Real-Time Texture Synthesis by Patch-Based Sampling

LIN LIANG, CE LIU, YING-QING XU, BAINING GUO,
and HEUNG-YEUNG SHUM
Microsoft Research China, Beijing

We present an algorithm for synthesizing textures from an input sample. This patch-based sampling algorithm is fast and it makes high-quality texture synthesis a real-time process. For generating textures of the same size and comparable quality, patch-based sampling is orders of magnitude faster than existing algorithms. The patch-based sampling algorithm works well for a wide variety of textures ranging from regular to stochastic. By sampling patches according to a nonparametric estimation of the local conditional MRF density function, we avoid mismatching features across patch boundaries. We also experimented with documented cases for which pixel-based nonparametric sampling algorithms cease to be effective but our algorithm continues to work well.

Categories and Subject Descriptors: I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*texture*; I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement—*texture*

General Terms: Algorithms

Additional Key Words and Phrases: Texture synthesis, patch-pasted nonparametric sampling

1. INTRODUCTION

Texture synthesis has a variety of applications in computer vision, graphics, and image processing. An important motivation for texture synthesis comes from texture mapping. Texture images usually come from scanned photographs, and the available photographs may be too small to cover the entire object surface. In this situation, a simple tiling will introduce unacceptable artifacts in the forms of visible repetition and seams. Texture synthesis solves this problem by generating textures of the desired sizes. Other applications of texture synthesis include various image processing tasks such as occlusion fill-in and image/video compression.

The texture synthesis problem may be stated as follows. Given an input sample texture I_{in}, synthesize a texture I_{out} that is sufficiently different from the given sample texture, yet appears perceptually to be generated by the same underlying stochastic process. In this work, we use the Markov Random Field

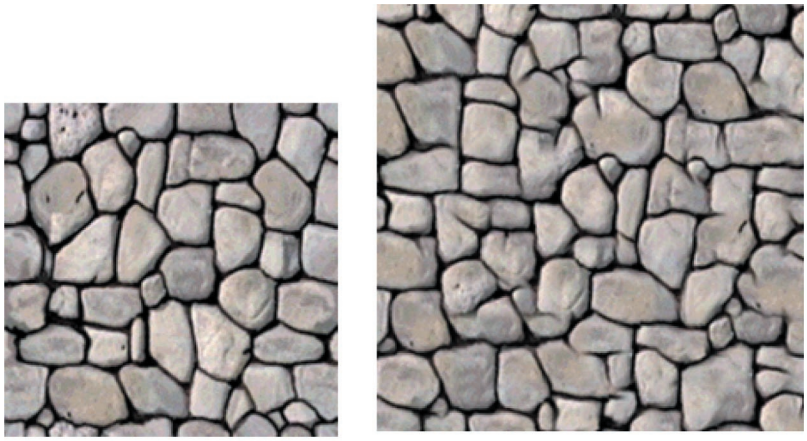
Authors' address: Microsoft Research China, 5F, Beijing Sigma Center, No. 49., Zhichun Road, Haidian District, Beijing, 100080, PRC; email: bainguo@microsoft.com.

(MRF) as our texture model and assume that the underlying stochastic process is both local and stationary. We choose MRF because it is known to accurately model a wide range of textures. Other successful but more specialized models include reaction-diffusion [Turk 1991; Witkin and Kass 1991], frequency domain [Lewis 1984], and fractals [Fournier 1982; Worley 1996] (see also Perlin [1985]).

In recent years, a number of successful texture synthesis algorithms have been proposed in graphics and vision. Motivated by psychology studies, Heeger and Bergen [1995] developed a pyramid-based texture synthesis algorithm that approximately matches marginal histograms of filter responses. Zhu et al. [1997, 1998] introduced a mathematical model called FRAME, which integrates filters and histograms into MRF models and uses a minimax entropy principle to select feature statistics. Several texture synthesis algorithms are based on matching joint statistics of filter responses. De Bonet's [1997] algorithm matches the joint histogram of a long vector of filter responses. Portilla and Simoncelli [1999] developed an iterative projection method for matching the correlations of certain filter responses. These methods, along with many others in the literature [Iverson and Lonnestad 1994; Wu et al. 2000], represent two different approaches to texture synthesis. The first is to compute global statistics in feature space and sample images from the texture ensemble [Zhu et al. 2000] directly.[1] The second approach is to estimate the local conditional probability density function (PDF) and synthesize pixels incrementally [Zhu et al. 1997].

The texture synthesis algorithm we propose follows the second approach. In their pioneer work, Zhu et al. [1997] explored this approach using the analytical FRAME model and an accurate but expensive Markov chain Monte Carlo method. More recently, Efros and Leung [1999] demonstrated the power of sampling from a local PDF by presenting a nonparametric sampling algorithm that works well for a wide variety of textures ranging from regular to stochastic. Efros and Leung's algorithm, while much faster than Zhu et al. [1997], is still too slow. Inspired by a cluster-based texture model [Popat and Picard 1993], Wei and Levoy [2000] significantly accelerated Efros and Leung's algorithm using tree-structured vector quantization (TSVQ). However, this TSVQ-accelerated nonparametric sampling algorithm is still not real-time. Another problem with Efros and Leung [1999] and Wei and Levoy [2000] is that they break down for some textures. Efros and Leung [1999] attribute the problem of their algorithm to the fact that it is a greedy algorithm and it can "slip" into a wrong part of the search space and start to grow "garbage." Wei and Levoy's algorithm also suffers quality problems. Ashikhmin [2001] noted the quality problems of Wei and Levoy's algorithm on a class of textures and has developed a special-purpose algorithm for that class. It is possible to combine Ashikhmin's algorithm with that of Wei and Levoy, as Hertzmann et al. [2001] have done.

[1]Please also see Heeger and Bergen [1995], DeBonet [1997], Portilla and Simoncelli [1999], and Zhu et al. [2000].



(a) (b)

Fig. 1. Texture synthesis example: (a) 192 × 192 input sample texture; (b) 256 × 256 texture synthesized by patch-based sampling. The synthesis takes 0.02 seconds on a 667 MHz PC.

In this article we show that high-quality texture can be synthesized in real-time on a midlevel PC. A key ingredient of our patch-based sampling algorithm is a sampling scheme that uses texture patches of the sample texture as building blocks for texture synthesis. The patch-based sampling algorithm is fast. For synthesizing textures of the same size and comparable (or better) quality, our algorithm is orders of magnitude faster than existing texture synthesis algorithms including Wei and Levoy [2000]. Figure 1 shows an example produced by our algorithm. After spending 0.6 seconds analyzing the input sample, our algorithm took 0.02 seconds to synthesize this texture on a 667 MHz PC. The patch-based sampling algorithm works well for a wide variety of textures ranging from regular to stochastic. We examined documented cases for which Efros and Leung [1999], Wei and Levoy [2000], and Ashikhmin [2001] cease to be effective and have found that our algorithm continues to produce good results. In particular, the texture patches in our sampling scheme provide implicit constraints for avoiding garbage as found in some textures synthesized by Efros and Leung [1999].

The patch-based sampling algorithm is an extension of our earlier work on texture synthesis by random patch pasting [Xu et al. 2000]. Praun [2000] has successfully adapted this patch-pasting algorithm for texture mapping on 3-D surfaces. Unfortunately, these patch-pasting algorithms suffer from mismatching features across patch boundaries. The patch-based sampling algorithm, on the other hand, avoids mismatching features across patch boundaries by sampling texture patches according to the local conditional MRF density. Patch-based sampling includes patch pasting as a special case, in which the local PDF implies a null statistical constraint.

Patch-based sampling is amenable to acceleration and the fast speed of our algorithm is partially attributable to our carefully designed acceleration scheme. The core computation in patch-based sampling can be formulated as a search for approximate nearest neighbors (ANN). We accelerate this search by combining

an optimized technique for general ANN search, a novel data structure called the quadtree pyramid for ANN search of images, and principal components analysis of the input sample texture.

The patch-based sampling algorithm is easy to use and flexible. It is applicable to both unconstrained and constrained texture synthesis. Examples of constrained texture synthesis include hole filling and tileable texture synthesis. The patch-based sampling algorithm has an intuitive randomness parameter. The user can utilize this parameter to interactively control the randomness of the synthesized texture.

In concurrent work, Efros and Freeman [2001] developed a texture quilting algorithm. For unconstrained texture synthesis, texture quilting is very similar to patch-based sampling. There are several differences between our work and that of Efros and Freeman [2001]. First, we accelerate patch-based sampling and demonstrate that it can run in real-time, whereas they do not explore the issue of speed. Second, we show how to solve constrained texture synthesis problems, which they do not address. Finally, patch-based sampling and Efros and Freeman [2001] use different techniques to improve the transitions between texture patches. We apply feathering [Szeliski and Shum 1997] while Efros and Freeman use a minimum error boundary cut. Later, we compare these two boundary treatment techniques in detail.

