Pyramid-Based Texture Analysis/Synthesis

David J. Heeger*
Stanford University

James R. Bergen†
SRI David Sarnoff Research Center

Abstract

This paper describes a method for synthesizing images that match the texture appearance of a given digitized sample. This synthesis is completely automatic and requires only the "target" texture as input. It allows generation of as much texture as desired so that any object can be covered. It can be used to produce solid textures for creating textured 3-d objects without the distortions inherent in texture mapping. It can also be used to synthesize texture mixtures, images that look a bit like each of several digitized samples. The approach is based on a model of human texture perception, and has potential to be a practically useful tool for graphics applications.

1 Introduction

Computer renderings of objects with surface texture are more interesting and realistic than those without texture. Texture mapping [15] is a technique for adding the appearance of surface detail by wrapping or projecting a digitized texture image onto a surface. Digitized textures can be obtained from a variety of sources, e.g., cropped from a photoCD image, but the resulting texture chip may not have the desired size or shape. To cover a large object you may need to repeat the texture; this can lead to unacceptable artifacts either in the form of visible seams, visible repetition, or both.

Texture mapping suffers from an additional fundamental problem: often there is no natural map from the (planar) texture image to the geometry/topology of the surface, so the texture may be distorted unnaturally when mapped. There are some partial solutions to this distortion problem [15] but there is no universal solution for mapping an image onto an arbitrarily shaped surface.

An alternative to texture mapping is to create (paint) textures by hand directly onto the 3-d surface model [14], but this process is both very labor intensive and requires considerable artistic skill.

Another alternative is to use computer-synthesized textures so that as much texture can be generated as needed. Furthermore, some of the synthesis techniques produce textures that tile seamlessly.

Using synthetic textures, the distortion problem has been solved in two different ways. First, some techniques work by synthesizing texture directly on the object surface (e.g., [31]). The second solution is to use *solid textures* [19, 23, 24]. A solid texture is a 3-d array of color values. A point on the surface of an object is colored by the value of the solid texture at the corresponding 3-d point. Solid texturing can be a very natural solution to the distortion problem: there is no distortion because there is no mapping. However, existing techniques for synthesizing solid textures can be quite cumbersome. One must learn how to tweak the parameters or procedures of the texture synthesizer to get a desired effect.

This paper presents a technique for synthesizing an image (or solid texture) that matches the appearance of a given texture sample. The key advantage of this technique is that it works entirely from the example texture, requiring no additional information or adjustment. The technique starts with a digitized image and analyzes it to compute a number of texture parameter values. Those parameter values are then used to synthesize a new image (of any size) that looks (in its color and texture properties) like the original. The analysis phase is inherently two-dimensional since the input digitized images are 2-d. The synthesis phase, however, may be either two- or three-dimensional. For the 3-d case, the output is a solid texture such that planar slices through the solid look like the original scanned image. In either case, the (2-d or 3-d) texture is synthesized so that it tiles seamlessly.

2 Texture Models

Textures have often been classified into two categories, deterministic textures and stochastic textures. A deterministic texture is characterized by a set of primitives and a placement rule (e.g., a tile floor). A stochastic texture, on the other hand, does not have easily identifiable primitives (e.g., granite, bark, sand). Many real-world textures have some mixture of these two characteristics (e.g. woven fabric, woodgrain, plowed fields).

Much of the previous work on texture analysis and synthesis can be classified according to what type of texture model was used. Some of the successful texture models include reaction-diffusion [31, 34], frequency domain [17], fractal [9, 18], and statistical/random field [1, 6, 8, 10, 12, 13, 21, 26] models. Some (e.g., [10]) have used hybrid models that include a deterministic (or periodic) component and a stochastic component. In spite of all this work, scanned images and hand-drawn textures are still the principle source of texture maps in computer graphics.

This paper focuses on the synthesis of stochastic textures. Our approach is motivated by research on human texture perception. Current theories of texture discrimination are based on the fact that two textures are often difficult to discriminate when they produce a similar distribution of responses in a bank of (orientation and spatial-frequency selective) linear filters [2, 3, 7, 16, 20, 32]. The method described here, therefore, synthesizes textures by matching distributions (or histograms) of filter outputs. This approach depends on the principle (not entirely correct as we shall see) that all of the spatial information characterizing a texture image can be captured in the first order statistics of an appropriately chosen set of linear filter outputs. Nevertheless, this model (though incomplete) captures an interesting set of texture properties.

