1-1-2000

Fast algorithms for histogram matching: Application to texture synthesis

J. P. Rolland
University of Central Florida

V. Vo
University of Central Florida

B. Bloss
University of Central Florida

C. K. Abbey

Journal of Electronic Imaging 9(1), 39–45 (January 2000).

Fast algorithms for histogram matching: Application to texture synthesis

J. P. Rolland
University of Central Florida
School of Optics and CREOL
School of Electrical Engineering and Computer Science
Orlando, Florida 32816
E-mail: rolland@creol.ucf.edu

V. Vo
B. Bloss
University of Central Florida
School of Electrical Engineering and Computer Science
Orlando, Florida 32816

C. K. Abbey*
University of Arizona
Program in Applied Mathematics
Tucson, Arizona 85724

Abstract. *Texture synthesis is the ability to create ensembles of images of similar structures from sample textures that have been photographed. The method we employ for texture synthesis is based on histogram matching of images at multiple scales and orientations. This paper reports two fast and in one case simple algorithms for histogram matching. We show that the sort-matching and the optimal cumulative distribution function (CDF)-matching (OCM) algorithms provide high computational speed compared to that provided by the conventional approach. The sort-matching algorithm also provides exact histogram matching. Results of texture synthesis using either method show no subjective perceptual differences. The sort-matching algorithm is attractive because of its simplicity and speed, however as the size of the image increases, the OCM algorithm may be preferred for optimal computational speed.* [S1017-9909(00)00601-2]

1 Introduction

Texture synthesis is the ability to create ensembles of images, that look visually similar in structure yet differ pixel to pixel, from sample textures that have been photographed. An important common application of texture synthesis is real-time computer graphics where the objective is to generate textures ''on the fly'' to simulate realistic scenes,[1–4] without the artifacts created from texture maps.[5] Procedural techniques have been developed for real-time texture synthesis.[6] Such approaches, however, are not necessarily optimal for natural textures.

We instead propose to use an approach that employs multiscale decomposition and filtering of both a texture sample and a realization of white noise image for each synthesis. The utilization of multiple realizations of white noise images allows photorealistic generation of ensembles of statistical textures. Textures such as marble, grass, and sand have been synthesized,[7,8] and we have extended the method to include synthesis of medical textures.[9]

Texture synthesis may indeed play an important role in the assessment of image quality in medical imaging, where quality is defined in relation to medical tasks efficacy.[10] To assess the ability to detect lesions in various types of medical images (e.g., liver ultrasound, mammogram), a large ensemble of statistically equivalent images is required. These images may serve as background images into which one may or may not insert objects of interest.[11–15] For example, in optimizing or assessing a mammography imaging system to detect cancerous lesions, a large number of statistically equivalent mammography backgrounds, half with inserted lesions, half without, can be generated.[16] The ensemble of images can be created using texture synthesis as an alternative to establishing a large pool of certified ''normal'' mammograms.

An approach to mammography texture synthesis is shown in Fig. 1 where some underlying small-scale texture is extracted from the larger scale. Two realizations of the synthesized small-scale texture are shown. The larger scale may be synthesized using, for example, lumpy backgrounds.[11] By recombining the synthetic structures, syn-

*Current address: Dept. of Medical Physics and Imaging, Cedars-Sinai Medical Center, 8700 Beverly Blvd., Davis 6065, Los Angeles, CA 90048.

Paper 98048 received Apr. 15, 1998; revised manuscript received May 11, 1999 and Oct. 29, 1999; accepted for publication Nov. 3, 1999.
1017-9909/2000/$15.00 © 2000 SPIE and IS&T.