The rest of the article is organized as follows. In Section 2, we introduce patch-based sampling and its applications to unconstrained and constrained texture synthesis. In Section 3, we present our acceleration techniques. The texture synthesis results and speed are discussed in Section 4, followed by conclusions and suggestions for future work in Section 5.

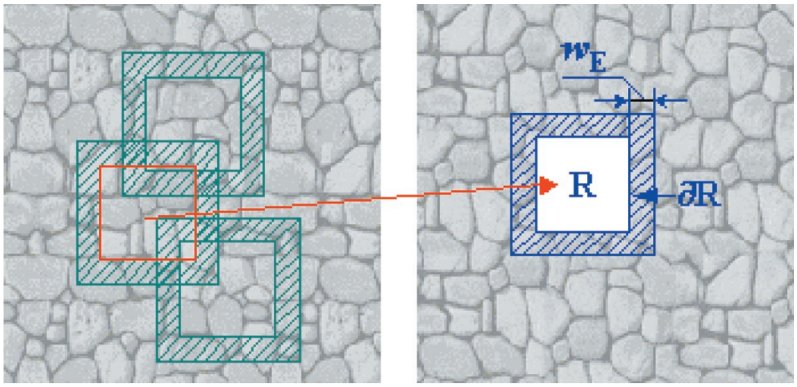
2. PATCH-BASED SAMPLING

The patch-based sampling algorithm uses texture patches of the input sample texture I_{in} as the building blocks for constructing the synthesized texture I_{out}. In each step, we paste a patch B_k of the input sample texture I_{in} into the synthesized texture I_{out}. To avoid mismatching features across patch boundaries, we carefully select B_k based on the patches already pasted in I_{out}, $\{B_0, \ldots, B_{k-1}\}$. The texture patches are pasted in the order shown in Figure 3. For simplicity, we only use square patches of a prescribed size $w_B \times w_B$.

2.1 Sampling Strategy

Let I_{R_1} and I_{R_2} be two texture patches of the same shape and size. We say that I_{R_1} and I_{R_2} match if $d(R_1, R_2) < \delta$, where $d()$ represents the distance between two texture patches and δ is a prescribed constant.

Assuming the Markov property, the patch-based sampling algorithm estimates the local conditional MRF (FRAME or Gibbs) density $p(I_R \mid I_{\partial R})$ in a nonparametric form by an empirical histogram. Define the boundary zone ∂R of a texture patch I_R as a band of width w_E along the boundary of R as shown in Figure 2. When the texture on the boundary zone $I_{\partial R}$ is known, we would like to estimate the conditional probability distribution of the unknown texture patch I_R. Instead of constructing a model, we directly search the input sample



(a) (b)

Fig. 2. A patch-based sampling strategy. In the synthesized texture shown in (b), the hatched area is the boundary zone. In the input sample texture shown in (a), three patches have boundary zones matching the texture patch I_R in (b) and the red patch is selected.

texture I_{in} for all patches having the known $I_{\partial R}$ as their boundary zones. The results of the search form an empirical histogram Ψ for the texture patch I_R. To synthesize I_R, we just pick an element from Ψ at random. Mathematically, the estimated conditional MRF density is

$$p(I_R \mid I_{\partial R}) = \sum_i \alpha_i \delta(I_R - I_{R^i}), \qquad \sum_i \alpha_i = 1,$$

where I_{R^i} is a patch of the input sample texture I_{in} whose boundary zone $I_{\partial R^i}$ matches the boundary zone $I_{\partial R}$ and $\delta()$ is Dirac's delta. The weight α_i is a normalized similarity scale factor.

With patch-based sampling, the statistical constraint is implicit in the boundary zone ∂R. A large boundary zone implies a strong statistical constraint. Generally speaking, a nonparametric local conditional PDF such as in Efros and Leung [1999] and Wei and Levoy [2000] is faster to estimate than the analytical FRAME model in Zhu et al. [1997]. On the down side, the nonparametric density estimation is subject to greater statistical fluctuations, because in a small sample texture I_{in} there may be only a few sites that satisfy the local statistical constraints.

2.2 Unconstrained Texture Synthesis

Now we use the patch-based sampling strategy to choose the texture patch B_k, the kth texture patch to be pasted into the output texture I_{out}.

As Figure 3(a) shows, only part of the boundary zone of B_k overlaps the boundary zone of the already pasted patches $\{B_0, \ldots, B_{k-1}\}$ in I_{out}. We say that two boundary zones match if they match in their overlapping region. In Figure 3(a), B_k has a boundary zone E_{B_k} of width w_E. The already pasted patches in I_{out} also have a boundary zone E^k_{out} of width w_E. According to the patch-based sampling strategy, E_{B_k} should match E^k_{out}.

For the randomness of the synthesized texture I_{out}, we form a set Ψ_B consisting of all texture patches of I_{in} whose boundary zones match E^k_{out}. Let $B_{(x,y)}$

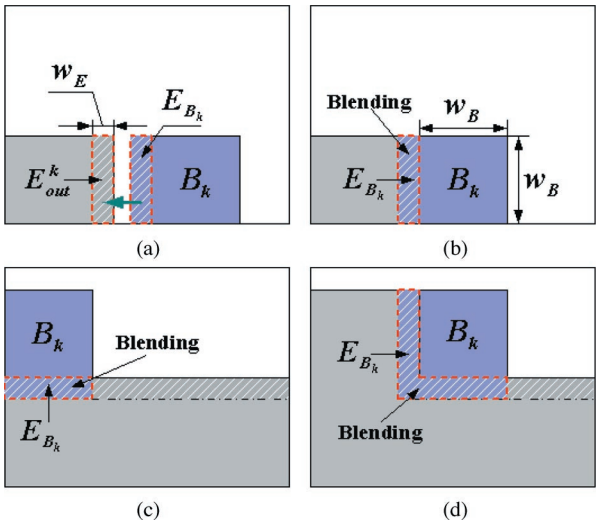


Fig. 3. Texture synthesis by patch-based sampling. The grey area is already synthesized. The hatched areas are the boundary zones. (a) The boundary zones E^k_{out} and E_{B_k} should match. (b), (c), and (d) are three configurations for boundary zone matching. The overlapping boundary zones are blended together.

be the texture patch whose lower left corner is at (x, y) in I_{in}. We form

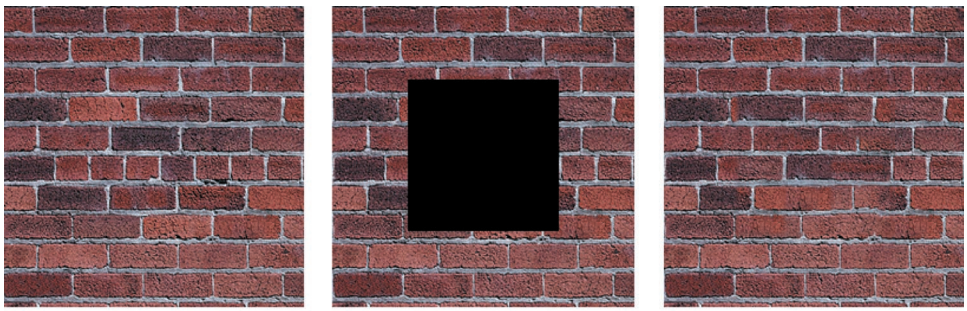
$$\Psi_B = \left\{ B_{(x,y)} \mid d\left(E_{B_{(x,y)}}, E^k_{out}\right) < d_{\max}, B_{(x,y)} \text{ in } I_{in} \right\}, \tag{1}$$

where $d_{\max}$ is the distance tolerance of the boundary zones. Later we give details on how to compute $d_{\max}$ as a function of E^k_{out}. From Ψ_B we randomly select a texture patch to be the kth patch to be pasted. For a given $d_{\max}$, the set Ψ_B could be empty. In that case, we choose B_k to be a texture patch in I_{in} with the smallest distance $d(E_{B_k}, E^k_{out})$.

The patch-based sampling algorithm proceeds as follows.

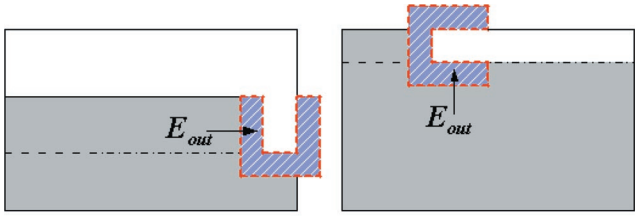
1. Randomly choose a $w_B \times w_B$ texture patch B_0 from the input sample texture I_{in}. Paste B_0 in the lower left corner of I_{out}. Set $k = 1$.
2. Form the set Ψ_B of all texture patches from I_{in} such that for each texture patch of Ψ_B, its boundary zone matches E^k_{out}.
3. If Ψ_B is empty, set $\Psi_B = \{B_{\min}\}$, where $B_{\min}$ is chosen such that its boundary zone is the closest to E^k_{out}.
4. Randomly select an element from Ψ_B as the kth texture patch B_k. Paste B_k onto the output texture I_{out}. Set $k = k + 1$.
5. Repeat steps 2, 3, and 4 until I_{out} is fully covered.
6. Perform blending in the boundary zones.