*Department of Psychology, Stanford University, Stanford, CA 94305. heeger@white.stanford.edu http://white.stanford.edu

†SRI David Sarnoff Research Center, Princeton, NJ 08544. jrb@sarnoff.com

Computational efficiency is one of the advantages of this approach compared with many of the previous texture analysis/synthesis systems. The algorithm involves a sequence of simple image processing operations: convolution, subsampling, upsampling, histograming, and nonlinear transformations using small lookup tables. These operations are fast, simple to implement, and amenable to special purpose hardware implementations (e.g., using DSP chips).

3 Pyramid Texture Matching

The pyramid-based texture analysis/synthesis technique starts with an input (digitized) texture image and a noise image (typically uniform white noise). The algorithm modifies the noise to make it look like the input texture (figures 2, 3, 4). It does this by making use of an invertible image representation known as an *image pyramid*, along with a function, `match-histogram`, that matches the histograms of two images. We will present examples using two types of pyramids: the Laplacian pyramid (a radially symmetric transform) and the steerable pyramid (an oriented transform).

3.1 Image Pyramids

A linear image transform represents an image as a weighted sum of *basis* functions. That is, the image, $I(x,y)$, is represented as a sum over an indexed collection of functions, $g_i(x,y)$:

$$I(x,y) = \sum_i y_i g_i(x,y),$$

where y_i are the transform coefficients. These coefficients are computed from the signal by projecting onto a set of *projection* functions, $h_i(x,y)$:

$$y_i = \sum_{x,y} h_i(x,y) I(x,y).$$

For example, the basis functions of the Fourier transform are sinusoids and cosinusoids of various spatial frequencies. The projection functions of the Fourier transform are also (co-)sinusoids.

In many image processing applications, an image is decomposed into a set of subbands, and the information withing each subband is processed more or less independently of that in the other subbands. The subbands are computed by convolving the image with a bank of linear filters. Each of the projection functions is a translated (or shifted) copy of one of the convolution kernels (see [28] for an introduction to subband transforms and image pyramids).

An *image pyramid* is a particular type of subband transform. The defining characteristic of an image pyramid is that the basis/projection functions are translated and dilated copies of one another (translated and dilated by a factor or 2^j for some integer j). The subbands are computed by convolving and subsampling. For each successive value of j, the subsampling factor is increased by a factor of 2. This yields a set of subband images of different sizes (hence the name image pyramid) that correspond to different frequency bands.

In an independent context, mathematicians developed a form of continuous function representation called *wavelets* (see [30] for an introduction to wavelets), that are very closely related to image pyramids. Both wavelets and pyramids can be implemented in an efficient recursive manner, as described next.

Laplacian Pyramid. The Laplacian pyramid [4, 5, 22] is computed using two basic operations: `reduce` and `expand`. The `reduce` operation applies a low-pass filter and then subsamples by a factor of two in each dimension. The `expand` operation upsamples by a factor of two (padding with zeros in between pixels) and then applies the same low-pass filter. A commonly used low-pass filter kernel (applied separably to the rows and columns of an image) is: $\frac{1}{16}(1,4,6,4,1)$.

One complete level of the pyramid consists of two images, l^0 (a low-pass image), and b^0 (a high-pass image), that are computed as follows:

```
l0 = Reduce(im)
b0 = im - Expand(l0),
```

where `im` is the original input image. Note that the original image can be trivially reconstructed from l^0 and b^0:

```
reconstructed-im = b0 + Expand(l0).
```

The next level of the pyramid is constructed by applying the same set of operations to the l^0 image, yielding two new images, l^1 and b^1. The full pyramid is constructed (via the `make-pyramid` function) by successively splitting the low-pass image l^i into two new images, l^{i+1} (a new low-pass image) and b^{i+1} (a new band-pass image).

