_j = (p_o + o_j)\ j \in [1 \ldots k]$ covered by the brush, we will compute the new pixel's color resulting from a weighted blending dependent on the transparencies values associated with the sites p_i, s_j and brush offset o_j.

Let $a_j \in [0 \ldots 1]$ and $b_j \in [0 \ldots 1]$ the respective transparency values associated with o_j and s_j. Let $t_j \in [0 \ldots 1]$ the brush value of o_j. Then the new output pixel color becomes

$$s'_j = \frac{I_i(p_i + o_j)\, a_j t_j + I_o(p_o + o_j)\, b_j (1 - t_j)}{b_j + t_j (a_j - b_j)}$$

Note that each color and transparency channels of the output pixel will be updated with the same formula. The weighted pixel blending operation smooth the transitions between patches by feathering. The resulting color could alter the original texture histogram, hence altering the original structure of the input texture. However this pixel blending approach has become a standard in most digital painting software, thus corresponds to the user's intuition and expected result.



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

Figure 5: Various brushes. (a) Single pixel pen. (b) Smooth disc of 13 pixel diameter. (c) Hard disc of 13 pixels diameter. (d) Texture splatch.

The brushes can have any shape and size as shown on Figure 5. By using a pencil brush, the stroke operation have similar behavior than the pixel-based methods described in [6, 17].

4 Applications and Results

The texture synthesis technique described in this work was primarily intended to replace the common *clone tool* present in major pixel based image editing software. To use the *clone tool*, the user defines a constant offset by picking a coordinate in a texture region of the edited image. Then, the brush strokes copy the pixels values located at the corresponding offset coordinate. However, this technique does not guarantee proper edge handling and does not constrain the synthesis to current output. Seams and other visual artifacts appear. Moreover, the pivot point must be redefined very often in practice. Using the *clone tool* seems to requires too much skills of the user.

To use the proposed technique, the user simply needs to *(i)* Indicate the texture pixels, for instance by masking a portion of the image. *(ii)* Define the size of the neighborhood K. *(iii)* Define the brush shape. *(iv)* Paint a region of the image.

By repeating these steps, texture mixtures, image inpainting, image extrapolation (Figure 6) and foreground removal (Figure 7) become natural operations.

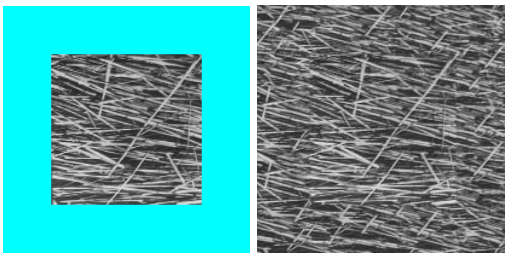


Figure 6: Image extrapolation. Only the border of the input template was filled in place with the non-tillable center considered as input sample.

Figure 4 shows comparative results with previous techniques on the most common *green gradient* benchmark texture. The various images are chosen as the best representative quality provided by each technique, regardless of the required computing time, except for our result which ensures interactive performance on a workstation.

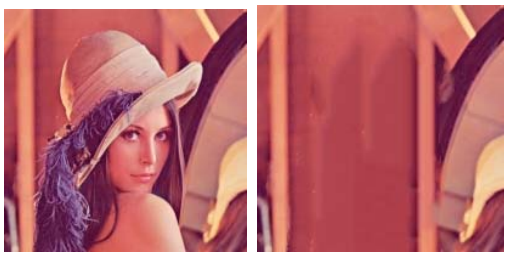


Figure 7: Ever wondered what is hidden behind Lena? Starting from the original image, we painted a rough mask to indicate where Lena is lying on the image. Then we inpainted the masked region while using the image itself as input example.

We tested our algorithm with success against various representative photo-realistic texture samples of the Brodatz catalog [4], some De Bonet's images [3] and other photo-realistic inputs were tested (Figure 2).

Highly structured textures are difficult to synthesize with believable results as shown in Figure 8. This behavior is unfortunately intrinsic to the MRF model of the texture surface.

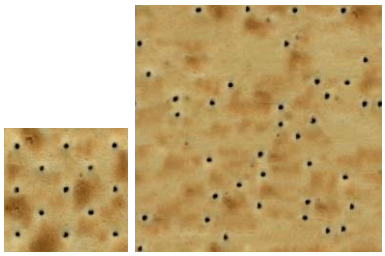


Figure 8: Synthesis failure. The MRF model perceptual metric did not emphasize the regularity of little holes positions. Therefore, this main structural feature of the input texture is not preserved.

To generate the wood texture shown by Figure 9, we used a search window as big as the input source. Then, only the perceptual similarity metric was tested.

Figure 9: A difficult case. With a neighborhoods of 128 pixels, our perceptual metric failed to capture the laminated structure of the wood. Hence the result loose some structural information. However, it is disputable that the input sample exhibits the properties of texture.

Beside the cases where the technique shows its limits, results are encouraging. Most of result images seem to bring more variance in comparison with patch-based methods and less visual artifacts than pixel-based methods.

5 Conclusion

We have implemented a method to produce variations and extrapolation of example texture images. Our approach is based on the principles of *locality* and *stationarity* brought by the Markov Random Field theory. Under these two fundamental hypotheses, we have proposed algorithms for nearest neighborhood search and texture synthesis of a brush stroke. The proposed perceptual similarity metric is adapted to our specific application and based on the use of radial neighborhoods weighted by Gaussian kernels and an optional *feature map*.

The algorithm only needs a few intuitive parameters to work and was intended to be used by average users. Experimental results shows that the output result is sufficiently different from the original, and appears to have been created by the same underlying generative process.

The brush stroke concept allows the integration of the texture synthesis process with blending or feathering and the consideration of boundary conditions. It gives the user control over the synthesis process, because bad results can quickly be overdrawn.

The method respects the basic framework of modern digital image editors and provides a reliable generalization of the *clone tool* at the expense of more computing resources.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *ACM Symposium on Interactive 3D Graphics*, volume 2, pages 217–226. ACM, 2001.

[2] Z. Bar-Joseph. Statistical learning of multidimensional textures. Master's thesis, Hebrew University of Jerusalem, 1999.

[3] J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 361–368. ACM, ACM Press / ACM SIGGRAPH, 1997.

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

[5] A. Efros and W.T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 341–346. ACM, ACM Press / ACM SIGGRAPH, 2001.

[6] A. Efros and T. Leung. Texture synthesis by nonparametric sampling. In *International Conference on Computer Vision*, volume 2, pages 1033–8. IEEE, 1999.

[7] S. Geman and D. Geman. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[8] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 229–238. ACM, ACM Press / ACM SIGGRAPH, 1995.

[9] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques*, pages 327–340. ACM Press, 2001.

[10] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, New Jersey, 1989.

[11] L. Liang, C. Liu, Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. Technical report, Microsoft Research, 2001.

[12] S. Magda and D. Kriegman. Fast texture synthesis on arbitrary meshes. In *Eurographics Workshop on Rendering*, pages 82–89. Eurogrpahics Association, 2003.

[13] A. Naelen and M. Alexa. Hybrid texture synthesis. In *Eurographics Symposium on Rendering*, pages 217–226, 2003.

[14] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 19(1):989–1003, 1997.

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

[16] E. Simoncelli and J. Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Fifth International Conference on Image Processing*, pages 62–66. IEEE, 1998.

[17] L. Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 479–488. ACM, ACM Press / ACM SIGGRAPH, 2000.

[18] S. Zelinka and M. Garland. Interactive texture synthesis on surfaces using jump maps. In *Eurographics Symposium on Rendering*, pages 90–96, 2003.