

**An Implementation of Heeger and Bergen's Texture
Analysis/Synthesis Algorithm**

by

Thomas F. El-Maraghi

Department of Computer Science

University of Toronto

Toronto, Ontario

2 September 1997

Copyright © Thomas F. El-Maraghi, 1997

1. Introduction

In their paper entitled “Pyramid-Based Texture Analysis/Synthesis” [4], Heeger and Bergen present an algorithm for generating synthetic copies of stochastic textures. Based on a self-inverting oriented bandpass pyramid and a histogram matching technique, the process is tantalizing both for the results it produces and for its intrinsic simplicity. Given a texture sample, either grayscale or color, the algorithm can synthesize as much of it as desired.

Here, we present an implementation and discussion of Heeger and Bergen’s algorithm. The focus is on the use of the algorithm for texture generation (i.e., as opposed to texture analysis). An attempt is made to elaborate on the key components of the design, especially in areas covered briefly in [4]. In particular, details are provided about the steerable pyramid, and the iterative histogram matching that forms the core of the process. Also presented are a number of textures generated by the algorithm, including several produced from the samples used in [4] for comparison.

2. Heeger and Bergen's Algorithm

The algorithm described by Heeger and Bergen [4] generates stochastic textures from target samples. Unlike deterministic textures, such as regular polka dots or bricks, stochastic textures are not formed from obvious primitives. Although many textures are neither strictly stochastic nor deterministic, a purely stochastic texture model can still capture many interesting qualities of a sample. In practice, Heeger and Bergen's algorithm does a good job of capturing and reproducing a broad range of textures.

The texture-matching algorithm relies on two main components: a self-inverting steerable pyramid, and a histogram matching process. The next sections give a brief overview of Heeger and Bergen's texture matching algorithm. Emphasis is placed on details covered briefly in [4].

2.1 Overview

The input to Heeger and Bergen's algorithm is a target texture sample, and the output is a synthetically generated image that matches the appearance of that sample. The following is pseudo-code for the algorithm:

```
MakeUniformNoise( noise )
MatchHistogram( target, noise )
ExpandPyramid( target, t-pyramid )
for several iterations
  ExpanadPyramid( noise, n-pyramid )
  for all pyramid subbands
    MatchHistogram( t-pyramid-subband, n-pyramid-subband )
  CollapsePyramid( n-pyramid, noise )
  MatchHistogram( target, noise )
```

The process begins by generating a sample of uniform random noise. The noise image, which has the dimensions of the desired output image, is modified by so that its histogram matches that of the target texture (the histogram matching process is discussed in Section 2.3). The texture sample is decomposed into an oriented bandpass pyramid (discussed in Section 2.2). Next, the pyramid decomposition is applied to the modified noise image and the histogram matching process is employed between corresponding subbands of the two pyramids. Following subband matching, the noise pyramid is collapsed and the histogram matching process is again applied to the new noise image and the texture sample. The decompose-match-collapse-match process is iterated several

times. After a few iterations, the output image—which began as random noise—begins to resemble the target texture.

2.2 The Steerable Pyramid

An image pyramid is a transformation that takes an image and decomposes it into a set of subbands, typically through a recursive set of convolution and subsampling operations. The image is usually subsampled by a factor of 2 between each pyramid level, and thus each level corresponds to twice the wavelength of its predecessor. Of interest here is a particular type of pyramid that is known as “self-inverting”. A self-inverting pyramid has two useful properties: (1) it provides a complete representation of an image, and (2) the original image can be reconstructed by applying the same filters used to generate the pyramid.

In addition to being self-inverting, the pyramid used by Heeger and Bergen [4] is steerable [5]. Each level of a steerable pyramid consists of a set of oriented basis subbands generated from rotated copies of a single filter kernel. The response of the filter at any desired orientation can be obtained from the basis subbands by applying a “steering” function [1][5][6]. Since the pyramid is self-inverting and the subbands are oriented, it can be thought of as a form of non-orthogonal wavelet transform [5].