The combined effect of the recursive low-pass filtering and sub/upsampling operations yields a subband transform whose basis functions are (approximately) Gaussian functions. In other words, the transform represents an image as a sum of shifted, scaled, and dilated (approximately) Gaussian functions. The projection functions of this transform are (approximately) Laplacian-of-Gaussian (mexican-hat) functions, hence the name Laplacian pyramid. Note that the pyramid is *not* computed by convolving the image directly with the projection functions. The recursive application of the `reduce` and `expand` operations yields the same result, but much more efficiently.

In the end, we get a collection of pyramid subband images consisting of several bandpass images and one leftover lowpass image. These images have different sizes because of the subsampling operations; the smaller images correspond to the lower spatial frequency bands (coarser scales). Note that the original image can always be recovered from the pyramid representation (via the `collapse-pyramid` function) by inverting the sequence of operations, as exemplified above.

Steerable Pyramid. Textures that have oriented or elongated structures are not captured by the Laplacian pyramid analysis because its basis functions are (approximately) radially symmetric.

To synthesize anisotropic textures, we adopt the steerable pyramid transform [25, 29]. Like the Laplacian pyramid, this transform decomposes the image into several spatial frequency bands. In addition, it further divides each frequency band into a set of orientation bands.

The steerable pyramid was used to create all of the images in this paper. The Laplacian pyramid was used (in addition to the steerable pyramid, see Section 4) for synthesizing the solid textures shown in figure 5.

Figure 1(a) shows the analysis/synthesis representation of the steerable pyramid transform. The left-hand side of the diagram is the analysis part (`make-pyramid`) and the right hand side is the synthesis part (`collapse-pyramid`). The circles in between represent the decomposed subband images. The transform begins with a high-pass/low-pass split using a low-pass filter with a radially symmetric frequency response; the high-pass band corresponds to the four corners of the spatial frequency domain. Each successive level of the pyramid is constructed from the previous level's low-pass band by a applying a bank of band-pass filters and a low-pass filter.

The orientation decomposition at each level of the pyramid is "steerable" [11], that is, the response of a filter tuned to any orientation can be obtained through a linear combination of the responses

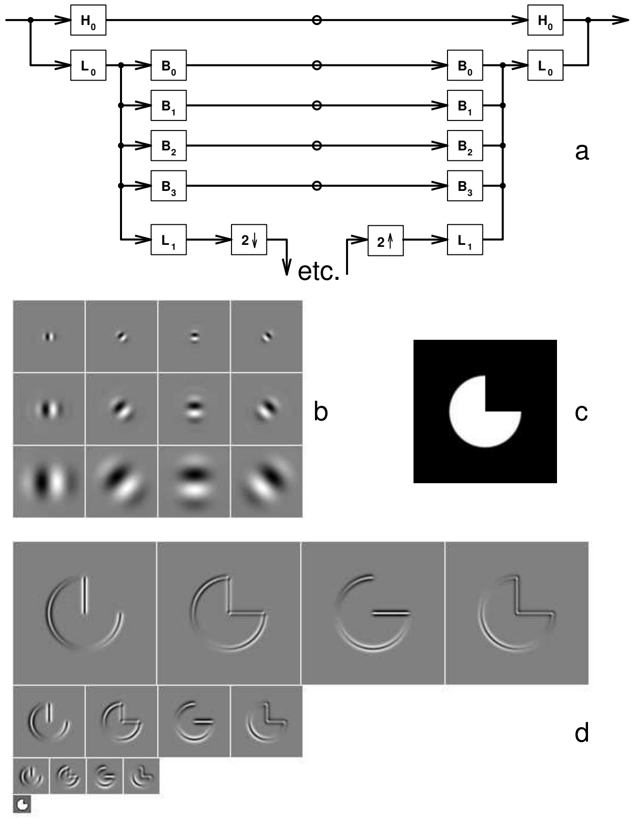

Figure 1: (a) System diagram for the first level of the steerable pyramid. Boxes represent filtering and subsampling operations: H_0 is a high-pass filter, L_0 and L_i are low-pass filters, and B_i are oriented bandpass filters. Circles in the middle represent the decomposed subbands. Successive levels of the pyramid are computed by applying the B_i and L_1 filtering and subsampling operations recursively (represented by "etc." at the bottom). (b) Several basis/projection functions of the steerable pyramid. Note that these are *not* the B_i filters, although the B_i filters do look similar to the top row of basis/projection functions. (c) Input image. (d) Steerable pyramid subband images for this input image.

of the four basis filters computed at the same location. The steerability property is important because it implies that the pyramid representation is locally rotation-invariant.