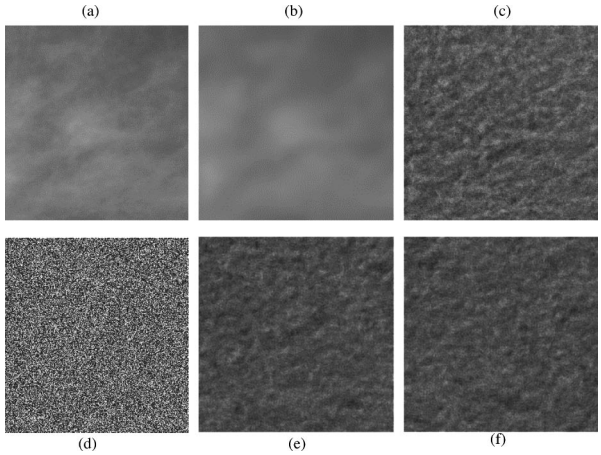

Fig. 1 In the top row, (a) shows a 256×256 pixel image of a sample mammography tissue; the same image blurred with a Gaussian of σ 6 pixels is shown in (b). The underlying mammography texture shown in (c) is obtained by subtracting image (b) from image (a). In the bottom row, a realization of a white noise image is shown in (d); and two examples of synthesis of the underlying mammography texture shown in (c) are displayed in (e) and (f), respectively.

thetic mammograms may be obtained. Naturally, the need for large ensembles of statistically equivalent images for image quality assessment may apply equally well to domains of image science other than medical imaging.

The method of texture synthesis, that we employ to make, for example, the mammography texture samples shown in Fig. 1, comprises a technique known as histogram matching between two images recursively. Histogram matching, sometimes referred to as histogram specification, is an image processing technique, specifically a point operation, which modifies a candidate image so that its histogram matches that of a model image.[17] While histogram matching is not widely employed in image processing, it is a generalization of histogram equalization, an image processing technique commonly employed to enhance low-contrast images.[18–20] The synthesis algorithm will be further detailed in Sec. 7.1. Based on applications that require either the generation of on-the-fly synthetic textures (i.e., computer graphics) or large ensemble of synthetic textures (i.e., image quality assessment in image science) high-speed computation is necessary for all procedures of the texture synthesis algorithm, including histogram matching. A further important point for the motivation of a faster histogram-matching algorithm is that we shall encounter multiple histogram matching steps in the synthesis of a single realization of a texture as a consequence of the multiple-scale and multiple-orientation decomposition required by the texture synthesis algorithm. The histogram-matching algorithm thus needs to be efficient for small images (e.g., 16×16 pixels), as well as large ones (e.g., 256 ×256 pixels).

A basic question that thus motivated this research is how to speed up the texture synthesis algorithm. For equally fast algorithms, we shall also value the simplicity of the algorithm. The investigation of how various components of the texture synthesis algorithm can be optimized for increased computational speed will be reported elsewhere. This paper reports on establishing fast, and, if possible, simple, histogram-matching algorithms. In this paper, two algorithms are presented. Within the context of texture synthesis, the overall computational-speed performance and simplicity of the algorithms are also assessed.

2 Approaches to Histogram Matching

The most common approach to matching the histogram of two images starts by computing their gray level histograms and their CDFs. Matching the two CDFs is then conducted as summarized in Sec. 3. We refer to this technique as the CDF-matching approach.[17,21–24] It is to our knowledge the only algorithm of histogram matching given in the literature. Based on this basic algorithm, an optimal CDF-matching algorithm, referred to as the OCM algorithm, is proposed. The algorithm is optimal in the sense that, among possible implementations of CDF-matching, it minimizes computational time by employing not only look-up tables (LUTs) for performing histogram matching, but importantly, the property that the CDF is a monotonically nondecreasing function. Once established, this improvement seems a simple idea. However when not accounted for, it leads to suboptimal computational times and higher complexity.