As described in [4], a steerable pyramid can capture the variation of a texture in both intensity and orientation. Combined with the histogram matching process discussed in the next section, this representation can be used to synthesis copies of textures. Before investigating that process, however, it is instructional to examine the steerable pyramid and more closely.

In the Fourier domain, the steerable pyramid decomposes an image into polar-separable frequency subbands. Figure 2.1 depicts the idealized frequency response of a steerable pyramid with 4 orientations. The annular regions cut by radial spokes represent the portions of the 2-dimensional frequency space isolated by each of the oriented bandpass filters. In addition to the oriented subbands, the pyramid contains residual highpass and lowpass bands (see below). Thus, for a steerable pyramid with 4 orientations, a set of 7 filters are required: the 4 oriented bandpass filters B_i , a lowpass filter used for subsampling L_1 , and the initial highpass and lowpass filters H_0 and L_0 . Typically, $L_0(\omega)$ is chosen to be $L_1(\omega/2)$ [5]. It is important to ensure that the responses

of H_0 and L_0 filters together provide a complete and representation of the image with no aliasing, and the same is true for the L_1 and B_i filters. These requirements are illustrated in Figure 2.2, and the system diagram for the steerable pyramid is shown in Figure 2.3.

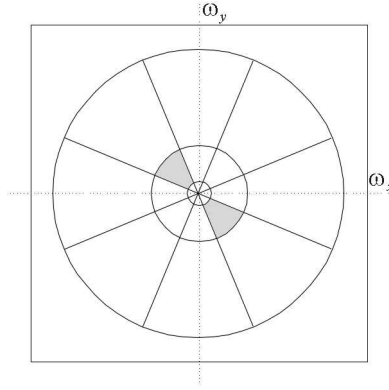


Figure 2.1: Idealized decomposition of 2-dimensional frequency space by the bandpass filters of a steerable pyramid with 4 orientations. The axes extend from $-\pi$ to π . Circles towards the center of the diagram represent lower frequencies, while those toward the outside represent higher frequencies. The shaded areas represent the region of support for a single subband tuned to 135° .

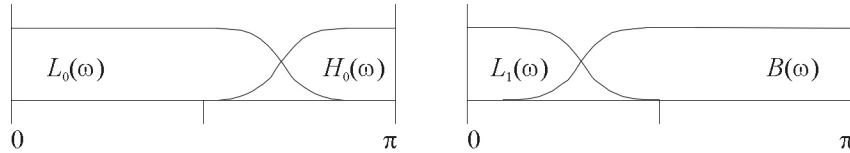


Figure 2.2: Idealized frequency responses of the $L_0(\omega)$ and $H_0(\omega)$ filters (left), and of the $L_1(\omega)$ and $B(\omega)$ filters (right).