The steerable pyramid, unlike most discrete wavelet transforms used in image compression algorithms, is non-orthogonal and overcomplete; the number of pixels in the pyramid is much greater than the number of pixels in the input image (note that only the low-pass band is subsampled). This is done to minimize the amount of aliasing within each subband. Avoiding aliasing is critical because the pyramid-based texture analysis/synthesis algorithm treats each subband independently.

The steerable pyramid is self-inverting; the filters on the synthesis side of the system diagram are the same as those on the analysis side of the diagram. This allows the reconstruction (synthesis side) to be efficiently computed despite the non-orthogonality.

Although the steerable pyramid filter kernels are nonseparable, any nonseparable filter can be approximated (often quite well) by a sum of several separable filter kernels [25]. Using these separable filter approximations would further increase the computational efficiency.

A C code implementation of the steerable pyramid is available at `http://www.cis.upenn.edu/~eero/home.html`.

Psychophysical and physiological experiments suggest that image information is represented in visual cortex by orientation and spatial-frequency selective filters. The steerable pyramid captures some of the oriented structure of images similar to the way this information is represented in the human visual system. Thus, textures synthesized with the steerable pyramid look noticeably better than those synthesized with the Laplacian pyramid or some other non-oriented representation. Other than the choice of pyramid, the algorithm is exactly the same.

3.2 Histogram Matching

Histogram matching is a generalization of histogram equalization. The algorithm takes an input image and coerces it via a pair of lookup tables to have a particular histogram. The two lookup tables are: (1) the cumulative distribution function (cdf) of one image, and (2) the inverse cumulative distribution function (inverse cdf) of the other image. An image's histogram is computed by choosing a binsize (we typically use 256 bins), counting the number of pixels that fall into each bin, and dividing by the total number of pixels. An image's cdf is computed from its histogram simply by accumulating successive bin counts.

The cdf is a lookup table that maps from the interval [0,256] to the interval [0,1]. The inverse cdf is a lookup table that maps back from [0,1] to [0,256]. It is constructed by resampling (with linear interpolation) the cdf so that its samples are evenly spaced on the [0,1] interval.

These two lookup tables are used by the `match-histogram` function to modify an image (`im1`) to have the same histogram as another image (`im2`):

```
Match-histogram (im1,im2)
  im1-cdf = Make-cdf(im1)
  im2-cdf = Make-cdf(im2)
  inv-im2-cdf = Make-inverse-lookup-table(im2-cdf)
  Loop for each pixel do
       im1[pixel] =
         Lookup(inv-im2-cdf,
                Lookup(im1-cdf,im1[pixel]))
```

3.3 Texture Matching

The `match-texture` function modifies an input `noise` image so that it looks like an input `texture` image. First, match the histogram of the noise image to the input texture. Second, make pyramids from both the (modified) noise and texture images. Third, loop through the two pyramid data structures and match the histograms of each of the corresponding pyramid subbands. Fourth, collapse the (histogram-matched) noise pyramid to generate a preliminary version of the synthetic texture. Matching the histograms of the pyramid subbands modifies the histogram of the collapsed image. In order to get *both* the pixel and pyramid histograms to match we iterate, rematching the histograms of the images, and then rematching the histograms of the pyramid subbands.

```
Match-texture(noise,texture)
  Match-Histogram (noise,texture)
  analysis-pyr = Make-Pyramid (texture)
  Loop for several iterations do
     synthesis-pyr = Make-Pyramid (noise)
     Loop for a-band in subbands of analysis-pyr
          for s-band in subbands of synthesis-pyr
          do
          Match-Histogram (s-band,a-band)
     noise = Collapse-Pyramid (synthesis-pyr)
     Match-Histogram (noise,texture)
```

Whenever an iterative scheme of this sort is used there is a concern about convergence. In the current case we have not formally investigated the convergence properties of the iteration, but our experience is that it always converges. However, stopping the algorithm after several (5 or so) iterations is critical. As is the case with nearly all discrete filters, there are tradeoffs in the design of the steerable pyramid filters (e.g., filter size versus reconstruction accuracy). Since the filters are not perfect, iterating too many times introduces artifacts due to reconstruction error.