We shall also present an alternative to the CDF-matching approach referred to as the sort-matching algorithm. The benefits of this approach are its simplicity, lower overhead than CDF matching, and high speed. Its simplicity lies in the fact that it does not employ either histogram or cumulative distribution computations. Rather, the approach is based on the matching of two sorted arrays, thus its name. A theoretical time analysis shows, however, that the sort-matching algorithm is more complex than the OCM algorithm. This implies that as the images get larger, the time complexity overwhelms the benefits of lower overhead. For images larger than 64×64 pixels, OCM is effectively faster. However, for the application of texture synthesis demonstrated in Sec. 7, which involves images of various sizes due to the multiple-scale decomposition from 256×256 to 16×16 pixels images, it will be shown that the sort-matching algorithm is still the fastest algorithm. An additional important feature of the algorithm is the exact matching of the histograms obtained as discussed in Sec. 7.2. The sort-matching algorithm described assumes that the two images have equal size. A strategy to apply the algorithm to images of different sizes is discussed in Sec. 7.2 as well.

3 Histogram Matching with the CDF-Matching Algorithm

Given a candidate image B, whose histogram is to be matched to that of a model image A, the CDF-matching algorithm proceeds in four steps:[23] 1. The normalized histograms H_A and H_B of images A and B, respectively, are computed by dividing the histogram values by the total number of pixels in the image. 2. The CDFs, CDF_A and CDF_B of images A and B, respectively, are then formed from the normalized histograms. Each CDF then operates as a LUT, where the indices of the LUT are the gray level values (0–255), and the content of the LUT at each index is the value of the CDF. 3. For each pixel in image B of associated gray level i, the corresponding value of CDF_B corresponding to gray level i is determined via the LUT.

The corresponding gray level value j in image A such that $(\mathrm{CDF\_A})_j$ equal $(\mathrm{CDF\_B})_i$ must then be found. 4. The final step is to substitute gray level j for gray level i in image B.

The specific implementation of the CDF-matching algorithm will impact the computational speed. We shall estimate the minimum number of operations that are required to perform CDF matching under the assumption that all computations are executed on one processor. We shall also assume a bin size for histogram computation of one, where the gray levels of the image are binned into 256 different levels spanning 0–255 in increments of one. Any smaller binning of the gray level values would require a larger number of operations for CDF computation. We shall use K to denote the number of bins used in histogram computation and N to denote the number of pixels in the image. A time analysis of this algorithm breaks the steps into several parts: N operations per image to compute the histogram; $K-1$ operations to make the CDFs; on average $K/2$ operations to find the CDF_A value that is closest to that of the CDF_B value; and N operations to update the gray level values of the image. The $K/2$ matching operation is done for each of the N update operations. In summary, the total time analysis yields an $O(N+K+KN)$ or $O(KN)$ algorithm.

4 The Optimal CDF-matching Algorithm

To improve the speed of performing CDF matching, a LUT between the two functions to be matched can be built. The two functions must first be set up in two arrays with CDFs. A LUT is then formed based on the target and the source arrays by starting with the first target value and then searching the source array for the nearest value to the target value. This search can be performed in a monotonically nondecreasing manner because any new matching value will necessarily be greater than or equal to the previously acquired matching value.

The C code is essentially two lines in a loop:

```
void OCM(float target[], float source[], int AOUT[])

{
int i, cur_ix=0;
for (i=0;i<256;i++)/** Get the value of target [i]**/
      {
   /**Find the source value equal to or
   **greater than the current target value**/
while (source[cur_ix]<target[i])
   cur_ix++;
   /**Find out if the current or previous source
   **value is closer to the current target value**/
     if  (source[cur_ix]-target[i])<(target[i]-source
     [cur_ix-1])
AOUT[i]=cur_ix;
   else
AOUT[i]=cur_ix-1;
      }
}
```

As previously established, it takes $N+K$ operations for histogram and CDF computations. Based on the linear search, $K*K$ operations are needed to build the table. N operations are required to update image B. Thus, the total time analysis yields an $O(N+K+KK+N)$ or $O(N+KK)$ algorithm. If a binary search instead of a linear search is employed to build up the table, building the LUT takes $K \log K$ operations instead. This yields an $O(N+K \log K)$ algorithm. Finally, when accounting for the fact that the two CDF functions are monotonically nondecreasing, the LUT can be constructed in only K operations. The algorithm performance thus improves from $O(N+K \log K)$ to $O(N+K)$.