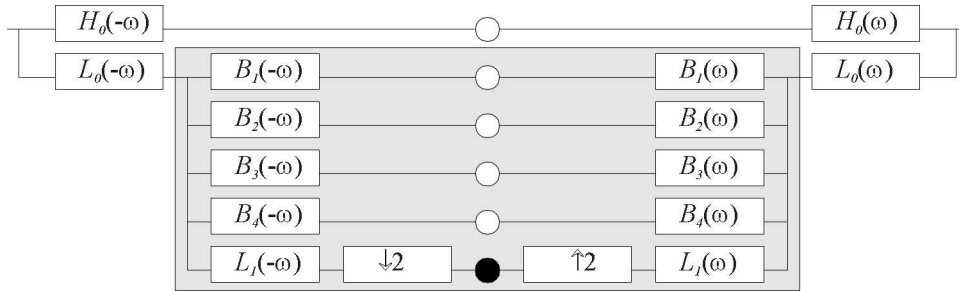


Figure 2.3: System diagram for the steerable pyramid representation of an image. The original image is applied at the left-hand side, and the circles in the center of the figure represent the pyramid itself. Starting at the left side of the diagram, the image is split into lowpass and highpass portions by applying the $L_0(-\omega)$ and $H_0(-\omega)$ filters, respectively. The oriented subband filters $B_i(-\omega)$, and the second lowpass filter $L_1(-\omega)$, are applied to the output of $L_0(-\omega)$. Finally, the output of $L_1(-\omega)$ is down-sampled by a factor of 2. Higher levels of the pyramid are generated by inserting a copy of the shaded rectangle in the position of the solid circle. The pyramid is inverted by applying the filters (flipped in both x and y) and adding the results.

The design of the bandpass filter is best described in the Fourier domain using polar coordinates [5]:

$$B_i(\vec{\omega}) = A(\theta - \theta_i)B(\omega) \quad (2.1)$$

where $\theta = \tan^{-1}\left(\frac{\omega_y}{\omega_x}\right)$, $\theta_i = \frac{2\pi}{k}$, $\omega = |\vec{\omega}|$, and k is the number of orientations. The

desired derivative order determines the form of $A(\theta)$, the angular portion of (2.1). When transformed into the Fourier domain, a directional derivative corresponds to multiplication by a linear ramp function [5]:

$$-j\omega_x = -j\omega \cos(\theta) \quad (2.2)$$

where the derivative operator is in the x direction. The relevant portion of (2.2) is $\cos(\theta)$, as the imaginary constant and the factor of ω can be absorbed into $B(\omega)$. In the Fourier domain, higher-order derivatives correspond to multiplication by powers of $j\omega$, thus the angular portion of the filter is $\cos(\theta)^N$ [5]. The radial portion of (2.1), $B(\omega)$, is constrained by the need to prevent aliasing during subsampling operations, and the desire to build the pyramid recursively [5][6]. The initial highpass and lowpass filters, H_0 and L_0 , are required to prepare the image for the recursion.

The constraints on the design of the steerable pyramid filters can be summarized as follows [5][6]:

1. *Bandlimiting (to prevent aliasing in the subsampling operation):*

$$L_1(\omega) = 0 \quad \text{for } |\omega| > \frac{\pi}{2} \quad (2.3)$$

2. *Flat System Response:*

$$|H_0(\omega)|^2 + |L_0(\omega)|^2 \left[|L_1(\omega)|^2 + |B(\omega)|^2 \right] = 1 \quad (2.4)$$

3. *Recursion:*

$$\left| L_1\left(\frac{\omega}{2}\right) \right|^2 = \left| L_1\left(\frac{\omega}{2}\right) \right|^2 \left[|L_1(\omega)|^2 + |B(\omega)|^2 \right] \quad (2.5)$$

Simoncelli and Freeman [5] have designed filters meeting these constraints using weighted least-squares techniques in the Fourier domain. They are the filters used in [4], and in the implementation of Heeger and Begen's algorithm described in this paper. The impulse responses of the pyramid filters are shown in Figure 2.4. Figure 2.5 shows the pyramid decomposition of a sample image.

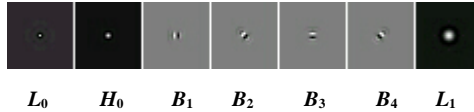


Figure 2.4: Impulse responses of the filters used to generate the 4-orientation steerable pyramid used in the implementation of Heeger and Bergen’s algorithm described in this paper.

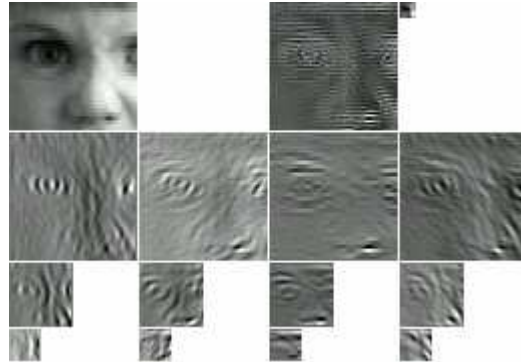


Figure 2.5: A 4-orientation steerable pyramid decomposition of the image of a face. The original image is shown at the top left. The residual highpass and residual lowpass images are shown at the top right. The bottom three rows show the oriented subbands.