The core of the algorithm is histogram matching which is a spatially local operation. How does this spatially local operation reproduce the spatial characteristics of textures? The primary reason is that histogram matching is done on a representation that has intrinsic spatial structure. A *local* modification of a value in one of the pyramid subbands produces a *spatially correlated* change in the reconstructed image. In other words, matching the *pointwise* statistics of the pyramid representation does match some of the *spatial* statistics of the reconstructed image. Clearly, only spatial relationships that are represented by the pyramid basis functions can be captured in this way so the choice of basis functions is critical. As mentioned above, the steerable pyramid basis functions are a reasonably good model of the human visual system's image representation.

If we had a complete model of human texture perception then we could presumably synthesize perfect texture matches. By analogy, our understanding of the wavelength encoding of light in the retina allows us to match the color appearance of (nearly) any color image with only three colored lights (e.g., using an RGB monitor). Lights can be distinguished only if their spectral compositions differ in such a way as to produce distinct responses in the three photoreceptor classes. Likewise, textures can be distinguished only if their spatial structures differ in such a way as to produce distinct responses in the human visual system.

3.4 Edge Handling

Proper edge handling in the convolution operations is important. For the synthesis pyramid, use circular convolution. In other words, for an image $I(x, y)$ of size NxN, define: $I(x, y) \equiv I(x \bmod N,\ y \bmod N)$. Given that the synthesis starts with a random noise image, circular convolution guarantees that the resulting synthetic texture will tile seamlessly.

For the analysis pyramid, on the other hand, circular convolution would typically result in spuriously large filter responses at the image borders. This would, in turn, introduce artifacts in the synthesized texture. A reasonable border handler for the analysis pyramid is to pad the image with a reflected copy of itself. Reflecting at the border usually avoids spurious responses (except for obliquely oriented textures).

3.5 Color

The RGB components of a typical texture image are *not* independent of one another. Simply applying the algorithm to R, G, and B separately would yield color artifacts in the synthesized texture.

Instead, color textures are analyzed by first transforming the RGB values into a different color space. The basic algorithm is applied to each transformed color band independently producing three synthetic textures. These three textures are then transformed back into the RGB color space giving the final synthetic color texture.

The color-space transformation must be chosen to decorrelate the color bands of the input texture image. This transformation is computed from the input image in two steps. The first step is to subtract the mean color from each pixel. That is, subtract the average of the red values from the red value at each pixel, and likewise for the green and blue bands. The resulting color values can be plotted as points in a three-dimensional color space. The resulting 3-d cloud of points is typically elongated in some direction, but the elongated direction is typically not aligned with the axes of the color space.

The second step in the decorrelating color transform rotates the cloud so that its principle axes align with the axes of the new color space. The transform can be expressed as a matrix multiplication, $\mathbf{y} = \mathbf{M}\mathbf{x}$, where $\mathbf{x}$ is the RGB color (after subtracting the mean) of a particular pixel, $\mathbf{y}$ is the transformed color, and $\mathbf{M}$ is a 3x3 matrix.

The decorrelating transform $\mathbf{M}$ is computed from the covariance matrix $\mathbf{C}$ using the singular-value-decomposition (SVD). Let $\mathbf{D}$ be a 3xN matrix whose columns are the (mean-subtracted) RGB values of each pixel. The covariance matrix is: $\mathbf{C} = \mathbf{D}\mathbf{D}^t$, where $\mathbf{D}^t$ means the transpose of $\mathbf{D}$. The SVD algorithm algorithm decomposes the covariance matrix into the product of three components, $\mathbf{C} = \mathbf{U}\mathbf{S}^2\mathbf{U}^t$. Here, $\mathbf{U}$ is an orthonormal matrix and $\mathbf{S}^2$ is a diagonal matrix. These matrices ($\mathbf{C}$, $\mathbf{U}$ and $\mathbf{S}^2$) are each 3x3, so the SVD can be computed quickly. The decorrelating transform is: $\mathbf{M} = \mathbf{S}^{-1}\mathbf{U}^t$, where $\mathbf{S}$ is a diagonal matrix obtained by taking the square-root of the elements of $\mathbf{S}^2$.