5 Histogram Matching with the Sort-matching Algorithm

The sort-matching algorithm utilizes sorting to implement an exact histogram matching rather than creating histograms as was done in the CDF-matching algorithm. By ''exact histogram matching'' we mean that upon completion of the algorithm, the number of pixels of a particular gray level is the same for both images. The assignment is such that the lowest gray level pixel of the candidate image is assigned the value of the lowest gray-level pixel of the model image. The next-to-lowest gray level pixel of the candidate image is assigned the value of the next-to-lowest gray level pixel of the model image. This process is repeated until all the pixel values have been assigned.

Let us again presume that B represents a candidate image we wish to match to a model image A of the same size. Rather than the two usual indexing of the images, we will assume that the pixel values are arranged into one-dimensional arrays. Bold characters are employed to denote vector quantities. This lexicographical indexing of the images, which is generally trivial to implement in most computing languages (e.g., C, Fortran), allows multidimensional arrays to be reduced to one dimension in procedures and functions. In higher-level languages such as IDL, redimensioning the array is unnecessary since the language allows a multidimensional array element to be accessed by its one-dimensional lexicographical index. The size of the linear arrays is the total number of pixels in each image. The algorithm is implemented in two simple steps:

1. The pixel values of the two images are sorted in ascending order. The lexicographic indices corresponding to the sorted gray levels are denoted **IND_A** and **IND_B** for images **A** and **B**, respectively. The first element of the list **IND_A** contains the index of the lowest gray level value of **A**. The second element of **IND_A** contains the index of the next-to-lowest gray level value of **A**, etc. The same applies to **IND_B**. Any of the standard sort routines can be used for this step. We used the generally accepted fastest sort algorithm, QuickSort. The average time complexity of QuickSort is $O(N \log_2 N)$.

2. The candidate image **B** is then assigned the gray level values of the model image **A** according to the sorted gray level values. We can define this step by the following equation:

$$\mathbf{B}(\mathbf{IND\_B}) = \mathbf{A}(\mathbf{IND\_A}). \tag{1}$$

This sequence of sorting and substitution yields an image **B** whose histogram is matched with that of **A**. An illustration of the sort-matching algorithm can be found in Rolland *et al.*[25]

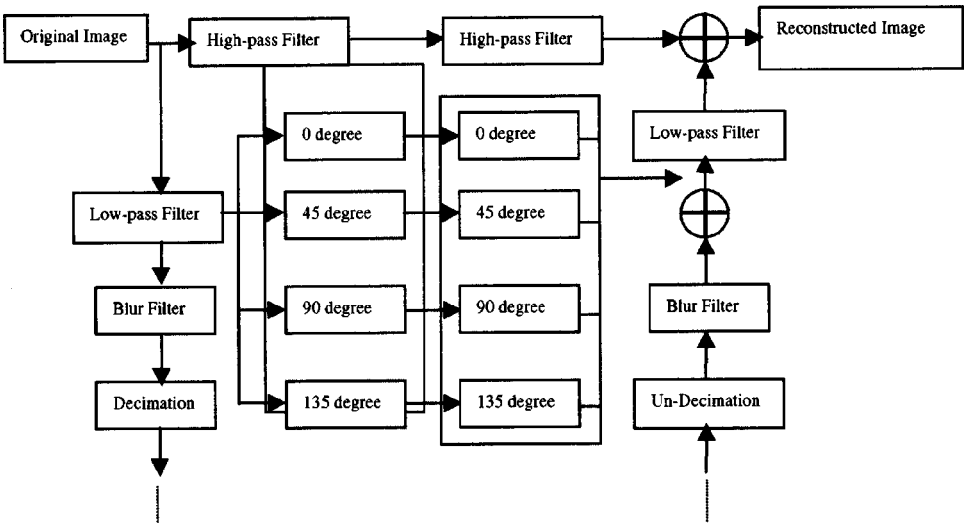

Fig. 2 Illustration of one scale level of the steerable pyramid transform used in the texture synthesis algorithm.