The blending step uses feathering [Szeliski and Shum 1997] to provide a smooth transition between adjacent texture patches after I_{out} is fully covered with texture patches. Alternatively, it is possible to perform a feathering operation after each new texture patch is found.



(a) (b) (c)

Fig. 4. Constrained texture synthesis: (a) 256×256 texture; (b) 128×128 hole is created; (c) result of constrained synthesis.



(a) (b)

Fig. 5. Boundary zone matching for tileable texture synthesis. The grey area is the texture already synthesized. The hatched purple areas are the areas to be matched. (a) Boundary zone matching for the last patch in a row. (b) Boundary zone matching for the last row.

Forming the set Ψ_B is the main computation for patch-based sampling. This computation is essentially an ANN search in high-dimensional space. We discuss fast searching techniques in Section 3.

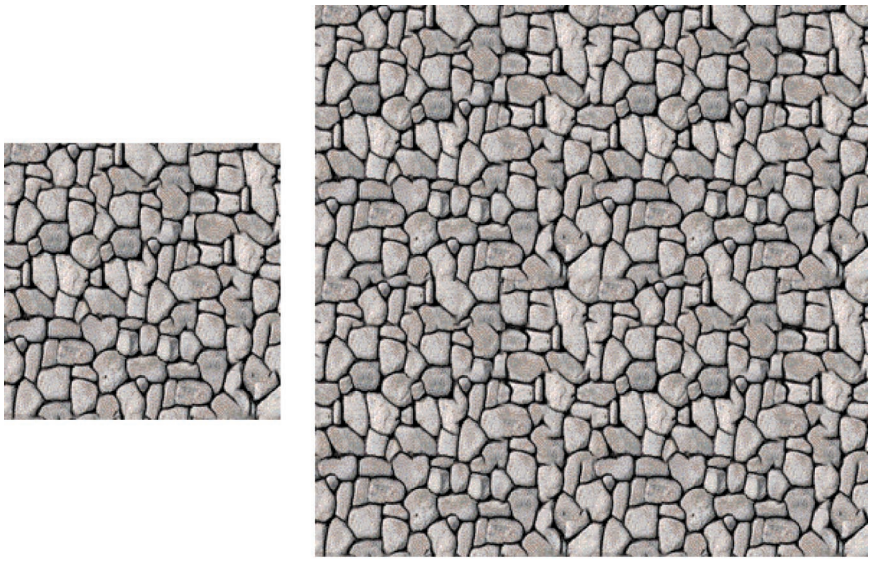
2.3 Constrained Texture Synthesis

Hole Filling. It is straightforward to extend the patch-based sampling algorithm to handle constrained texture synthesis. To better match the features across patch boundaries between the known texture around the hole and newly pasted texture patches, we fill the hole in spiral order. Figure 4 shows an example.

Tileable Texture Synthesis. This is another form of constrained texture synthesis. Figure 5 shows the boundary zones to be matched for the last patch in a row and for the patches of the last row. In the synthesized texture I_{out}, the pixel values in the boundary zone should be

$$E_{out}(x, y) = I_{out}(x \bmod w_{out}, y \bmod h_{out}),$$

where (x, y) is the location of the pixel in I_{out}. w_{out} and h_{out} are the width and height of the synthesized texture. This equation defines the pixels in the boundary zone E_{out} even if either $x > w_{out}$ or $y > h_{out}$ as shown in Figure 5. Figure 6 shows a result of tileable texture synthesis.



(a) (b)

Fig. 6. Result of tileable texture synthesis: (a) tileable texture synthesized from the input sample in Figure 1; (b) 2×2 tiling of the synthesized texture.

2.4 Implementation Details

Patch Size (w_B). The size of the texture patch affects how well the synthesized texture captures the local characteristics of the input sample texture I_{in}. A smaller w_B allows more matching possibilities between texture patches and thus implies weaker statistical constraints and less similarity between the synthesized texture I_{out} and the input sample texture I_{in}. Up to a certain limit, a bigger w_B means better capturing of texture characteristics in the texture patches and thus more similarity between I_{out} and I_{in}.

Figure 7 shows the effect of w_B on the synthesized textures I_{out}. When $w_B = 16$, the texture patches contain less structural information of the input sample texture I_{in} (size 64×64). As a result, the synthesized texture I_{out} appears more random. For patch size $w_B = 32$, the synthesized texture I_{out} become less random and resembles the input sample texture of I_{in} more.

For an input sample texture of size $w_{in} \times h_{in}$, the patch size w_B should be $w_B = \lambda \min(w_{in}, h_{in})$, where $0 < \lambda < 1$ is the *randomness parameter* of our system.[2] The intuitive meaning of w_B is the scale of the texture elements in the input sample texture I_{in}. For texture synthesis, it is usually assumed that the approximate scale of the texture elements is known, although we can also use texture analysis techniques such as those of Zucker and Terzopoulos [1980] to find the scale of the texture elements from the given texture sample. The patch size serves a similar function as the window size in Efros and Leung

[2]The boundary zone width w_E can also be used as a control parameter for the randomness of the synthesized texture.

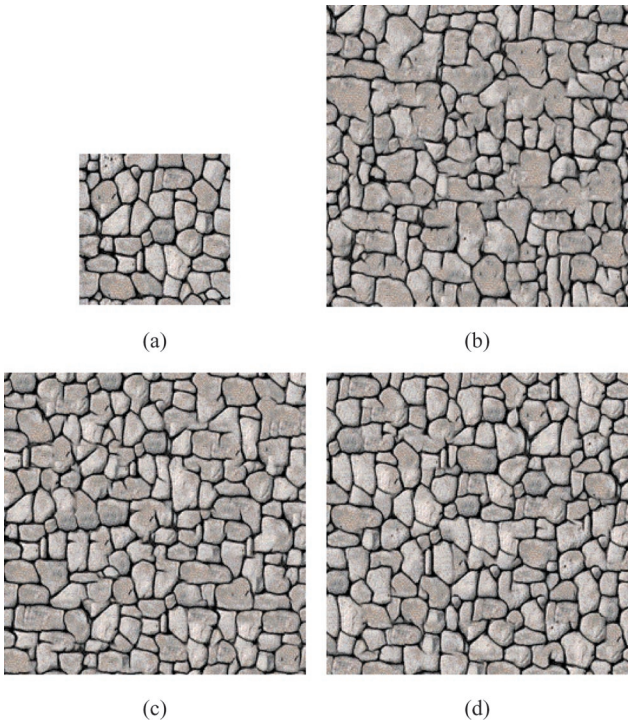


Fig. 7. The effect of the patch size on synthesized texture: (a) input sample texture of size 64×64; (b) synthesized texture when the patch size $w_B = 16$; (c) synthesized texture when $w_B = 24$; (d) synthesized texture when $w_B = 32$. The synthesized texture resembles the sample texture more as w_B increases.

[1999], which is also a randomness parameter. The main difference is that a large window size in Efros and Leung [1999] drastically reduces the texture synthesis speed; the patch size w_B has only minor impact on the synthesis speed.

Unless stated otherwise, all examples in this article are generated with λ values between 0.25 and 0.5.

Distance Metric. We choose the following measure of the distance between two boundary zones

$$d\left(E_{B_k}, E^k_{out}\right) = \left[\frac{1}{A}\sum_{i=1}^{A}\left(p^i_{B_k} - p^i_{out}\right)^2\right]^{1/2}, \tag{2}$$

where A is the number of pixels in the boundary zone. $p^i_{B_k}$ and p^i_{out} represent the values (greyscale or color) of the ith pixel in the boundary zones E_{B_k} and E^k_{out}, respectively.

Boundary Zone Width (w_E). The boundary zone width w_E should be sufficiently large to avoid mismatching features across patch boundaries. A wide boundary zone implies strong statistical constraints, which force a natural

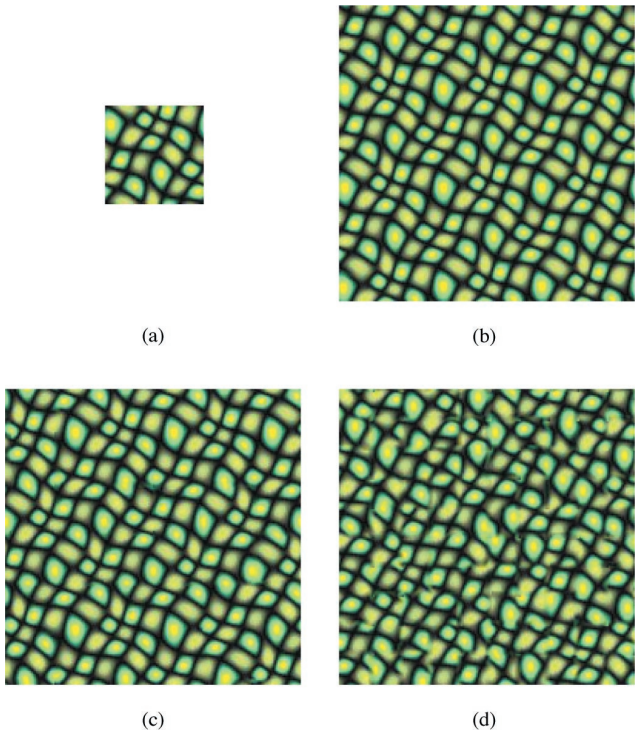


Fig. 8. The effect of different relative error ϵ: (a) input sample texture; (b) $\epsilon = 0$; (c) $\epsilon = 0.2$; (d) $\epsilon = 1.0$.

transition of features across patch boundaries. However, when the boundary zone is too wide, the statistical constraint will become so strong that there will be very few texture patches satisfying the constraints in a small sample texture I_{in}. In that case, patch-based sampling suffers serious statistical fluctuations. As we show, when w_E is too large it is also more costly to construct the kd-tree for accelerating the search for the texture patches of Ψ_B. As a balance, we can set w_E to be a small fraction (e.g., 1/6) of the patch size. For the results reported in this article, w_E is typically four pixels wide.

Distance Tolerance ($d_{\max}$). When the distance between two boundary zones is defined by Equation (2), we define $d_{\max}$ as

$$d_{\max} = \epsilon \left[\frac{1}{A} \sum_{i=1}^{A} \left(p_{out}^{i} \right)^2 \right]^{1/2},$$

where A is the number of pixels in the boundary zone. p_{out}^{i} represents the values of the ith pixel in the boundary zone E_{out}^{k}. $\epsilon \geq 0$ is the relative matching error between boundary zones.

The parameter ϵ controls the similarity of the synthesized texture I_{out} with the input sample texture I_{in} and the quality of I_{out}. The smaller the ϵ, the more similar are the local structures of the synthesized texture I_{out} and the sample texture I_{in}. If $\epsilon = 0$, the synthesized texture I_{out} looks like the tiling of the

sample texture I_{in}, as Figure 8(b) shows. When ϵ is too big, the boundary zones of adjacent texture patches may be very different and thus there may not be a natural transition across the patch boundaries, as Figure 8(d) shows. To ensure the quality of the synthesized texture, we set $\epsilon = 0.2$.

Edge Handling. Let $B_{(x,y)}$ be the texture patch whose lower left corner is at (x, y) in I_{in}. To construct the set Ψ_B we have to test $B_{(x,y)}$ for inclusion in Ψ_B. For (x, y) near the border of the input sample texture I_{in}, part of $B_{(x,y)}$ may be outside I_{in}. If the sample texture I_{in} is tileable, then we set the value of $B_{(x,y)}$ toroidally. $B_{(x,y)}(u, v) = I_{in}(u \bmod w_{in}, v \bmod h_{in})$, where (u, v) is the location of a pixel of $B_{(x,y)}$ inside the sample texture I_{in}, and w_{in} and h_{in} are the width and the height of I_{in}, respectively. If the sample texture I_{in} is not tileable, we only search for the texture patches that are completely inside I_{in}.

2.5 Discussion

Patch-Versus Pixel-Based Sampling. When the patch size $w_B = 1$, patch-based sampling becomes the nonparametric sampling of Efros and Leung [1999] and Wei and Levoy [2000]. When $w_B = 1$, the estimated conditional MRF density becomes

$$p(I(v) \mid I_{\partial v}) = \sum_i \alpha_i \delta(I(v) - I(v_i)), \quad \sum_i \alpha_i = 1,$$

where $I(v_i)$ is a pixel of the input sample texture I_{in} whose neighborhood $I_{\partial v_i}$ matches $I_{\partial v}$. The weight α_i is a normalized similarity scale factor. This is the nonparametric sampling described in Efros and Leung [1999] and Wei and Levoy [2000]. When $w_B = 1$, the window size of Efros and Leung [1999] is $w = 2w_E + 1$, where w_E is the boundary zone width.

Patch-Based Sampling Versus Patch Pasting. When the relative matching error between the boundary zones becomes sufficiently large, say $\epsilon = 1.0$, the patch-based sampling algorithm is essentially the patch-pasting algorithm of Xu et al. [2000].

3. PERFORMANCE OPTIMIZATION

When constructing the set Ψ_B as defined in Equation (1) in Section 2.2, we need to search the set of all $w_B \times w_B$ patches of the input sample texture I_{in} for patches whose boundary zones match E^k_{out}. We formulate this search as a k nearest neighbors search problem in the high-dimensional space consisting of texture patches of the same shape and size as E^k_{out}. The k nearest neighbor problem is a well-studied problem. If we insist on getting the exact nearest neighbors in high dimensions, it is hard to find search algorithms that are significantly better than brute-force search. However, if we are willing to accept approximate nearest neighbors (i.e., ANN), there are efficient algorithms available [Arya et al. 1998].

When choosing acceleration techniques, our principle is to avoid acceleration techniques that will introduce noticeable artifacts in synthesized textures. With this principle in mind, we accelerate our ANN search at three levels.

General acceleration. We accelerate the search at general level using an optimized kd-tree [Mount 1998]. For patch-based sampling, this optimized kd-tree performs as well as the bd-tree, which is optimal for ANN search [Arya et al. 1998].

Domain-specific acceleration. We introduce a novel data structure called the quadtree pyramid for accelerating the search based on the fact that the datapoints in our search space are images.

Data-specific acceleration. We use principal components analysis (PCA) [Jollife 1986] to accelerate the search for the given input sample texture.

The acceleration at all three levels can be combined to get a compound speed-up of the ANN search.

3.1 Optimized KD-Tree

We use an optimized kd-tree [Mount 1998] as our general technique for accelerating the ANN search. Initially we experimented with the bd-tree, which is optimal for ANN search [Arya et al. 1998]. However, our experiments indicate that bd-trees introduce minor but noticeable artifacts in the synthesized textures. In terms of speed, bd and kd-trees are about the same for our searching needs. As pointed out in Arya et al. [1998], the optimized kd-tree, with all its optimizations [Mount 1998], performs as well as the bd-tree on most data sets.[3] We have also experimented with Nene and Nayar's [1997] algorithm as well as TSVQ [Wei and Levoy 2000]. For patch-based sampling, Nene and Nayar [1997] and TSVQ introduce noticeable artifacts in some synthesized textures.

A kd-tree partitions the data space into hypercubes using axis-orthogonal hyperplanes [Friedman et al. 1977; Mount 1998]. Each node of a kd-tree corresponds to a hypercube enclosing a set of datapoints. When constructing a kd-tree, an important decision is to choose a splitting rule for breaking the tree nodes. We use the sliding midpoint rule [Mount 1998]. An alternative choice is the standard kd-tree splitting rule, which splits the dimension with the maximum spread of datapoints. The standard kd-tree splitting rule has a good guarantee on the height and size of the kd-tree. However, this rule produces hypercubes of arbitrarily high aspect ratio. Since we only allow small errors in boundary zone matching, we want to avoid high aspect ratios. The sliding midpoint rule can also lead to hypercubes of high aspect ratios, but these hypercubes have a special property that prevents them from causing problems in nearest neighbor searching [Mount 1998].

To search a kd-tree, we use an adapted version of the search algorithm from Friedman et al. [1977] with the incremental distance computation of Arya et al. [1998]. When the allowed matching errors are small, as is the case for us, the algorithm of Friedman et al. [1977] is slightly faster than the priority search algorithm, which is superior for finding exact nearest neighbors or for large matching errors [Mount 1998].

[3]One case in which the bd-trees do perform significantly better is when the datapoints are clustered in low-dimensional subspaces, but this is not the case with our texture data.

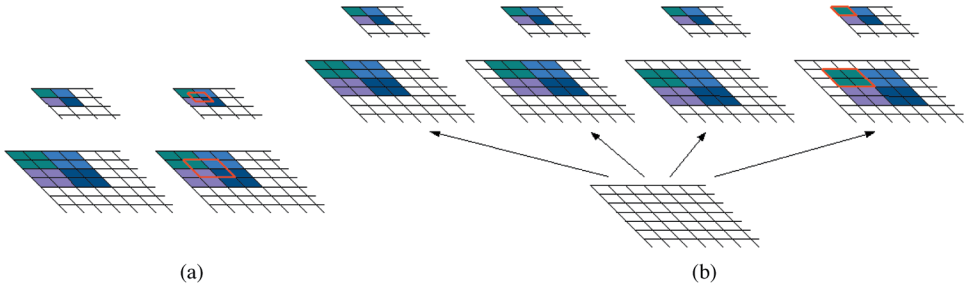


Fig. 9. The quadtree pyramid: (a) two levels in a standard Gaussian pyramid. The pixels in the red rectangle in the lower level do not have a corresponding pixel in the higher level; (b) two levels in a quadtree pyramid. Every set of four pixels has a corresponding pixel in the higher level.

For implementation, we use the ANN library [Mount 1998] to build a kd-tree for each of the three boundary zone configurations shown in Figure 3.

3.2 Quadtree Pyramid

The kd-tree acceleration does not directly take advantage of the fact that our datapoints correspond to images. We introduce the quadtree pyramid (QTP) to address this problem. QTP is a data structure for hierarchical search of image data. To find approximate nearest neighbors for a query vector $\mathbf{v}$, we find the m initial candidates using the low-resolution datapoints and query vector $\mathbf{v}$. In general we should choose $m \ll n$, where n is the number of datapoints. In our system, we set $m = 40$. From the initial candidates, we can find the k nearest neighbors using high-resolution query vector $\mathbf{v}$ and datapoints.

In order to accelerate the search of the m initial candidates, we need to filter all datapoints and the query vector $\mathbf{v}$ into low resolution. A naive approach to filter them is to do it one by one, which will be very expensive in terms of both time and storage because of the large number of datapoints. With QTP, we only need to filter the input sample texture I_{in}. The low-resolution data can be extracted from the filtered I_{in}. As Figure 9 shows, a problem with the standard Gaussian pyramid is that a patch in the high-resolution image may not have a corresponding patch in the low-resolution image. QTP solves this problem by building a tree pyramid. The tree node in QTP is a pointer to an image and a tree level corresponds to a level in the Gauss pyramid. The root of the tree is the input sample texture I_{in}. When we move from one level of the pyramid to the next lower resolution level, we compute four children (lower-resolution images) with different shifts along the x- and y-directions as shown in Figure 9. With QTP constructed this way, each patch in the higher-resolution image I corresponds to a patch in a child of I. In our system, we use a three-level QTP. There are 1, 4, and 16 images of the size of the sample texture at levels 1, 2, and 3, respectively.

3.3 Principal Components Analysis

The approximate nearest neighbors search can be further accelerated by considering the special property of the input sample texture. Specifically, we reduce

the dimension of the search space using PCA [Jollife 1986]. Suppose that we need to build a kd-tree containing n datapoints $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where each $\mathbf{x}_i$ is a d-dimensional vector. PCA finds the eigenvalues and eigenvectors of the covariance matrix of these datapoints. The eigenvectors of the largest eigenvalues span a subspace containing the main variations of the data distribution. The datapoints $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are projected into this subspace as $\{\mathbf{x}'_1, \ldots, \mathbf{x}'_n\}$, where each $\mathbf{x}'_i$ is a d'-dimensional vector with $d' \ll d$. We choose the subspace dimension d' so that 97% of the variation of the original data is retained. For example, let the texture patch size be $w_B = 64$ and the boundary zone width be $w_E = 4$. If we analyze data at the top level of a three-level pyramid for the L-shaped boundary zone in Figure 3(d), the dimension of datapoints is $d = 3 \cdot (16 + 15) = 93$ and PCA typically reduces the dimension to about $d' = 20$.

4. RESULTS

We have tested the patch-based sampling algorithm on a wide variety of textures ranging from regular to stochastic. Figure 10 shows some typical results. All patch-based sampling results in this article are generated by the accelerated algorithm; the results by the unaccelerated algorithm of Section 2 do not look different visually and hence are not included. The time to synthesize a texture in Figure 10 is about 0.02 seconds on a 667 MHz PC. For space economy, the synthesized textures in Figure 10 are set to be about the same size as the sample textures. Figure 11 provides an example of synthesizing a large texture.

We examined documented cases for which Efros and Leung [1999], Wei and Levoy [2000], and Ashikhmin [2001] cease to be effective and have found that our algorithm continues to produce good results. As pointed out in Efros and Leung [1999], a problem with pixel-based nonparametric sampling techniques is that they tend to wander into the wrong part of the search space and grow garbage in the synthesized texture. The example in Figure 12(b) is taken from Efros and Leung [1999], which uses this example to demonstrate the garbage generated by pixel-based nonparametric sampling. The patches in our sampling scheme implicitly provide constraints for avoiding such garbage. The result of our patch-based sampling is shown in Figure 12(c). A texture synthesized by patch-based sampling can be divided into two types of areas: one includes the majority of the synthesized texture and is the middle part of a pasted texture patch; this type of area has no garbage. The other type of area is a blend of two boundary zones and cannot have garbage either because the boundary zones themselves have no garbage and the blending is done on boundary zones with matched features.

Another problem documented in Efros and Leung [1999] is verbatim copying of the sample texture at large scale, that is, the scale of the input sample texture. Figure 13(b), taken from Efros and Leung [1999], shows a large area of the sample texture directly copied onto the synthesized texture (see lower left corner). For patch-based sampling, verbatim copy at the scale of the patch size w_B is obviously unavoidable. However, with our algorithm it is easy to obtain textures that do not exhibit verbatim copying at large scale. Because

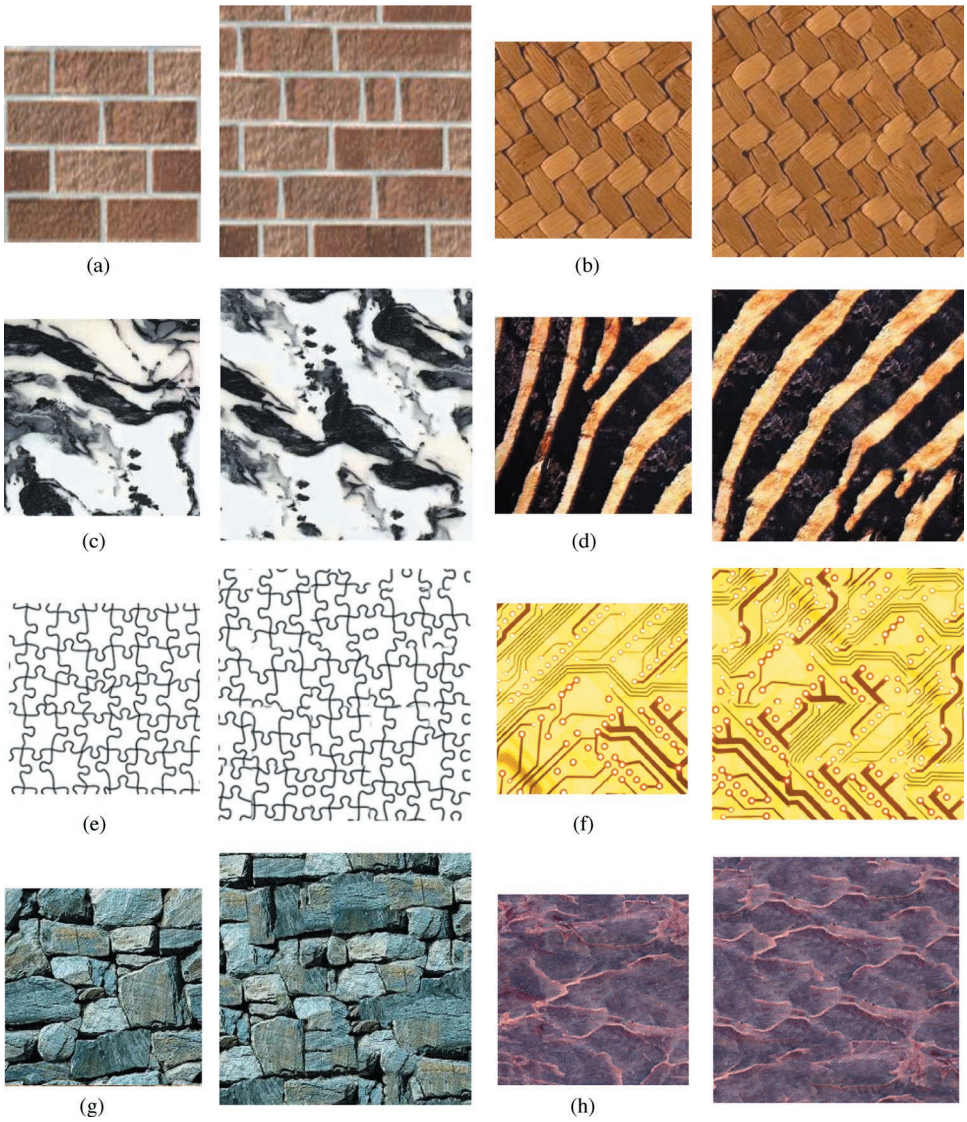


Fig. 10. Texture synthesis results. Each example includes the 200×200 input sample and 256×256 result. The time to synthesize a texture is about 0.02 seconds on a 667 MHz PC.

patch-based sampling is fast, the user can synthesize visually different textures at an interactive rate with the slightly different randomness parameter w_B. Most of these textures do not exhibit verbatim copying at large scale. Figure 13(c) and (d) are examples. In our experiment, we generated 200 textures with slightly different values of w_B and found that 93% of the synthesized textures had no verbatim copying at large scale.

The quality problems of Wei and Levoy's algorithm [2000] on a class of textures have been reported by Ashikhmin [2001]. He calls this class of textures "natural textures"; that is, textures "consisting of small objects of familiar but

Fig. 11. Texture synthesis from a 200×200 input sample to an 800×800 result. The time to synthesize the texture is about 0.65 seconds on a 667 MHz PC.

irregular shapes and sizes." Figure 14 compares Wei and Levoy's [1999] algorithm, Ashikhmin's [2001] algorithm, and patch-based sampling. The results of Wei and Levoy's algorithm were downloaded from their Web page. In the first two examples in Figure 14, "strawberry" and "pebbles," Wei and Levoy's algorithm produces poor results, whereas Ashikhmin's algorithm works well.

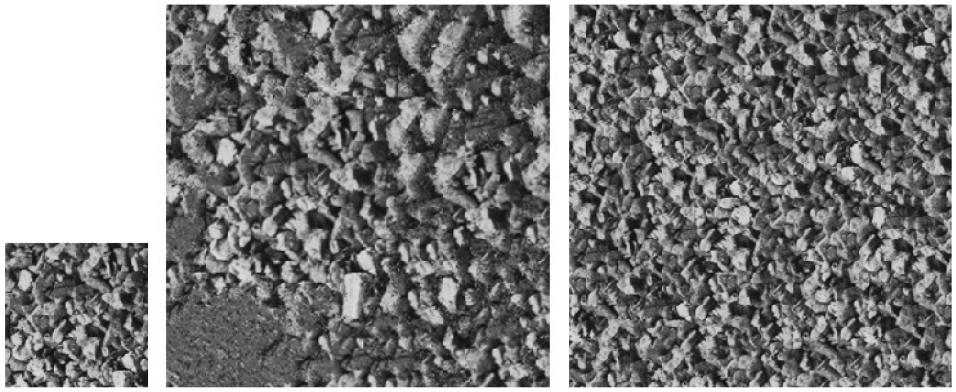


Fig. 12. Patch-based sampling continues to generate good results when pixel-based nonparametric sampling ceases to be effective: (a) input sample texture; (b) result by Efros and Leung [1999], showing the "garbage" generated by pixel-based nonparametric sampling. The result by patch-based sampling in (c) does not have this problem.

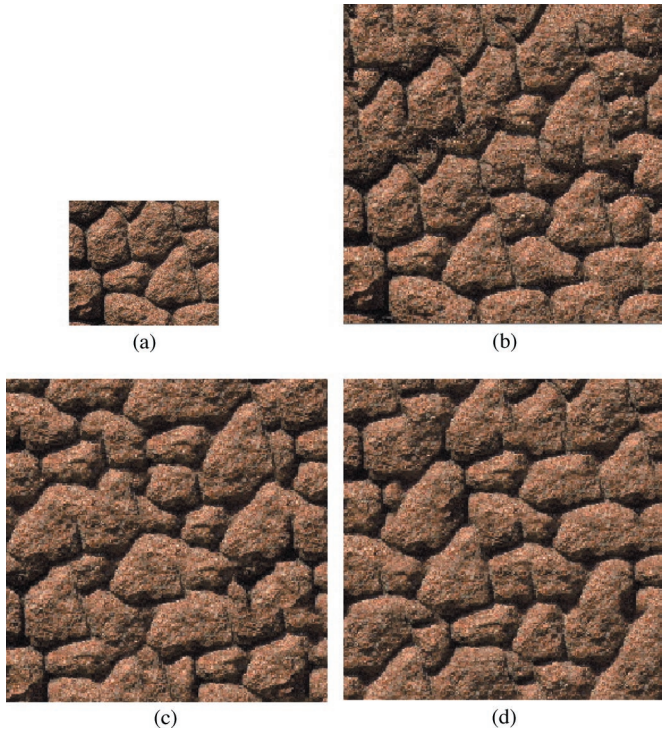


Fig. 13. (a) 132×110 input sample texture; (b) 216×216 result by Efros and Leung [1999], showing verbatim copying at large scale; (c) result of patch-based sampling with patch size $w_B = 32$; (d) result of patch-based sampling with patch size $w_B = 40$. Both (c) and (d) do not exhibit verbatim copying at large scale.

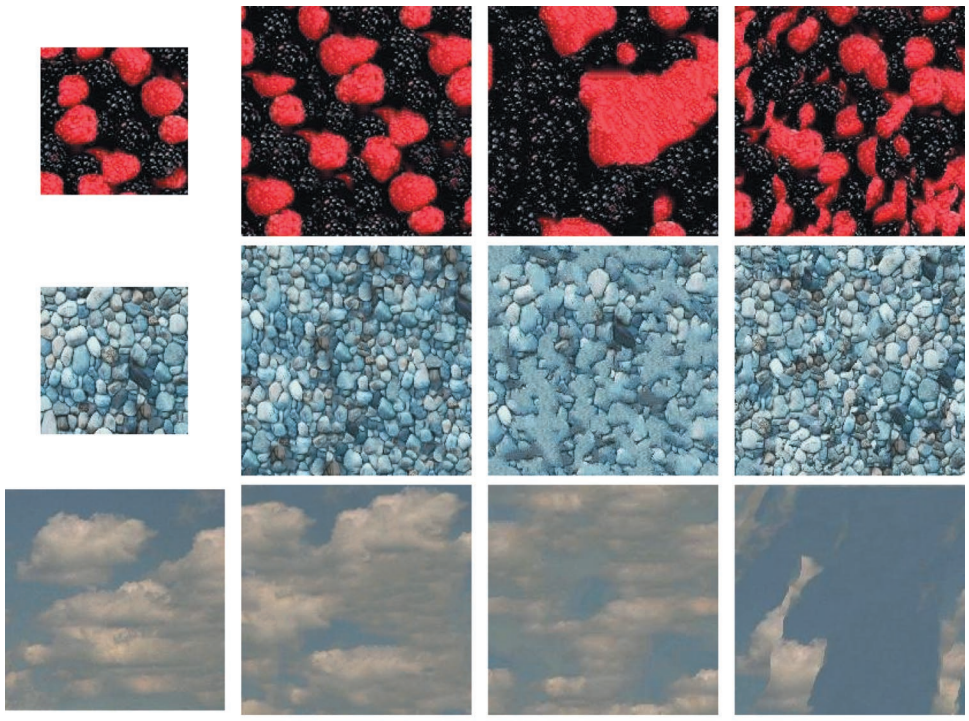


Fig. 14. Column (a): input sample textures; Column (b): results of patch-based sampling; Column (c): results of Wei and Levoy's algorithm; Column (d): results of Ashikhmin's algorithm. Results of Wei and Levoy's algorithm are taken from their Web page.

However, Ashikhmin's algorithm is a special-purpose algorithm designed for "natural textures"; it performs poorly for other textures, such as those that are relatively smooth or have more or less regular structures. In the third example in Figure 14, "clouds," Ashikhmin's algorithm fails while Wei and Levoy's algorithm generates a good texture. The patch-based sampling works well in all three examples and is about an order of magnitude faster than Ashikhmin's algorithm.

Figure 15 compares patch-based sampling with our earlier patch-pasting algorithm [Xu et al. 2000]. The patch-pasting algorithm works well on stochastic textures such as the example shown in the top row of Figure 15. However, when the sample texture has a more or less regular structure (e.g., a brick wall) the patch-pasting algorithm fails to produce good results because of mismatched features across patch boundaries. See the middle and bottom rows of Figure 15 for examples. The patch-based sampling algorithm works well on all examples.

Figure 16 compares patch-based sampling with concurrent work by Efros and Freeman [2001] on texture quilting. In this comparison, we are mainly interested in comparing texture quality for feathering, used by patch-based sampling, and the minimum error boundary cut (MEBC) used by texture quilting.

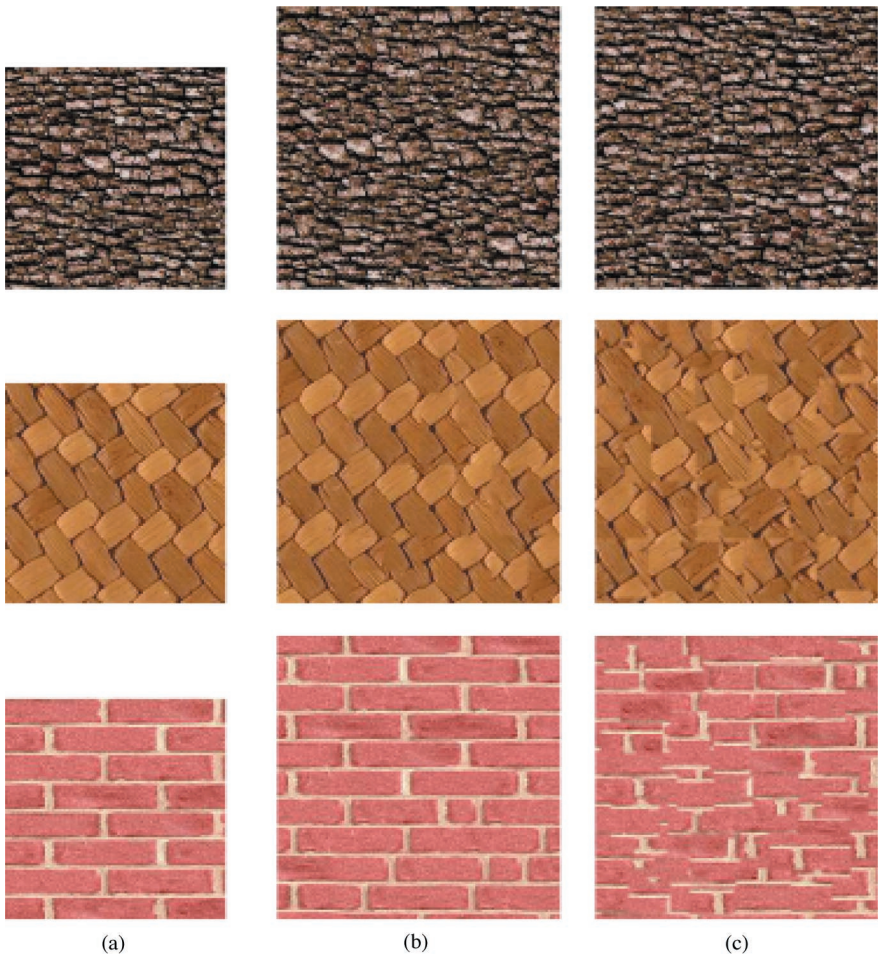
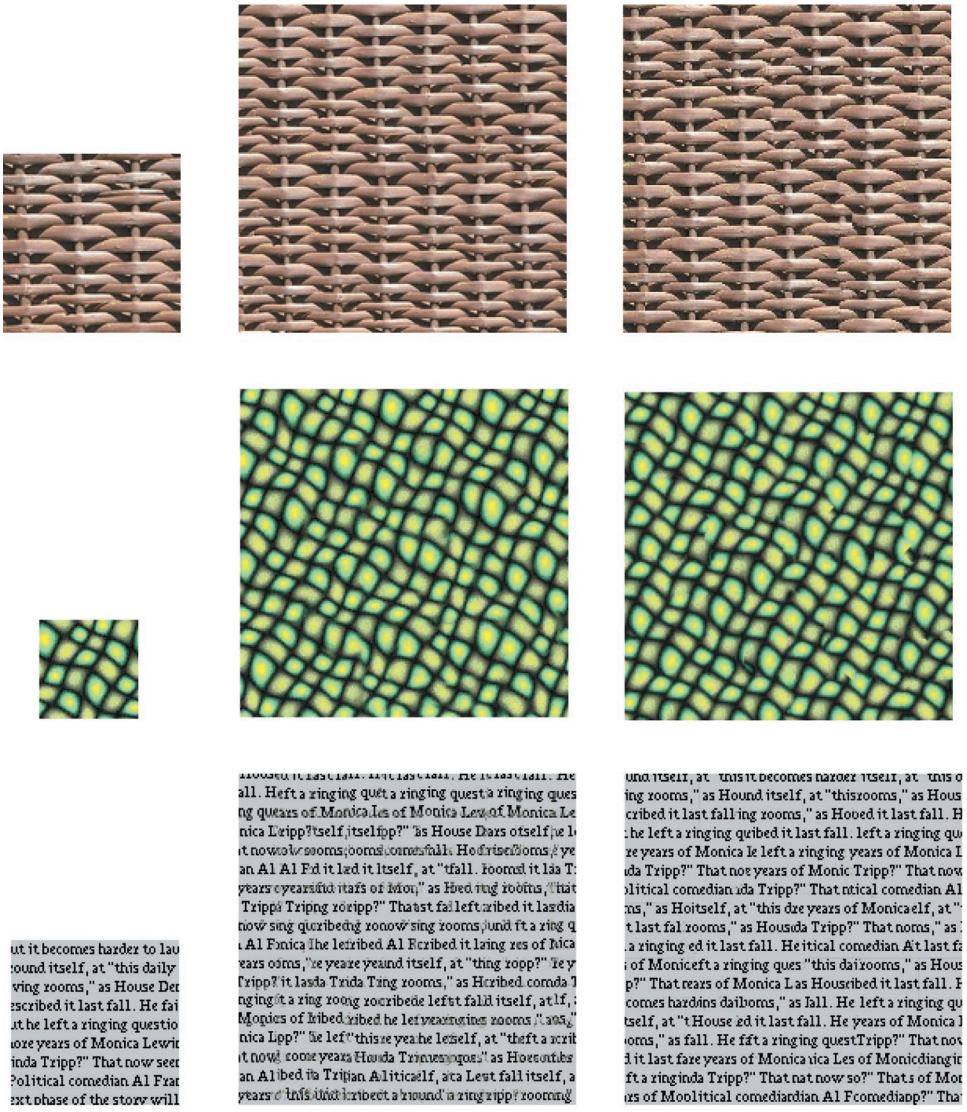


Fig. 15. Column (a): input sample textures; Column (b): results of patch-based sampling; Column (c): results of patch pasting.

We found that feathering produces more smooth color changes than MEBC, which can generate abrupt color changes at various places along the boundary cut. See the top and middle rows of Figure 16 for examples. However, there are examples in which MEBC produces better results than feathering. For the text shown in the bottom row of Figure 16, feathering produces a smeary effect that is less apparent in the result obtained with MEBC.

Table I summarizes the performance of the patch-based algorithm with and without acceleration techniques applied. The table also compares the speed of patch-based sampling with Heeger's algorithm [Heeger and Bergen 1995] and Wei and Levoy's [2000] algorithm. We choose to compare timing with Heeger's algorithm and Wei and Levoy's algorithm because they are fast general-purpose texture synthesis algorithms. The timings for the patch-based sampling algorithm and Heeger's algorithm are measured by averaging the times of



(a) (b) (c)

Fig. 16. Column (a): input sample textures; Column (b): results of patch-based sampling; Column (c): results of Efros and Freeman's algorithm.

500 trial runs with different textures. Timings are also taken from Wei and Levoy [2000].

Table I also demonstrates three possible ways to combine our acceleration techniques. The QTP method uses QTP to filter all datapoints into low resolution. Then exhaustive search is invoked for finding the initial candidates in low resolution and for choosing the ANNs from the initial candidates in high resolution. The QTP + KDtree method is similar to the QTP method; the

Table I. Timing Comparison: Patch-Based Sampling and Other Algorithms*

Method	Analysis Time	Synthesis Time
QTP + KDTree + PCA	0.678	0.020
QTP + KDTree	0.338	0.044
QTP	0.017	0.256
Exhaustive**	0.000	1.415
Heeger	0.0	32
Wei and Levoy***	22.0***	7.5***

*Timings are measured in seconds on a 667 MHz PC for synthesizing 200 × 200 textures from 128 × 128 samples.
**Exhaustive means no acceleration is used.
***Wei and Levoy's timings are taken on a 195 MHz R10000 processor.

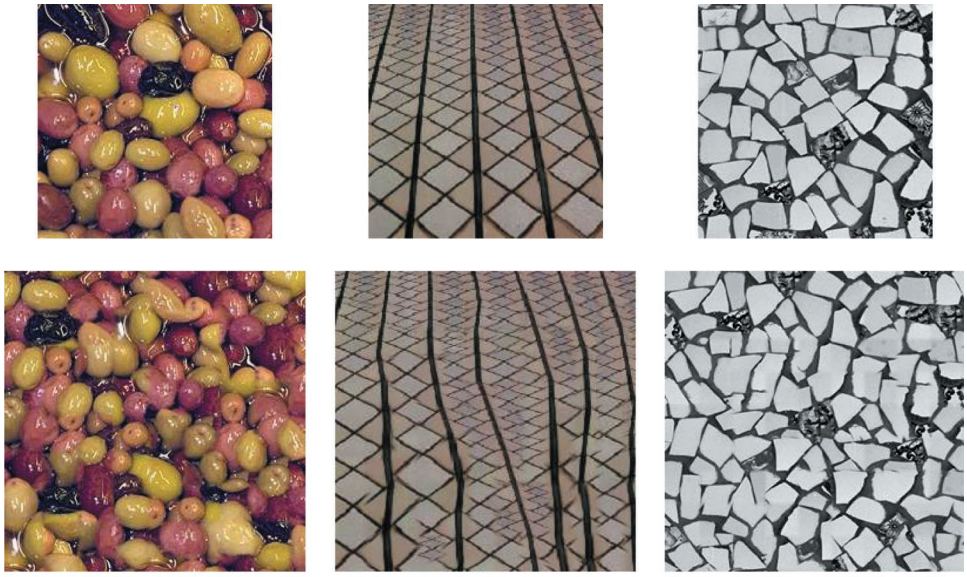


Fig. 17. Some failure examples of the patch-based sampling. The input sample textures are shown in the top row. The synthesized textures are in the bottom row.

only difference is that a kd-tree is built when searching for the initial candidates among the low-resolution datapoints. The QTP + KDtree + PCA method further accelerates the QTP + KDtree method by projecting all low-resolution datapoints into a low-dimensional subspace when searching for the initial candidates using a kd-tree.

Finally, Figure 17 shows some failure examples of the patch-based sampling. In column (a), we have a sample texture of easily recognizable objects. Since our texture synthesis algorithm has no explicit model for these objects, objects in the synthesized texture do not always resemble those in the sample texture. Like most existing texture synthesis techniques, patch-based sampling can only handle frontal-parallel textures. In column (b), the sample texture is not frontal parallel and the synthesized texture is of poor quality. In Figure 17, column (c),

we see interrupted object boundaries in the synthesized texture. This is because the distance metric for the patch boundary zone matching gives no preference to object boundaries or any other high-frequency features. A possible future research topic is to develop a better distance metric to improve the continuity of high-frequency features across patch boundaries.

5. CONCLUSION

We have presented a patch-based sampling algorithm for texture synthesis. Our algorithm synthesizes high-quality textures for a wide variety of textures ranging from regular to stochastic. The algorithm is fast enough to enable real-time texture synthesis on a midlevel PC. For generating textures of the same size and comparable (or better) quality, our algorithm is orders of magnitude faster than existing texture synthesis algorithms. Patch-based sampling combines the strengths of nonparametric sampling [Efros and Leung 1999; Wei and Levoy 2000] and patch pasting [Xu et al. 2000]. In fact, both patch-pasting and the pixel-based nonparametric sampling [Efros and Leung 1999; Wei and Levoy 2000] are special cases of the patch-based sampling algorithm. We examined documented cases for which Efros and Leung [1999], Wei and Levoy [2000], and Ashikhmin [2001] cease to be effective and our results indicate that patch-based sampling continues to produce good results in these cases. In particular, the texture patches in our sampling scheme provide implicit constraints for avoiding garbage as found in some textures synthesized by Efros and Leung [1999].

For future work, we are interested in extending the ideas presented here for texture synthesis on surfaces of arbitrary topology [Wei and Levoy 2001; Turk 2001; Ying et al. 2001]. With patch-based sampling, we can eliminate a number of problems with existing techniques, for example, the need for manual texture patch creation and feature mismatches across patch boundaries [Praun et al. 2000]. Other interesting topics include texture mixtures and texture movie synthesis [Heeger and Bergen 1995; Bar-Joseph et al. 2001].

ACKNOWLEDGMENTS

Many thanks to the anonymous TOG reviewers for their constructive critique. This article is a revised version of our ill-fated SIGGRAPH '01 submission (available as a technical report Liang et al. 2001) and we want to thank SIGGRAPH '01 reviewers for their critique. A special thanks to Dr. Song-Chun Zhu for useful discussions on many texture-related issues.

REFERENCES

ARYA, S., MOUNT, D. M., NETANYAHU, N. S., SILVERMAN, R., AND WU, A. Y. 1998. An optimal algorithm for approximate nearest neighbor searching. *J. ACM 45*, 891–923.

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the ACM Symposium on Interactive 3D Graphics* (March), 217–226.

BAR-JOSEPH, Z., EL-YANIV, R., LISCHINSKI, D., AND WERMAN, M. 2001. Texture mixing and texture movie synthesis using statistical learning. *IEEE Trans. Vis. Comput. Graph.*

De Bonet, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture image. In *Computer Graphics Proceedings*, *Annual Conference Series* (August), 361–368.

Efros, A. and Freeman, W. 2001. Image quilting for texture synthesis and transfer. In *Computer Graphics Proceedings*, *Annual Conference Series* (August).

Efros, A. A. and Leung, T. K. 1999. Texture synthesis by non-parametric sampling. In *Proceedings of International Conference on Computer Vision*.

Fournier, A., Fussell, D., and Carpenter, L. 1982. Computer rendering of stochastic models. *Commun. ACM 25*, 6 (June), 371–384.

Friedman, J., Bentley, J., and Finkel, R. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw. 3*, 3, 209–226.

Heeger, D. J. and Bergen, J. R. 1995. Pyramid-based texture analysis/synthesis. In *Computer Graphics Proceedings*, *Annual Conference Series* (July), 229–238.

Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., and Salesin, D. 2001. Image analogies. In *Computer Graphics Proceedings*, *Annual Conference Series* (August).

Iversen, H. and Lonnestad, T. 1994. An evaluation of stochastic models for analysis and synthesis of gray scale texture. *Pattern Recogn. Lett. 15*, 575–585.

Jollife, I. T. 1986. *Principal Component Analysis*. Springer-Verlag, New York.

Lewis, J.-P. 1984. Texture synthesis for digital painting. In *Comput. Graph.* (*SIGGRAPH '84 Proceedings*) *18*, 245–252.

Liang, L., Liu, C., Xu, Y. Q., Guo, B., and Shum, H. Y. 2001. Real-time texture synthesis by patch-based sampling. Microsoft Research Tech. Rep. MSR-TR-2001-40, March.

Mount, D. M. 1998. *ANN Programming Manual*. Department of Computer Science, University of Maryland, College Park, Maryland.

Nene, S. A. and Nayar, S. K. 1997. A simple algorithm for nearest-neighbor search in high dimensions. *IEEE Trans. PAMI 19*, 9 (Sept.), 989–1003.

Perlin, K. 1985. An image synthesizer. *Comput. Graph.* (*Proceedings of SIGGRAPH '85*) *19*, 3 (July), 287–296.

Popat, K. and Picard, R. W. 1993. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Proceedings of SPIE Visual Communication and Image Processing*, 756–768.

Portilla, J. and Simoncelli, E. 1999. Texture modeling and synthesis using joint statistics of complex wavelet coefficients. In *Proceedings of the IEEE Workshop on Statistical and Computational Theories of Vision*.

Praun, E., Finkelstein, A., and Hoppe, H. 2000. Lapped texture. In *Computer Graphics Proceedings*, *Annual Conference Series* (July), 465–470.

Szeliski, R. and Shum, H.-Y. 1997. Creating full view panoramic mosaics and environment maps. In *Proceedings of SIGGRAPH '97* (August), 251–258.

Turk, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. In *Computer Graphics* (*SIGGRAPH '91 Proceedings*) *25* (July), 289–298.

Turk, G. 2001. Texture synthesis on surfaces. In *Computer Graphics Proceedings*, *Annual Conference Series* (August).

Wei, L. and Levoy, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *Computer Graphics Proceedings*, *Annual Conference Series* (August).

Wei, L. Y. and Levoy, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Computer Graphics Proceedings*, *Annual Conference Series* (July), 479–488.

Witkin, A. and Kass, M. 1991. Reaction-diffusion textures. In *Computer Graphics* (*SIGGRAPH '91 Proceedings*), *25,* (July), 299–308.

Worley, S. P. 1996. A cellular texturing basis function. In *SIGGRAPH'96 Conference Proceedings*, Holly Rushmeier, Ed., Annual Conference Series (August), 291–294.

Wu, Y. N., Zhu, S. C., and Liu, X. W. 2000. Equivalence of Julesz ensemble and FRAME models. *Int. J. Comput. Vis. 38*, 30, 245–261.

Xu, Y. Q., Guo, B., and Shum, H. Y. 2000. Chaos mosaic: Fast and memory efficient texture synthesis. Microsoft Res. Tech. Rep. MSR-TR-2000-32, April.

Ying, L., Hertzmann, A., Biermann, H., and Zorin, D. 2001. Texture and shape synthesis on surfaces. In *Proceedings of the Twelfth Eurographics Workshop on Rendering* (June).

ZHU, S. C., LIU, X., AND WU, Y. 2000. Exploring texture ensembles by efficient Markov chain Monte Carlo. *IEEE Trans. PAMI 22*, 6.

ZHU, S. C., WU, Y., AND MUMFORD, D. B. 1997. Minimax entropy principle and its application to texture modeling. *Neural Comput.* 9, 1627–1660 (first appeared in CVPR '96).

ZHU, S. C., WU, Y., AND MUMFORD, D. 1998. Filters, random-fields and maximum-entropy (Frame). *Int. J. Comput. Vis. 27*, 2 (March), 107–126.

ZUCKER, S. AND TERZOPOULOS, D. 1980. Finding structure in co-occurence matrices for texture analysis. *Comput. Graph. Image Process. 12*, 286–307.

Received May 2001; revised August 2001; accepted August 2001