After applying this color transform, the covariance of the transformed color values is the identity matrix. Note that the transformed color values are: $\mathbf{M}\mathbf{D} = \mathbf{S}^{-1}\mathbf{U}^t\mathbf{U}\mathbf{S}\mathbf{V}^t = \mathbf{V}^t$. It follows that the covariance of the transformed color values is: $\mathbf{V}^t\mathbf{V} = \mathbf{I}$.

The color transform is inverted after synthesizing the three texture images in the transformed color space. First, multiply the synthetic texture's color values at each pixel by $\mathbf{M}^{-1}$. This produces three new images (color bands) transformed back into the (mean subtracted) RGB color space. Then, add the corresponding mean values (the means that were subtracted from the original input texture) to each of these color bands.

4 Solid Textures

Pyramid-based texture analysis/synthesis can also be used to make isotropic 3-d solid textures. We start with an input image and a block of 3-d noise. The algorithm coerces the noise so that any slice through the block looks like the input image.

The solid texture synthesis algorithm is identical to that described above, except for the choice of pyramid: use a 2-d Laplacian pyramid for analysis and a 3-d Laplacian pyramid for synthesis. As usual, match the histograms of the corresponding subbands. Note that since the Laplacian pyramid is constructed using separable convolutions, it extends trivially to three-dimensions.

We have obtained better looking results using a combination of Laplacian and steerable pyramids. On the analysis side, construct a 2-d Laplacian pyramid and a 2-d steerable pyramid. On the synthesis side, construct a 3-d Laplacian pyramid and construct steerable pyramids from all two-dimensional (x-y, x-z, and y-z) slices of the solid. Match the histograms of the 3-d (synthesis) Laplacian pyramid to the corresponding histograms of the 2-d (analysis) Laplacian pyramid. Match the histograms of each of the many synthesis steerable pyramids to the corresponding histograms of the analysis steerable pyramid. Collapsing the synthesis pyramids gives four solids (one from the 3-d Laplacian pyramid and one from each set of steerable pyramids) that are averaged together. Some examples are shown in figure 5.

5 Texture Mixtures

Figure 6 shows some *texture mixtures* that were synthesized by choosing the color palette (decorrelating color transform) from one image and the pattern (pyramid subband statistics) from a second image.

One can imagine a number of other ways to mix/combine textures to synthesize an image that looks a bit like each of the inputs: apply `match-texture` to a second image rather than noise, combine the high frequencies of one texture with the low frequencies of another, combine two or more textures by averaging their pyramid histograms, etc.

6 Limitations and Extensions

The approach presented in this paper, like other texture synthesis techniques, has its limitations. The analysis captures some but not all of the perceptually relevant structure of natural textures. Hence, this approach should be considered one of many tools for texturing objects in computer graphics.

It is critical that the input image be a *homogeneous* texture. Figure 7 shows two input textures (cropped from different areas of the same photoCD image) and two corresponding synthetic textures. When the input is inhomogeneous (due to an intensity gradient, contrast gradient, perspective distortion, etc.) then the synthesized texture has a blotchy appearance.

The approach also fails on quasi-periodic textures and on random mosaic textures (figure 8). Although the results look interesting, they do not particularly resemble the inputs. We have had some success synthesizing quasi-periodic textures using a hybrid scheme (e.g., like [10]) that combines a periodic texture model with the pyramid decomposition. Methods that are specifically designed to capture long range statistical correlation [26] have also been successful with textures of this type. The issue with random mosaic textures is mainly one of scale. If the repeated micro-patterns are small enough, then the pyramid analysis/synthesis scheme works well (e.g., see the ivy example in figure 3).

Figure 9 shows more examples of failures. There are two aspects of these images that the pyramid texture model misses. First, these textures are locally oriented but the dominant orientation is different in different parts of the image. In a sense, they are inhomogeneous with respect to orientation. Second, they contain extended, fine structure (correlations of high frequency content over large distances). The pyramid scheme captures correlations of low frequency content over large distances, but it captures correlations of high frequency content only over very short distances.