6 Comparison of Computational Speeds of the Various Approaches

We have estimated the minimum number of operations for the conventional CDF matching, the OCM, and the sort-matching algorithms to be $O(NK)$, $O(N+K)$, and $O(N \log_2 N)$, respectively, where N is the number of pixels and K is the number of gray level values. Therefore, the OCM and the sort-matching algorithms outperform the conventional CDF-matching algorithm by a large margin. The OCM algorithm outperforms the sort-matching algorithm where N is much larger than K. If K was sufficiently large to wash out the effect of N, which would be the case for true-color values ($K=2^{24}$), then the sort-matching algorithm would be the fastest.

7 Application to Texture Synthesis

7.1 *Texture Synthesis Algorithm*

The algorithm for texture synthesis we have developed is based on a multiple-scale and multiple-orientation decomposition of a sample from a model texture image and the same decomposition of a realization of a uniformly distributed white noise image.[7,9] The decomposition is depicted in Fig. 2. The decomposition may use, for example, the steerable pyramid transform.[26–29] The histograms of the decomposed white noise images existing at multiple scales are then matched with that of the sample texture. After decomposition and histogram matching at all scales and orientations, the noise subband images are recombined to yield a synthetic image. The algorithm was implemented in the IDL language and is further summarized in Rolland and Strickland.[9]

7.2 *Results and Discussion*

Synthesized textures using either the OCM, the conventional CDF matching, or the sort-matching algorithms are shown in Fig. 3 for two different model texture images, respectively. Two realizations of image synthesis were generated for each texture model using two different realizations of the input noise image.

Given one realization of a noise image, the synthesized images are perceptually identical using the three methods. Their histograms, however, are slightly different because only the sort-matching algorithm performs exact histogram matching as shown. The CDF-matching algorithm performs only a relatively close match. A comparison of performance based on the same noise realizations for each method allows assessment of potential artifacts (e.g., random scrambling of pixel values) that could have been introduced by an approach (e.g., the sort-matching algorithm).

The computational speeds of the algorithms are summarized in Table 1. The timing data were collected on a Sun-Sparc4 workstation running SOLARIS 2.5.1 and using the IDL programming language. For each synthetic image, a computational time was recorded for the histogram matching procedure at each scale of the decomposition and for a single iteration of the overall processing of the synthesis. Seven iterations were typically conducted. The texture synthesis employing the sort-matching and OCM algorithms are, respectively, about 10 and 60 times faster than when employing the conventional CDF-matching algorithm for a 256×256 image. Averaged over the two computational trials, the sort-matching algorithm is the fastest of the three.

The OCM algorithm comes next and quite close. The OCM also does not degrade as rapidly with image size past 128×128 pixel images. It can be noted that for texture synthesis, algorithms that differ in the histogram-matching algorithm employed, the overall computational time required for histogram matching is small compared to the decomposition and filtering contributions. We are currently investigating other decomposition schemes that will further contribute to a higher computational speed. However, a faster histogram matching procedure is a first step toward faster synthesis of images using subband decomposition and histogram matching.

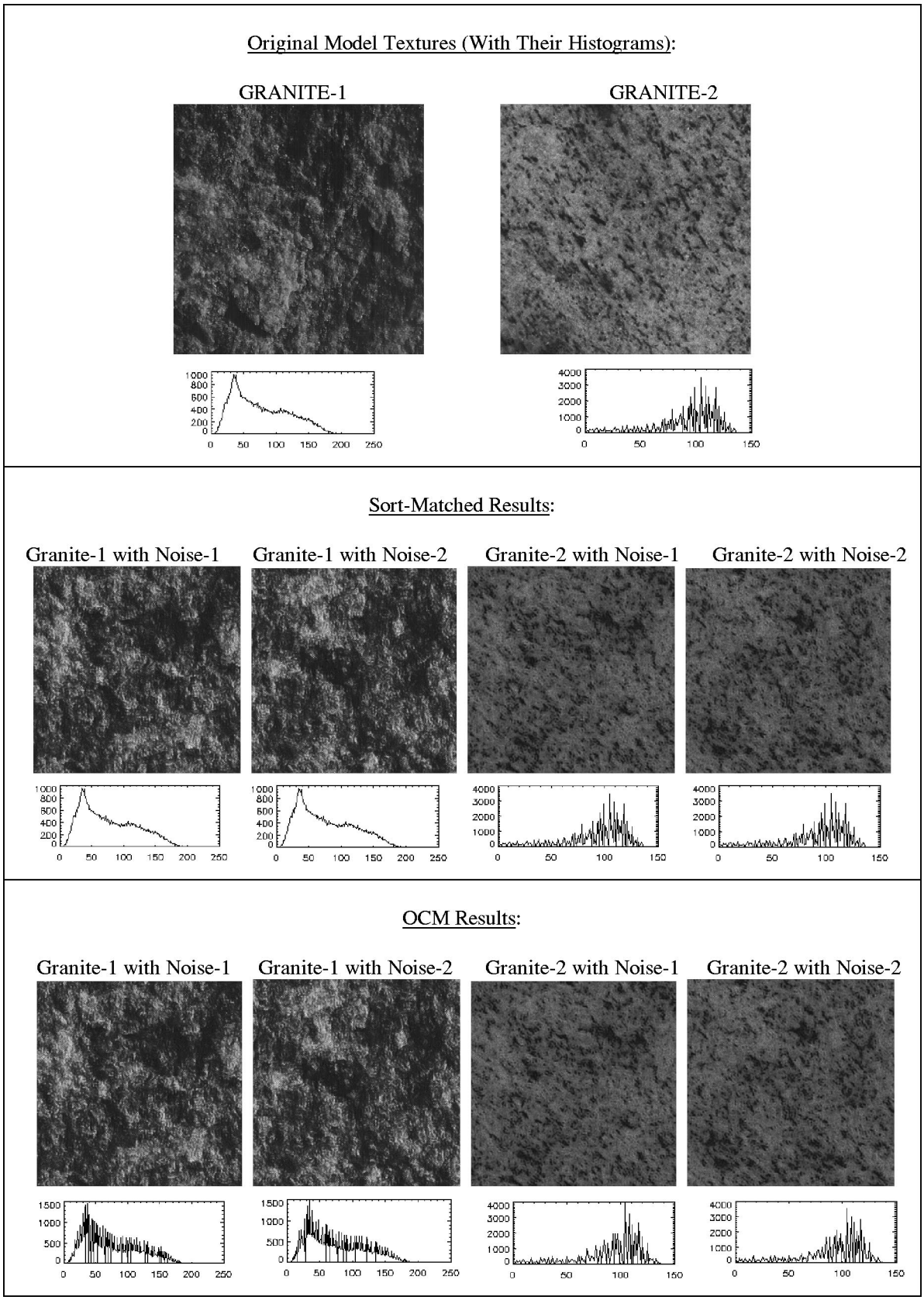

Fig. 3 Results of texture synthesis by two methods of histogram matching for two textures. The histograms of each texture and the syntheses are also shown.

Table 1 Timing data in s. The computational speeds for the OCM, the conventional CDF matching, and the sort-matching algorithms are reported for 256 gray scale images. The scale refers to the size of the decomposed subimages for texture synthesis. Synthesis 1 and 2 refer to the two different realizations of white noise as a starting point for the synthesis. Overall refers to an iteration of the synthesis algorithm.

Options / Scale	OCM algorithm (s)		CDF-matching algorithm		Sort-matching algorithm (s)	
	Synthesis 1	Synthesis 2	Synthesis 1	Synthesis 2	Synthesis 1	Synthesis 2
256×256	1.49	1.53	91.79	107.25	10.68	10.92
128×128	0.40	0.40	23.12	23.55	0.43	0.44
64×64	0.129	0.126	4.96	6.27	0.09	0.093
32×32	0.058	0.060	1.28	1.29	0.02	0.021
Overall	227.46	305.98	1402.73	1561.54	228.80	237.99

It is interesting to note that some expected, but uncommon, gray level mappings occurred with the sort-matching algorithm. Let us consider the 2×2 left-corner subimages A_L and B_L of two images A and B, respectively, given by

$$A_L=\begin{bmatrix}1 & 1\\ 2 & 2\end{bmatrix}$$

and

$$B_L=\begin{bmatrix}5 & 5\\ 5 & 5\end{bmatrix}.$$

As a result of the sort-matching algorithm, four identical gray levels given by B_L can be mapped to two different gray levels. It is indeed the nature of any exact histogram-matching algorithm to yield such mappings. If this mapping were not preferred, searching for elements of the sorted arrays with identical pixel values and resetting their gray levels to identical gray level values would modify the mapping. Such a remapping would lead to nonexact histogram matching and it would cost computational time. In the application of texture synthesis presented here, such mapping as a result of sort matching seems acceptable as shown by the perceptually similar syntheses when we applied both methods to a single realization of the noise. Ultimately, an analysis of the statistical properties may be required for quantitative analysis.[30]

Finally, the sort-matching algorithm assumes that the two images being matched have the same size, which can be satisfied in the subband-decomposition-based texture synthesis application. In the case of other applications or other texture synthesis frameworks, where the images have dissimilar sizes, the smaller image could be, for example, interpolated to equal the other image in size before the sort-matching algorithm is applied.

8 Conclusion

Texture synthesis based on the steerable pyramid transform requires multiple histogram-matching operations of images that have been decomposed at multiple scales and orientations. We have shown that the sort-matching and the OCM histogram-matching algorithms provide high computational speed for texture synthesis, compared to the conventional approach. In some applications, the sort-matching algorithm may be preferred because it provides exact histogram matching. Results of texture synthesis that use either of the methods presented here show no subjective perceptual differences. The sort-matching algorithm is attractive because of its simplicity and speed, however, as the size of the image increases, the OCM may be preferred for optimal computational speed.

Acknowledgments

This work was supported by the University of Central Florida Small Grant Program, the Florida Hospital Gala Endowed Program, and the Florida I4-Corridor. We thank Liyun Yu for his early contribution to this work concerning the implementation of the texture synthesis algorithm.

References

1. G. C. Cross and A. K. Jain, ''Markov random field texture models,'' *IEEE Trans. Pattern. Anal. Mach. Intell.* **5**, 25–39 (1983).
2. R. Chellappa and R. L. Kashyap, ''Texture synthesis using 2-D noncausal autoregressive models,'' *IEEE Trans. Acoust., Speech, Signal Process.* **33**, 194–203 (1985).
3. J. P. Lewis, ''Algorithms for solid noise synthesis,'' *Comput. Graph.* **23**, 263–270 (1989).
4. A. Witkin and M. Kass, ''Reaction-diffusion textures,'' *Comput. Graph.* **25**, 299–308 (1991).
5. P. S. Heckbert, ''Survey of texture mapping,'' *IEEE Comput. Graphics Appl.* **6**, 56–67 (1986).
6. G. Turk, ''Generating textures on arbitrary surfaces using reaction diffusion,'' *Comput. Graph.* **25**, 289–298 (1991).
7. D. J. Heeger and J. R. Bergen, ''Pyramid-based texture analysis/synthesis,'' in *Proc. Computer Graphics*, pp. 229–238, Los Angles, CA, August 6–11 (1995).
8. J. P. Rolland, A. Goon, and L. Yu, ''Synthesis of textured complex backgrounds,'' *Opt. Eng.* **37**(7), 2055–2063 (1998).
9. J. P. Rolland and R. Strickland, ''An approach to the synthesis of biological tissue,'' *Opt. Express* **1**(13), 414:423 (1997).
10. J. P. Rolland, ''Synthesizing anatomical images for image understanding and quality assessment,'' *Handbook of Medical Imaging, Progress in Medical Physics and Psychophysics*, Beutel, Van Metter, and Kundel, Eds., Vol. II, Chapter 13, SPIE, Bellingham, WA (to be published).
11. K. J. Myers, J. P. Rolland, H. H. Barrett, and R. F. Wagner, ''Aperture optimization for emission imaging: effect of a spatially varying backgrounds,'' *J. Opt. Soc. Am. A* **7**, 1279–1293 (1990).
12. J. P. Rolland and H. H. Barrett, ''Effect of random background inhomogeneity on observer detection performance,'' *J. Opt. Soc. Am. A* **9**, 649–658 (1992).
13. A. E. Burgess, ''Statistically-defined backgrounds: performance of a modified nonprewhitening observer model,'' *J. Opt. Soc. Am. A* **11**, 1237–1242 (1994).
14. P. F. Judy, ''Detection of clusters of simulated calcifications in lumpy

noise backgrounds,'' *Proc. SPIE* **2712**, 39–46 (1996).
15. R. N. Strickland, ''Wavelet transforms for detecting microcalcifications in mammograms,'' *IEEE Trans. Med. Imaging* **15**(2), 218–229 (1996).
16. H. H. Barrett, J. Yao, J. P. Rolland, and K. J. Myers, ''Model observers for assessment of image quality,'' *Proc. Natl. Acad. Sci. USA* **90**, 9758–9765 (1993).
17. K. R. Castleman, *Digital Image Processing*, Prentice Hall, Upper Saddle River, NJ (1996).
18. E. H. Hall, ''Almost uniform distributions for computer image enhancement,'' *IEEE Trans. Comput.* **C-23**(2), 207–208 (1974).
19. S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. T. H. Romeny, J. Zimmerman, and K. Zuiderveld, ''Adaptive histogram equalization and its variations,'' *Comput. Vis. Graph. Image Process.* **39**, 355–368 (1987).
20. R. B. Paranjape, W. M. Morrow, and R. M. Rangayyan, ''Adaptive-neighborhood histogram equalization for image enhancement,'' *Graphical Models Image Proc.* **54**(3), 259–267 (1992).
21. W. K. Pratt, *Digital Image Processing*, Wiley, New York (1978).
22. M. Wegener, ''Destriping multiple sensor imagery by improved histogram matching,'' *Int. J. Remote Sens.* **11**(5), 859–875 (1990).
23. R. C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, Reading, MA (1992).
24. A. R. Weeks, *Fundamentals of Electronic Image Processing*, SPIE Optical Engineering, Bellingham, WA (1996).
25. J. P. Rolland, V. Vo, L. Yu, B. Bloss, and C. K. Abbey, ''An optimal histogram matching algorithm,'' Technical Report TR99-001, May, University of Central Florida (1999).
26. E. P. Simoncelli and E. H. Adelson, ''Subband transforms,'' *Subband Image Coding*, J. W. Woods, Ed., Kluwer Academic, Norwell, MA (1990).
27. E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, ''Shiftable multi-scale transforms,'' *IEEE Trans. Inf. Theory* **38**, 587–607 (1992).
28. P. Perona, ''Deformable kernels for early vision,'' *IEEE Trans. Pattern. Anal. Mach. Intell.* **7**(5), 488–489 (1995).
29. J. W. Woods, *Subband Image Coding*, pp. 143–192, Kluwer Academic, Norwell, MA (1991).
30. A. A. Goon and J. P. Rolland, ''Texture classification based on comparison of second-order statistics: 2P-PDF estimation and distance measure,'' *J. Opt. Soc. Am. A* **16**(7), 1566–1574 (1999).