2.3 Histogram Matching

The second major component of Heeger and Bergen’s texture analysis/synthesis algorithm is based on histogram matching, which is a generalization of histogram equalization [4]. The idea is to take an input (uniform noise) image and modify it so that its histogram matches the histogram of a target texture. The matching procedure is applied between corresponding subbands of pyramid decompositions of the input and target images. Over several iterations, this coerces the noise image into a synthetic copy of the target texture. Even though histogram matching itself is a local point-wise operation, the pyramid decomposition adds the spatial correlation necessary to capture the qualities of many stochastic textures. That is, a local change in one of the subbands of the pyramid corresponds to a spatially correlated change when the pyramid is collapsed to form a new image.

The following code fragment summarizes the histogram matching procedure:

```
MatchHistogram( target, image )
  MakeCDF( image, image-cdf )
  MakeCDF( target, target-cdf )
  MakeInverseCDF( target-cdf, target-inv-cdf )
  for each pixel in image do
    image[pixel] = target-inv-cdf( image-cdf(pixel) )
```

The procedure relies on a pair of lookup tables: the cumulative distribution function (cdf) of the noise image, and the inverse cumulative distribution function (inverse cdf) of the target image [4]. The cdf is a lookup table formed by accumulating the bin counts of the histogram of an image and normalizing by the total number of pixels. Thus, the cdf maps from pixel values to the range [0,1]. Conversely, an inverse cdf is a lookup table that maps [0,1] back to pixel values. The inverse cdf of an image can be formed from its cdf by resampling (using linear interpolation) it so that the samples are evenly spaced on [0,1]. Thus, the process of matching an image's histogram with that of a target image involves:

- looking up each pixel in the cdf of the image to obtain values from 0 to 1,
- looking up these values in the inverse cdf of the target image, and finally,
- replacing the image pixel values with the target pixel values obtained from the inverse cdf.

Note that this description of the histogram matching process differs from [4] in an effort to emphasize that the values in the noise and target images need not lie in the range [0,256].

2.4 Edge Handling and Color

In an implementation of Heeger and Bergen's algorithm, it is important to use appropriate edge handling techniques when constructing the image pyramids. As noted in [4], circular edge handling (i.e., treating the image as a circular buffer) should be used when constructing and collapsing the noise pyramid. Doing so allows the generated texture to be tiled without obvious discontinuities. However, Heeger and Bergen [4] have found that using a circular edge handler when expanding the analysis pyramid produces spuriously large filter responses at the borders of the image. Thus, to avoid artifacts in the synthesized texture they suggest the use of an edge handler that pads the exterior of the image with reflected copies of itself.

Another detail to consider when implementing Heeger and Bergen's algorithm is the appropriate handling of color. Since the RGB components of a typical image are not independent, one cannot directly apply the algorithm to each channel individually. Instead, it is necessary to transform the color space of the input texture before processing

so that the channels are independent and can be processed separately. After processing, the channels are transformed back into RGB values, yielding the synthesized texture.

The technique used by Heeger and Bergen [4] is essentially Principal Components Analysis. First, one forms the $3 \times N$ matrix \mathbf{D} containing the RGB values of the image. One then applies singular value decomposition (SVD) to covariance matrix $\mathbf{C} = \mathbf{D}\mathbf{D}^T$, obtaining the decomposition $\mathbf{C} = \mathbf{U}\mathbf{S}^2\mathbf{U}^T$ (note that the SVD is usually written as $\mathbf{M} = \mathbf{U}\mathbf{S}^2\mathbf{V}^T$, but $\mathbf{U} = \mathbf{V}$ in the case because \mathbf{C} is symmetric). The decorrelating transform is $\mathbf{y} = \mathbf{S}^{-1}\mathbf{U}^T\mathbf{x} = \mathbf{M}\mathbf{x}$, where \mathbf{x} is the original RGB vector and \mathbf{y} is the transformed color vector. The transform is inverted by multiplying by \mathbf{M}^{-1} . The process is efficient because the covariance and transformation matrices are both 3×3 .

3. Implementation

We have implemented Heeger and Bergen’s texture analysis/synthesis algorithm in C++ using the Template Image Processing System (TIPS). TIPS is a new library, based on C++ templates, designed for image processing and vision research applications. TIPS was designed to be compatible with both the 32-bit Windows platform and Unix¹.

The texture synthesis program is capable of generating both grayscale and RGB textures, and accepts input in the form of either PGM or PPM files. It allows the number of orientations in the pyramid to be selected at runtime. Pyramids with 1, 2, and 4 orientations are supported (Heeger and Bergen used 4 orientations in [4]). The actual filter coefficients were taken from the steerable pyramid described in [5] with permission from the author.

The executable program is called `texture`. The command line is as follows:

```
texture input_file output_file [options]
```

The `input_file` may be either PGM or PPM. The `output_file` parameter specifies the name of the output file, will also be either PGM or PPM by default. The following options are available:

Option	Flag	Default
Number of histogram bins. Should be set to 256 for 8-bit color channels.	-b	256
Number of iterations. Too many iterations can lead to artifacts in the synthesized texture due to reconstruction error in the pyramid.	-i	5
Number of color channels in the texture (should be 1 or 3 to allow for PGM/PPM output). The program will apply the decorrelating transform if there is more than one channel.	-c	3
Number of levels for analysis/synthesis pyramid. Due to the size of the filters, the number of levels should be no more than 3 for a texture of size 128x128.	-l	3
Number of orientations for analysis/synthesis pyramid. Must be 1, 2, or 4.	-o	4
Output file type. If set to anything other than “PGM/PPM”, output will be in the native format of the TIPS library which requires the TIPS viewer.	-t	PGM/PPM
Size of the sample to extract from input file (i.e., $N \times N$). Must be evenly divisible by 2 at least L times, where L is the number of pyramid levels.	-ss	128
Size of synthesized texture (i.e., $N \times N$). Must be evenly divisible by 2 at least L times, where L is the number of pyramid levels.	-st	128

¹ Currently, TIPS has been tested only on Sun systems.

No space should be left between the option prefix and its value. The following is an example command line to generate a 256x256 pixel texture from a 200x200 pixel PPM sample called `gravel` using a 2 orientation pyramid:

```
texture gravel gravel-text -o2 -ss200 -st256
```

Source code for `texture` is available at <http://www.cs.toronto.edu/~tem>. Processing time is approximately 90 seconds for a 128x128-pixel RGB texture when the program is run on a 133 MHz Pentium system with 32 MB RAM.

4. Results

In [4], Heeger and Bergen present a selection of textures generated by their algorithm. Figure 4.1 shows a number of the same textures as generated by our implementation to allow for comparison. Figure 4.2 shows several textures (obtained from the Corel Draw 6.0 texture library) successfully synthesized by the program, and Figure 4.3 shows some failures (also obtained from the Corel Draw library). All of the synthesized images in Figures 4.1-4.3 were generated using 5 iterations of Heeger and Bergen's algorithm and a 4-orientation pyramid.

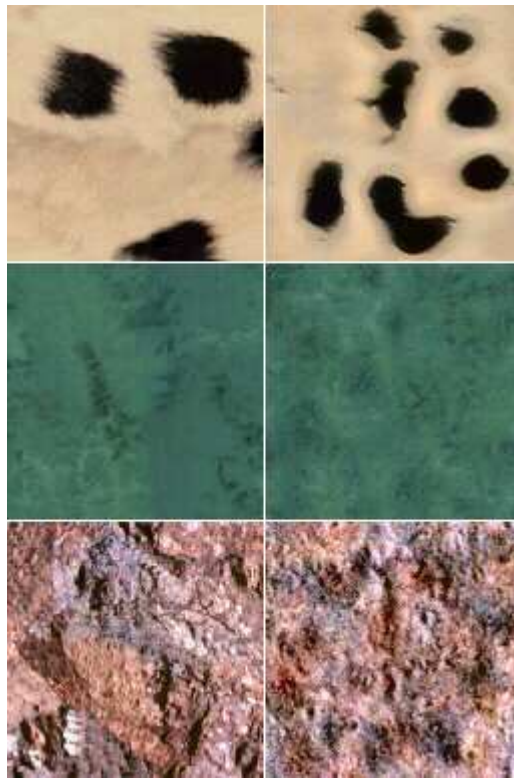


Figure 4.1: Results from our implementation of Heeger and Bergen's algorithm applied to textures extracted from the original paper (for comparison with [4]). The left-hand column shows the original textures (i.e., not synthetic), and the right-hand column shows the textures generated by our implementation. The textures are panda fur (top), marble (middle), and slag stone (bottom). The colors may vary slightly from the original paper because the images were re-scaled.

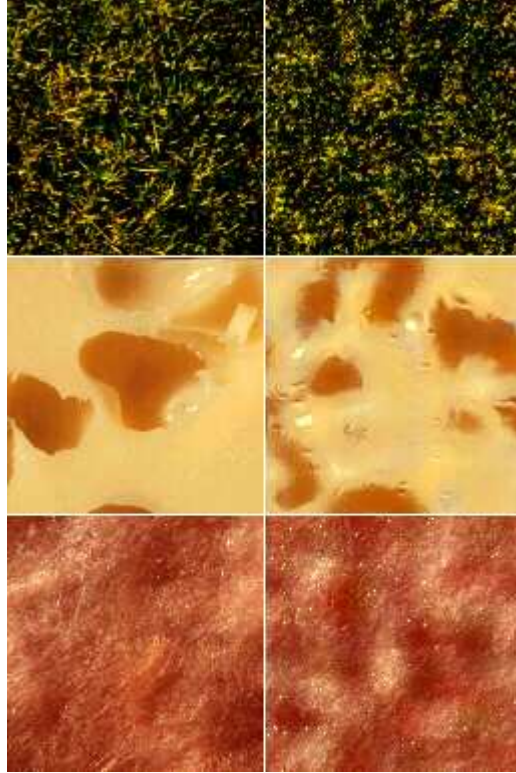


Figure 4.2: Some examples of successfully generated synthetic textures. Grass (top), cheese (middle), and fiberglass (bottom).

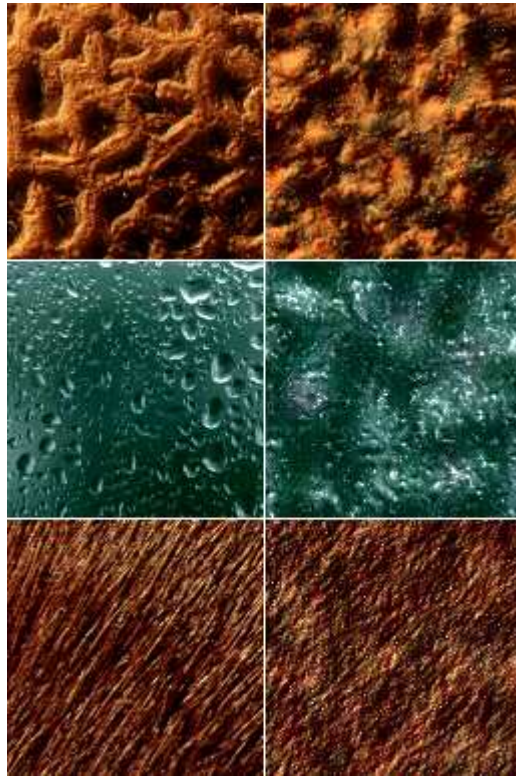


Figure 4.3: Some failures. Cantaloupe (top), raindrops on glass (middle), and fur (bottom).

Figure 4.4 shows the effects of changing number of oriented subbands used in the pyramid. The input sample, fur, is a texture containing spatial correlations that the algorithm is not able to reproduce. However, the synthesized texture looks more like the original sample when 4 orientations are used than it does with either 1 or 2 orientations.



Figure 4.4: The effect of changing the number of oriented subbands in the pyramid on the synthesized texture. The original sample (fur) is at the left-hand side, with the textures produced with 1, 2, and 4 orientations in order beside it. Although the algorithm is not capable of reproducing this sample, the synthesized texture looks more like the original as the number of orientations is increased.

Figure 4.5 shows the emergence of a piece of synthetic panda fur through 10 iterations of Heeger and Bergen’s algorithm. The process appears to converge very quickly. Although the pixel values continue to change by small amounts, a human observer would find it difficult—or even impossible—to discern any change after the 4th iteration. Although it is not a problem with the panda fur texture, allowing the algorithm to run for too many iterations can lead to artifacts due to the slight reconstruction error introduced by the pyramid.

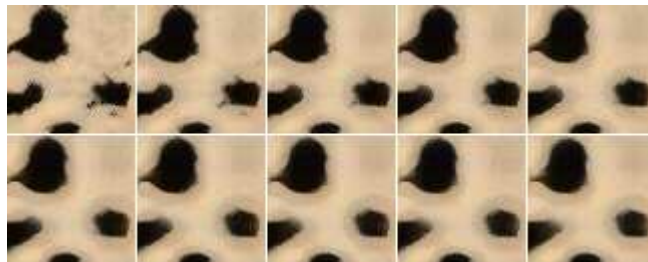


Figure 4.5: The emergence of synthetic panda fur over the first 10 iterations of Heeger and Bergen’s algorithm. The process converges (to within the tolerance of the human visual system) very quickly; it is difficult to discern any change after the 4th iteration.

In [4], Heeger and Bergen mention several ways their algorithm could be used to combine the properties of multiple textures. One of these methods is to average the histograms of multiple images. The simplest way to accomplish this is to form a composite image from two or more textures, which can then be used as input to the algorithm. One way to form the composite image is to tile the original texture images. However, this may lead to the introduction of artifacts due to the seams between the two textures. To avoid this, one could use the multiresolution spline technique described

in [2] to join the texture images seamlessly. Figure 4.6 shows two texture mixtures produced using the simple tiling technique.

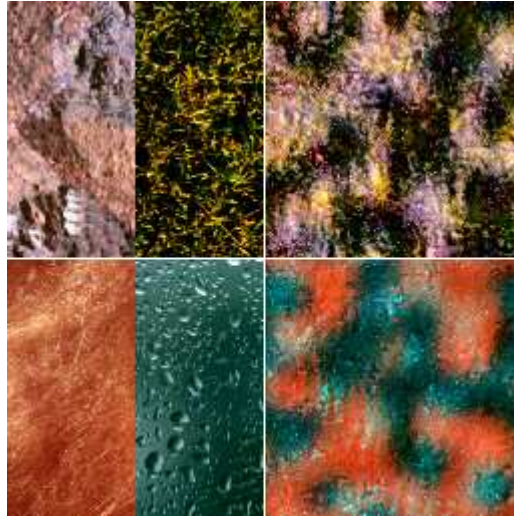


Figure 4.6: Texture mixtures created by using images formed by tiling different textures as input to Heeger and Bergen’s algorithm. The original tiled images are shown in the left-hand column, and the resulting synthetic textures are shown in the right-hand column. The mixtures are slag stone and grass (top), and fiberglass and mist (bottom).

One way to add spatial correlation to the textures produced by Heeger and Bergen’s algorithm is to modulate the noise image from which they are generated. By doing so, one influences the position in the histogram to which a particular pixel is matched. Although this technique provides only limited control over the generated texture, it can produce interesting results. Figure 4.7 shows two textures produced by modulating the noise image with an interference pattern formed by multiplying together vertical and horizontal sinusoidal gratings. In related work, De Bonet [1] has developed a multiresolution technique capable of synthesizing textures with higher levels of spatial correlation than is possible using Heeger and Bergen’s algorithm.

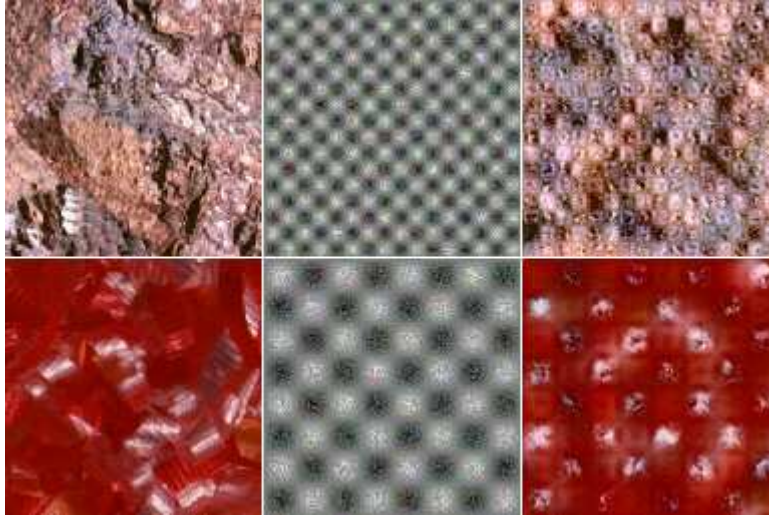


Figure 4.7: Textures produced by modulating the input noise image an interference pattern before applying Heeger and Bergen's algorithm. The original images (slag stone and ribbon) are in the left-hand column. The modulated noise images are in the center column, and the right-hand column shows the resulting synthetic textures.

5. Conclusion

An implementation of Heeger and Bergen’s texture analysis/synthesis algorithm was presented, and the key components of its design—the steerable pyramid and the histogram matching process—were discussed. We presented several textures generated by our implementation, including several produced from the same input textures used in [4] for comparison purposes. A simple technique for creating texture mixtures using input images consisting of multiple tiled images was tried. Finally, the effect of modulating the input noise on the generated texture was investigated.

5.1 Acknowledgements

We thank David Heeger for his comments on his paper. We also thank Eero Simoncelli for allowing us to use the steerable pyramid filters described in [5].

6. References

- [1] J. S. De Bonet, Multiresolution Sampling Procedures for Analysis and Synthesis of Texture Images, *Proceedings of SIGGRAPH '97*, pp. 361-368, 1997.
- [2] P. J. Burt and E. H. Adelson, A Multiresolution Spline With Application to Image Mosaics, *ACM Transactions on Graphics*, Vol. 2 No. 4, pp. 217-236, October 1993.
- [3] W. T. Freeman and E. H. Adelson, The Design and Use of Steerable Filters, *PAMI*, Vol. 13 No. 9, pp. 891-906, September 1991.
- [4] D. J. Heeger and J. R. Bergen, Pyramid-Based Texture Analysis/Synthesis, *Proceedings of SIGGRAPH '95*, pp. 229-238, 1995.
- [5] E. P. Simoncelli and W. T. Freeman, The Steerable Pyramid: A Flexible Architecture for Multi-scale Derivative Computation, *2nd Annual IEEE International Conference on Image Processing*, Washington, DC, October 1995.
- [6] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, Shiftable Multi-scale Transforms, *IEEE Trans. Information Theory*, Vol. 38(2), pp. 587-607, March 1992.