There is no general way to construct an anisotropic solid texture from a 2-d sample. However, there are several options including: (1) constructing a solid texture as the outer product of a 2-d anisotropic color texture image and a 1-d (monochrome) signal; (2) composing (adding, multiplying, etc.) several solid textures as Peachy [23] did; (3) starting with an isotropic solid, and introducing anisotropy procedurally, like Perlin's marble [24] and Lewis' woodgrain [19]; (4) starting with an isotropic solid, and using a paint program to introduce anisotropic "touch-ups".

Image pyramids and multi-scale image representations of one sort or another are the most often used data structures for antialiased texture mapping (e.g., Renderman, Silicon Graphics Iris GL, General Electric and E&S realtime flight simulators, and reference [33]). Pyramid-based texture synthesis, therefore, can be naturally integrated into an antialiased texture mapping system.

Finally, it may be possible to write an interactive tool for texture synthesis, with a slider for each parameter in the pyramid representation. In our current implementation, each subband histogram is encoded with 256 bins. However the subband histograms of many "natural" images have a characteristic shape [27], suggesting that a very small number of parameters may be sufficient.

7 Conclusion

This paper presents a technique for created a two- or three-dimensional (solid) texture array that looks like a digitized texture image. The advantage of this approach is its simplicity; you do not have to be an artist and you do not have to understand a complex texture synthesis model/procedure. You just crop a textured region from a digitized image and run a program to produce as much of that texture as you want.

Acknowledgements: The teapot images were rendered using Rayshade. Many of the source texture images were cropped from photoCDs distributed by Pixar and Corel. Special thanks to Eero Simoncelli for designing the filters for the steerable pyramid, to Patrick Teo for writing a solid texturing extension to Rayshade, to Alex Sherstinsky for suggesting the solid texturing application, to Marc Levoy for his help and encouragement, and to Charlie Chubb and Mike Landy for stimulating discussions. Supported by an NIMH grant (MH50228), an NSF grant (IRI9320017), and an Alfred P. Sloan Research Fellowship to DJH.

References

[1] BENNIS, C., AND GAGALOWICZ, A. 2-D Macroscopic Texture Synthesis. *Computer Graphics Forum 8* (1989), 291–300.

[2] BERGEN, J. R. Theories of Visual Texture Perception. In *Spatial Vision*, D. Regan, Ed. CRC Press, 1991, pp. 114–133.

[3] BERGEN, J. R., AND ADELSON, E. H. Early Vision and Texture Perception. *Nature 333* (1988), 363–367.

[4] BURT, P. Fast Filter Transforms for Image Processing. *Computer Graphics and Image Processing 16* (1981), 20–51.

[5] BURT, P. J., AND ADELSON, E. H. A Multiresolution Spline with Application to Image Mosaics. *ACM Transactions on Graphics 2* (1983), 217–236.

[6] CHELLAPPA, R., AND KASHYAP, R. L. Texture Synthesis Using 2-D Noncausal Autoregressive Models. *IEEE Transactions on Acoustics, Speech, and Signal Processing 33* (1985), 194–203.

[7] CHUBB, C., AND LANDY, M. S. Orthogonal Distribution Analysis: A New Approach to the Study of Texture Perception. In *Computational Models of Visual Processing*, M. S. Landy and J. A. Movshon, Eds. MIT Press, Cambridge, MA, 1991, pp. 291–301.

[8] CROSS, G. C., AND JAIN, A. K. Markov Random Field Texture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 5* (1983), 25–39.

[9] FOURNIER, A., FUSSEL, D., AND CARPENTER, L. Computer Rendering of Stochastic Models. *Communications of the ACM 25* (1982), 371–384.

[10] FRANCOS, J. M., MEIRI, A. Z., AND PORAT, B. A Unified Texture Model Based on a 2D Wold-Like Decomposition. *IEEE Transactions on Signal Processing 41* (1993), 2665–2678.

[11] FREEMAN, W. T., AND ADELSON, E. H. The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13* (1991), 891–906.

[12] GAGALOWICZ, A. Texture Modelling Applications. *The Visual Computer 3* (1987), 186–200.

[13] GAGALOWICZ, A., AND MA, S. D. Sequential Synthesis of Natural Textures. *Computer Vision, Graphics, and Image Processing 30* (1985), 289–315.

[14] HANRAHAN, P., AND HAEBERLI, P. Direct WYSIWYG Painting and Texturing of 3D Shapes. Proceedings of SIGGRAPH 90. In *Computer Graphics* (1990), vol. 24, ACM SIGGRAPH, pp. 215–223.

[15] HECKBERT, P. S. Survey of Texture Mapping. *IEEE Computer Graphics and Applications 6* (1986), 56–67.

[16] LANDY, M. S., AND BERGEN, J. R. Texture Segregation and Orientation Gradient. *Vision Research 31* (1991), 679–691.

[17] LEWIS, J. P. Texture Synthesis for Digital Painting. Proceedings of SIGGRAPH 84. In *Computer Graphics* (1984), vol. 18, ACM SIGGRAPH, pp. 245–252.

[18] LEWIS, J. P. Generalized Stochastic Subdivision. *ACM Transactions on Graphics 6* (1987), 167–190.

[19] LEWIS, J. P. Algorithms for Solid Noise Synthesis. Proceedings of SIGGRAPH 89. In *Computer Graphics* (1989), vol. 23, ACM SIGGRAPH, pp. 263–270.

[20] MALIK, J., AND PERONA, P. Preattentive Texture Discrimination with Early Vision Mechanisms. *Journal of the Optical Society of America A 7* (1990), 923–931.

[21] MALZBENDER, T., AND SPACH, S. A Context Sensitive Texture Nib. In *Communicating with Virtual Worlds*, N. M. Thalmann and D. Thalmann, Eds. Springer-Verlag, New York, 1993, pp. 151–163.

[22] OGDEN, J. M., ADELSON, E. H., BERGEN, J. R., AND BURT, P. J. Pyramid-Based Computer Graphics. *RCA Engineer 30* (1985), 4–15.

[23] PEACHY, D. R. Solid Texturing of Complex Surfaces. Proceedings of SIGGRAPH 85. In *Computer Graphics* (1985), vol. 19, ACM SIGGRAPH, pp. 279–286.

[24] PERLIN, K. An Image Synthesizer. Proceedings of SIGGRAPH 85. In *Computer Graphics* (1985), vol. 19, ACM SIGGRAPH, pp. 287–296.

[25] PERONA, P. Deformable Kernels for Early Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1995). To appear May 1995.

[26] POPAT, K., AND PICARD, R. W. Novel Cluster-Based Probability Model for Texture Synthesis, Classification, and Compression. In *Proceedings of SPIE Visual Communications and Image Processing* (1993), pp. 756–768.

[27] RUDERMAN, D. L., AND BIALEK, W. Statistics of Natural Images: Scaling in the Woods. *Physical Review Letters 73* (1994), 814–817.

[28] SIMONCELLI, E. P., AND ADELSON, E. H. Subband Transforms. In *Subband Image Coding*, J. W. Woods, Ed. Kluwer Academic Publishers, Norwell, MA, 1990.

[29] SIMONCELLI, E. P., FREEMAN, W. T., ADELSON, E. H., AND HEEGER, D. J. Shiftable Multi-Scale Transforms. *IEEE Transactions on Information Theory, Special Issue on Wavelets 38* (1992), 587–607.

[30] STRANG, G. Wavelets and Dilation Equations: A Brief Introduction. *SIAM Review 31* (1989), 614–627.

[31] TURK, G. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. Proceedings of SIGGRAPH 91. In *Computer Graphics* (1991), vol. 25, ACM SIGGRAPH, pp. 289–298.

[32] TURNER, M. R. Texture Discrimination by Gabor Functions. *Biological Cybernetics 55* (1986), 71–82.

[33] WILLIAMS, L. Pyramidal Parametrics. Proceedings of SIGGRAPH 83. In *Computer Graphics* (1983), vol. 17, ACM SIGGRAPH, pp. 1–11.

[34] WITKIN, A., AND KASS, M. Reaction-Diffusion Textures. Proceedings of SIGGRAPH 91. In *Computer Graphics* (1991), vol. 25, ACM SIGGRAPH, pp. 299–308.

Figure 2:

Figure 3:

Figure 4:

Figure 5:

Figure 6:

Figure 7:

Figure 8:

Figure 9: