On Pixel-Based Texture Synthesis by Non-parametric Sampling

Seunghyup Shin Sung Yong Shin

CS/TR-2004-196

January 26, 2004

K A I S T
Department of Computer Science

On Pixel-Based Texture Synthesis by Non-parametric Sampling

Seunghyup Shin Sung Yong Shin

Figure 1: Iterative texture refinement

Abstract

In this paper, we propose a pixel-based method for texture synthesis with non-parametric sampling. On top of the general framework of pixel-based approaches, our method has three distinguishing features: window size estimation, seed point planting, and iterative refinement. The size of a window is estimated to capture the structural components of the dominant scale embedded in the texture sample. To guide the pixel sampling process at the initial iteration, a grid of seed points are sampled from the example texture. Finally, an iterative refinement scheme is adopted to diffuse the non-stationality artifact over the entire texture. Our objective is to enhance texture quality as much as possible with a minor sacrifice in efficiency in order to support our conjecture that the pixel-based approach would yield the best quality.

1 Introduction

Texture synthesis by example has recently been investigated extensively in computer vision and computer graphics. This problem is stated as follows: Given a texture sample, synthesize a (tilable) new texture of an arbitrary size such that it is perceptually similar to the texture sample. The notion of perceptual similarity is well explained in [7, 18]. Rich results have been reported as solutions to the texture

synthesis problem [8, 14, 18, 2, 6, 11]. In particular, pixel-based, non-parametric methods [7, 18, 2] have drawn much attention.

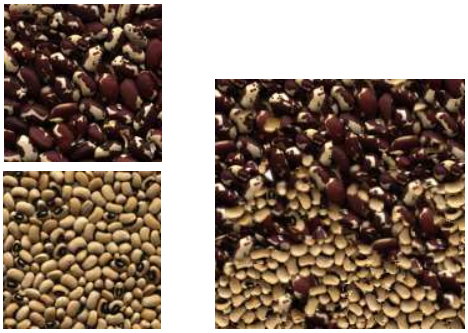
Relying on a simple strategy of copying one pixel at a time, these techniques have demonstrated their surprising capability of synthesizing a wide variety of high quality textures ranging from regular to stochastic. However, resulting textures have sometime shown visual artifacts such as "blurring" and "garbage growing" [7, 2, 18]. That is, a pixel-based method has a tendency to blur features or to grow small-scale structures in synthesized textures. As pointed out in [12], such a method also suffers from heavy searching time in sampling the pixel values from the input sample texture.

To remedy those drawbacks in both texture quality and time efficiency, patch-based methods have been proposed [6, 12, 11]. Unlike pixel-based methods, patch-based methods copy a patch of pixels at a time to show real-time performances in texture synthesis. Moreover, by copying a cluster of spatially coherent pixels simultaneously, the latter methods apparently remove visual artifacts such as blurring and garbage growing. However, a closer look at the resulting textures sometimes reveals at least two new types of artifacts, instead: texture discontinuity and repetition. The discontinuity artifact is caused when a texture sample exhibits a smooth spatial variation with no high frequency components. The other artifact is observed when a verbatim copy of a patch in the texture sample is transferred to a synthesized texture. Both of these artifacts may result from the lack of randomness of the patch-based copying strategy.

To introduce enough randomness into a texture while adopting the patch-based strategy, either patches must be sufficiently small and irregular, or patch-copying operations must be repeated sufficiently often to eventually remove all seams and patch repetitions. Then, the limiting behavior would be reduced to that of a pixel-based scheme. This conjecture naturally raises an interesting question: Are problems such as blurring and garbage growing indeed inherent in pixel-based schemes? As expected, our answer is negative as we will explain in later sections.

In this paper, we propose a novel pixel-based method which exhibits neither blurring nor garbage growing while not introducing any new artifacts. Our objective is to enhance texture quality with a little sacrifice in time complexity. On the top of the general framework of the pixel-based paradigm, our method is equipped with three distinguishing features: window size estimation, seed point planting, and iterative refinement. The window size is estimated to capture the structural texture components of the dominant scale. After initially sampling a grid of seed points, the method iteratively refines the output texture synthesized at the previous step (See Figure 1) until the termination criteria are satisfied.

As an application of our method, we pose an interesting problem called texture



(a) Texture samples (b) An output texture

Figure 2: Texture fusion

fusion: Given two or more sample textures, synthesize a new seamless texture of an arbitrary size with its designated parts similar to the input sample textures, guided by a user-specified map. After obtaining the new texture samples for texture fusion, our method can solve this problem to yield interesting textures as illustrated in Figure 2.

The remainder of this paper is organized as follows: After reviewing related work in Section 2, we present a novel texture synthesis method in Section 3. In Section 4, we apply our method to fuse multiple sample textures. We show experimental results in Section 5. Finally, Sections 6 and 7 provide discussion and conclusions, respectively.

2 Related Work

Rich results have been reported in texture analysis and synthesis. For our purposes, we focus on research results on example-based texture synthesis by non-parametric sampling, which are directly related to our work. Methods in this category model an (infinite) texture by Markov Random Fields (MRF). A texture is synthesized by simply copying pixels from a texture sample based on their local similarity. An underlying assumption is that the finite texture sample reflects the statistical properties of an MRF such as stationality and locality. Depending on the method of copying (sample) pixels from the texture sample, we distinguish two classes of methods.

Pixel-based methods: This class of methods adopts the strategy of copying one pixel at a time. Efros and Leung [7] initiated this strategy and demonstrated its

power of probability sampling by synthesizing high quality textures. They also reported their experience that the method sometimes causes visual artifacts such as garbage growing and verbatim copying. We believe that these artifacts result for two reasons: For the former artifact, the method lacks an additional guide for sampling a pixel even though only partial neighbors of the pixel are available. For the latter artifact, the window size probably might be too large to introduce enough randomness. In addition, the method required heavy computation time because of exhaustive pixel searching.

Wei and Levoy [18] tried to improve the previous method [7] in at least two directions: To guide the pixel sampling process beyond the already-generated neighbors of each pixel, they proposed a multi-resolution scheme. This scheme also exhibited the effect of enlarging the window size. In fact, exploiting the full neighbors at coarser levels, their method sampled some types of structural components relatively well, even with a small window size. To accelerate the pixel searching process, they adopted a heuristic called tree-structured vector quantization (TSVQ). Although the TSVQ greatly enhanced time efficiency, it also brought in the blurring artifact, which may obscure the quality enhancement obtained from the multi-resolution searching.

Ashikhmin [2] exploited spatial coherency to remedy the blurring artifact observed in a texture generated by the method of Wei and Levoy. The latter method tries to sample a better pixel from scratch at every pixel to be synthesized. The search domain covers the entire texture sample although it is reduced quickly due to the TSVQ technique. Ashikhmin restricted the search domain based on the observation that the pixel to be synthesized is spatially coherent with its already-generated neighbors. Therefore, given their positions in the texture sample, the candidates can be found by properly shifting these positions according to their displacements with respect to this pixel. Thus, Ashikhmin's method tends to show a patch-copying behavior and solves the blurring artifact as intended. This method also exhibited better time efficiency than that of Wei and Levoy even without employing the TSVQ acceleration. However, the method sometime introduced visual artifacts such as discontinuity and garbage growing due to its restricted search space.

Patch-based Methods: Unlike pixel-based methods, patch-based methods copy a patch of pixels at a time. These methods exploited spatial coherency in one form or another to accelerate their synthesis performances as well as to enhance texture quality. In this sense, Ashikhmin's work [2] can be considered as a "bridge" to the patch copying paradigm although Xu et al. [20] used texture patches even earlier

for texture synthesis.

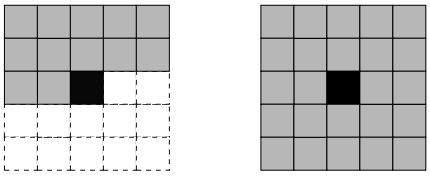
Efros and Freeman [6] presented a patch-based method called "image quilting." They observed that, in pixel-based texture synthesis, much effort is waisted on searching pixels, the positions of which can be derived trivially from their already-synthesized neighbors based on spatial coherency. Therefore, the granulity of synthesis was enlarged to be an aggregation of connected pixels contained in a patch of the texture sample. Their method first tiles the output texture with a set of overlapping square blocks (obtained from the texture sample) such that each block matches its neighbors along the overlapping regions. Then, each pair of adjacent blocks are stitched along the minimum cost path passing through their overlapping region.

Liang et al. [12] proposed a similar method. Their method used feathering instead of quilting to combine a pair of blocks. Based on an optimized kd-tree [1, 13] and principal component analysis (PCA) [9], the most notable feature of this method is its real-time performance while not introducing the blurring artifact. We will adopt their idea for pixel sampling.

In addition to performance acceleration, both methods [6, 12] were intended to enhance texture quality, in particular, removing the garbage growing observed in pixel-based methods [7, 18, 2]. However, these methods sometimes produced visual artifacts such as texture discontinuity and verbatim copying. In addition, neither of the methods was able to set the block size automatically. Cohen et al. [4] presented a stochastic method to tile the plane non-periodically with a small set of edge-constrained tiles called Wang tiles. Our seed point planting scheme is inspired by this work.

Recently, Kwatra et al. [11] proposed a new patch-based method to fix the problems of previous methods [6, 12]. Their patch-copying process is performed in two stages: First, the sample texture is placed at a proper location in the output texture synthesized at the previous step, which can be accelerated using a fast Fourier transform technique. Then, the optimal seam is searched within the region of overlap to minimize visual discontinuity using a graph-cut technique.

Unlike others, this method iteratively refines the texture quality without requiring a-priori knowledge on the patch size and demonstrated rather superb performance in both quality and efficiency. However, a small number of iterations may result in similar artifacts as observed with the previous patch-based methods. Intuitively, the limiting behavior of this method would be similar to a pixel-based paradigm due to a certain degree of randomness picked up at every step. Thus, excessive iterations would not only destroy the texture structure but also require heavy computation time. Unfortunately, it is hard, if not impossible, to find the proper termination criteria for automatic texture synthesis.



(a) L-shape window (b) Square window

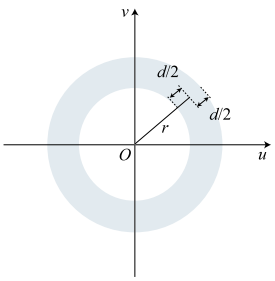
Figure 3: Window shapes

3 Texture Synthesis

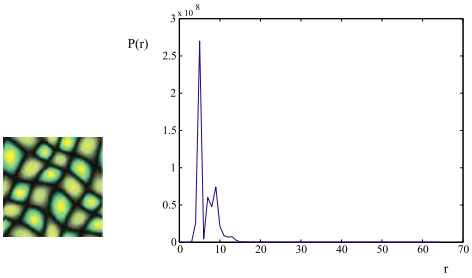
Overview: Our example-based texture synthesis method includes two major stages: texture modeling and texture sampling. The former is concerned with how to estimate the MRF underlying an ideal texture using a finite texture sample, and the latter deals with how to efficiently sample the pixels from the estimated MRF to synthesize an output texture. We first provide the skeleton of our method and then describe important steps:

```
    function TextureSynthesis (TextureSample)
    {
1       WindowSize ← 2 * ComputeDominantScale(TextureSample);
2       SearchTree ← BuildSearchTree(TextureSample);
3       OutputTexture ← PlantSeed(WindowSize);
4       ContinueFlag ← True;
5       while (ContinueFlag)
        {
6           CurrentOutput ← RefineTexture(OutputTexture);
7           if (CurrentOutput satisfies the termination criteria)
8               ContinueFlag ← False;
9           OutputTexture ← CurrentOutput;
        }
10      return OutputTexture;
    }
```

The modeling stage consists of window size estimation (step 1), search tree construction (step 2), and seed point planting (step 3) which determines the (full) neighbors of a pixel, accelerates pixel sampling, and initializes the output texture, respectively. The sampling stage is the main loop that iteratively refines texture quality (steps 4-9). At each iteration, the texture is refined in the scan-line order, that is, top-to-bottom and left-to-right order.



(a) Integration along a ring region



(b) A histogram $P(r)$ of a 64×64 texture sample. In this example, the dominant frequency is 6 and the dominant texture scale is $11(\approx 64/6)$

Figure 4: Texture scale estimation

Window Size Estimation: Our method adopts the pixel-based paradigm with non-parametric probability sampling. In this paradigm, the conditional probability of a pixel is solely dependent on its neighbors, which are contained in the window centered on the pixel. Therefore, the size of the window needs to be estimated to determine the neighbors (Figure 3). L-shaped windows have been used to reflect the availability of the neighbors when textures are synthesized in the scan-line order [18, 2]. To refine a texture iteratively, we use a square window although we also adopt the scan-line order.

In addition to probability sampling, the window also plays a key role in capturing the structure of a texture. Therefore, it should be at least as large as the scale of the most dominant structural component in the texture. Otherwise, the neighbors would not properly reflect the structural information. We set the window size to be twice as large as the dominant scale. The dominant scale also determines the density of the seed points, which initially guide the iterative texture refinement. Empirically, we set the density of the seed points to be the same as this scale (See details in Section 6). Now, the problem is estimating the dominant scale.

We adopt the statistical texture analysis based on a Fourier power spectrum [19, 16]. For an $N \times N$ texture sample $f(x, y)$, let $F(u, v)$ be its discrete Fourier transform:

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi\sqrt{-1}(ux/N+vy/N)}.$$

The Fourier power spectrum $P(u, v)$ represents the strength of each spatial fre-

quency and is defined by

$$P(u, v) = |F(u, v)|^2.$$

By representing the power spectrum $P(u, v)$ as its equivalent $P(r, \theta)$ in the polar coordinates and integrating them within a ring region (Figure 4(a)), we obtain $P(r)$ which is a sum of the strengths around the frequency r:

$$P(r) = \sum_{r_x=r-d/2}^{r+d/2} \sum_{\theta=0}^{\pi} P(r_x, \theta).$$

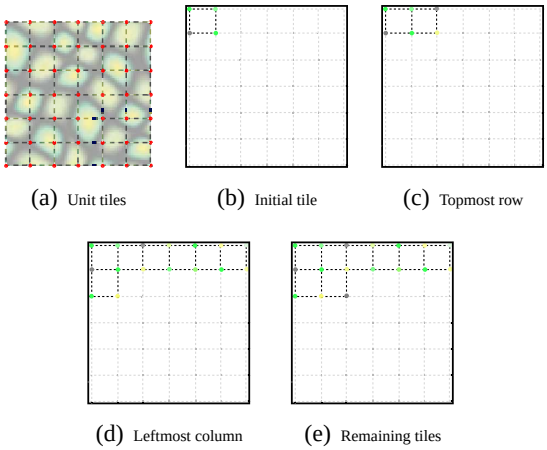
Here, d is the thickness of the ring region and was set to be 0.1 in our experiments. We compute the *dominant frequency* of the texture sample by selecting $\hat{r}$ whose corresponding $P(\hat{r})$ is the largest (Figure 4(b)). Finally, the *dominant texture scale* s is obtained by dividing the texture size N with the dominant frequency. That is, $s = N/\hat{r}$, where $\hat{r} = \arg\max_r P(r)$.

For non-structural textures, $P(r)$ may not have a prominent peak as shown in Figure 4(b), and thus one might claim that the estimated scale is not so meaningful. However, the role of the scale is not only to capture the structure of the texture sample but also to obtain an initial guess for the iterative texture refinement. For a non-structural texture, the dominant scale can still be used as the sampling density to construct the initial guess.

Search Tree Construction: Given the output pixel to be refined and its full neighbors, we need to search the pixel from the texture sample, whose neighbors are most similar to those of the output pixel. Considering the neighbors of a pixel as a k-dimensional vector where k is three times as large as the number of the neighbors of the pixel, we can formulate the pixel search as a k-dimensional nearest search problem as suggested in [18].

Inspired by the work in [12], we also use a kd-tree and PCA to accelerate the search while avoiding texture blurring. We do not adopt the multi-resolution texture synthesis scheme by Wei and Levoy [18] since we take an iterative refinement paradigm unlike their scheme. In addition, our seed point planting scheme is powerful enough to guide the initial iteration of the refinement process.

The kd-tree greatly accelerates the pixel search while guaranteeing the exact solution. PCA further accelerates the search by reducing the dimensionality of the search space with a negligible sacrifice in quality. We refer readers to the work in [12] for details.



(a) Unit tiles (b) Initial tile (c) Topmost row

(d) Leftmost column (e) Remaining tiles

Figure 5: Planting seed points

Seed Point Planting: Different texture synthesis methods may take different orders in pixel sampling. Regardless of pixel sampling orders to be used, however, about one half of the full neighbors of the current output pixel have not yet become available. With little information on unavailable neighbors, the structure of a texture may not be captured properly. Unlike the previous methods, we initially plant a grid of seed points covering the output texture region, which are sampled from the texture sample. To hopefully capture the structure of the texture sample, we set the sampling density of seed points as equal to the estimated dominant texture scale.

Inspired by the idea of stochastic tiling [4], our scheme plants the seed points on a square grid covering the output texture region in the scan-line order. Let a *unit tile* be a square with an edge length equal to the texture scale such that its four corner points lie on grid points. Also, we denote its upper, right, left, and lower sides by N, E, W, and S, respectively. We divide the texture sample into a set of unit tiles as illustrated in Figure 5(a).

To the top row, the slot at the top-left corner of the output texture is initially tiled by a randomly-selected unit tile (Figure 5(b)). Each remaining slot in the first row is tiled one by one by selecting a unit tile at a time such that the W edge of the new tile matches the E edge of its left one (Figure 5(c)). A pair of edges are said to match when the colors of their corresponding corner points are similar. To avoid the verbatim copying problem, we randomly select a unit tile among the k best matching ones (We empirically set k=5). When placing a unit tile on the output texture, we sample the values of two seed points on the E edge since those on the W edge have already been obtained.

From the second row, we slightly modify our sampling policy since more information on new output tiles are available. For the leftmost tile on the output region, we select a unit tile whose N edge matches the S edge of its upper tile (Figure 5(d)). We sample the values of the seed points on the S edge from the corresponding points of a unit tile randomly chosen among the k best matching tiles. For each of the remaining output tiles, we find the k best matching unit tiles such that the W and N edges of every tile match the E edge of its left tile and the S edge of its upper tile, respectively, to randomly choose a tile from which the value of the bottom-right point is sampled (Figure 5(e)).

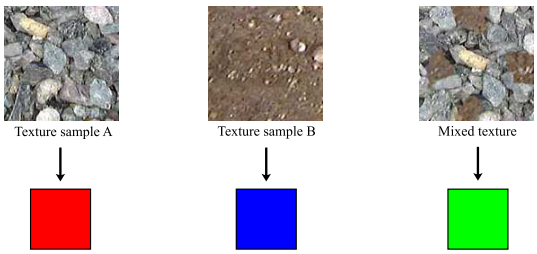
Iterative Refinement: As described clearly in [7, 18], the notion of perceptual similarity includes both stationality and locality. Exploiting the locality property for sampling efficiency, the previous methods [7, 18, 2] have exhibited this property rather well in synthesized textures. However, the stationality property has not been addressed.

The main sources of the non-stationality are the unavailability of the full neighbors of a pixel and the inhomogeneity of the neighborhood near the boundary, together with the improper setting of the window size. This effect has appeared in synthesized textures in various forms of visual artifacts such as garbage growing, discontinuity, and blurring.

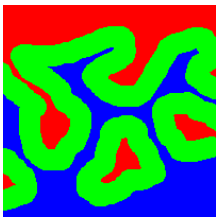
Assuming that the window size is set properly, the texture synthesis process can be enhanced to hopefully remove these artifacts. The key idea of this enhancement is to use the output texture at the previous iteration as the input texture at the current iteration. A similar idea was attempted in a different way (coherent search), with a different motivation (user control) by Ashikhmin [2]. The rationale for our move is that the full neighbors are available at every pixel in an output texture with possible wrapping around near the boundary. Therefore, our texture synthesis method refines the output texture iteratively to diffuse the non-stationality artifact over the entire texture with the support of the full neighbors of every pixel. In this sense, the output textures with the previous methods can be interpreted as the intermediate results after the first iteration.

Taking a grid of seed points as an initial guess, the output texture is iteratively refined until the termination criteria are satisfied. In the first iteration, only the seed points together with the newly-synthesized pixels in the window of each output pixel are used for the pixel search, and the rest of them are masked out in neighborhood comparison. From the second iteration, the full neighbors take part in the search.

We have tried two different criteria: the number of iterations and the texture



(a) Texture samples and a mixed texture with their corresponding colors



(b) A fusion map

(c) An output texture

Figure 6: Texture fusion

difference from the previous iteration. For the former, the maximum number of iterations is preset. Empirically, visually-good textures have been produced within ten iterations. For the latter, we define the texture difference d between a pair of consecutively-synthesized textures, f_{i-1} and f_i as follows:

$$d(f_{i-1}, f_i) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \frac{\|f_i(x,y) - f_{i-1}(x,y)\|}{\|f_{i-1}(x,y)\|}, \tag{1}$$

where $\|\cdot\|$ is a Euclidean norm. If $d(f_i, f_{i-1})$ is smaller than a user-specified threshold ϵ, then we stop the iterative refinement. Empirically, our method has worked well with $\epsilon = 0.01$. In either case, our texture refinement needs multiple iterations, which can be accelerated greatly with the help of the kd-tree and PCA.

4 Texture Fusion

The origin of our texture fusion problem is probably traced back to the constrained texture synthesis problem [7, 18]. Ashikhmin [2] addressed a more general problem of synthesizing a texture guided by a user-provided map. Recently, Kwatra et al. [11] showed how to merge multiple textures interactively by extending their

texture synthesis method based on a graph cut technique. The texture fusion problem is to merge multiple textures seamlessly with the guidance of a user-provided *fusion map*. We address this problem by combining our texture synthesis method and the graph cut technique. The Figure 6 illustrates how our texture fusion method works.

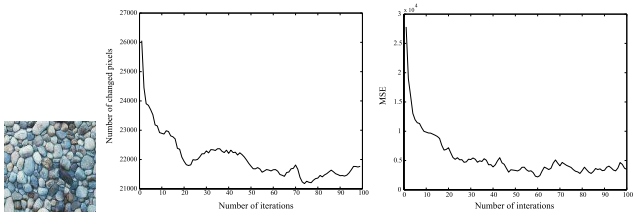
We assume that the fusion map is composed of a set of colored regions. The fusion map is an image which may be hand-drawn, photographed, or computer-generated. There is a one-to-one correspondence between the colors and the texture samples, which are specified by the user. To merge adjacent regions seamlessly, a narrow belt region is laid down along each boundary as shown in Figure 6(b) (green region). The width of a belt is set equal to one half of the width of the larger window between those of the adjacent regions. The size of the window of a belt region is made equal to that of the larger window while the initial grid density is set equal to the dominant scale of the smaller one.

For every pair of regions sharing a common belt, we create a new texture sample by merging their corresponding textures with the graph cut technique. The new texture is used as the texture sample for the belt. In this sense, the texture can be considered as training data for our texture fusion model. With all textures being ready, we first apply our texture synthesis method to each non-belt region with its texture sample and then fill in every belt with its own newly-synthesized texture sample. When filling in a belt, the window of each pixel in the belt is guaranteed to intersect both of the adjacent non-belt regions so that they can make contributions in forming the pixel neighborhood. All regions including the belts are filled in the scan-line order with no iterative refinement.

5 Experimental Results

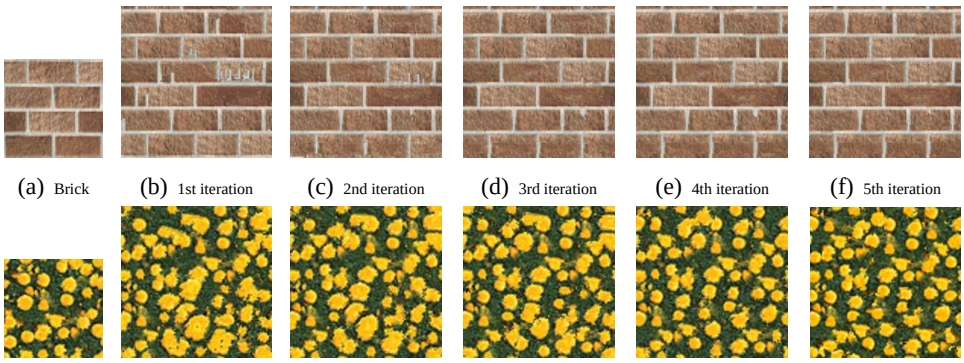
We performed our experiments on an Intel Pentium IV 3.2GHz processor with 4GB of main memory. The texture samples have a resolution of 128×128 or 192×192, and the output textures have a resolution of 200×200. We use Matlab codes for PCA and the ANN library of Mount [13] for kd-tree generation and searching.

We first show the error diffusion behaviors of our iterative refinement scheme with both the mean square errors (MSE) and the number of pixels for the stone texture (Figure 7(a)). As shown in Figures 7(b) and (c), both the MSE and the number of pixels changed decrease rapidly until around 20 iterations and then oscillate within a small range. Empirically, we find that visually-good quality is achieved within 10 iterations for most textures. However, the quality of synthesized textures looks quite good even with a single iteration in many cases.



(a) Stone (b) Number of changed pixels (c) Mean square errors

Figure 7: Convergency of iterative refinement for the rope texture



(a) Brick (b) 1st iteration (c) 2nd iteration (d) 3rd iteration (e) 4th iteration (f) 5th iteration

(g) Marigold (h) 1st iteration (i) 3rd iteration (j) 5th iteration (k) 7th iteration (l) 10th iteration

Figure 8: Examples of iterative texture refinement

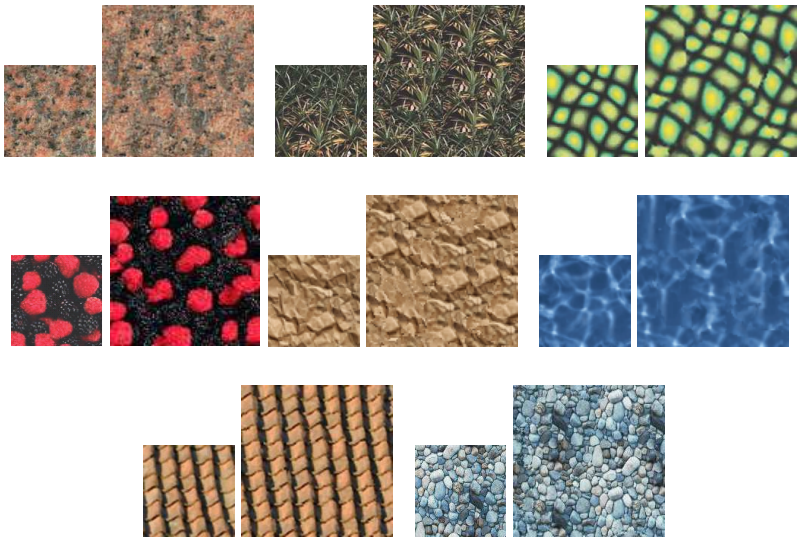


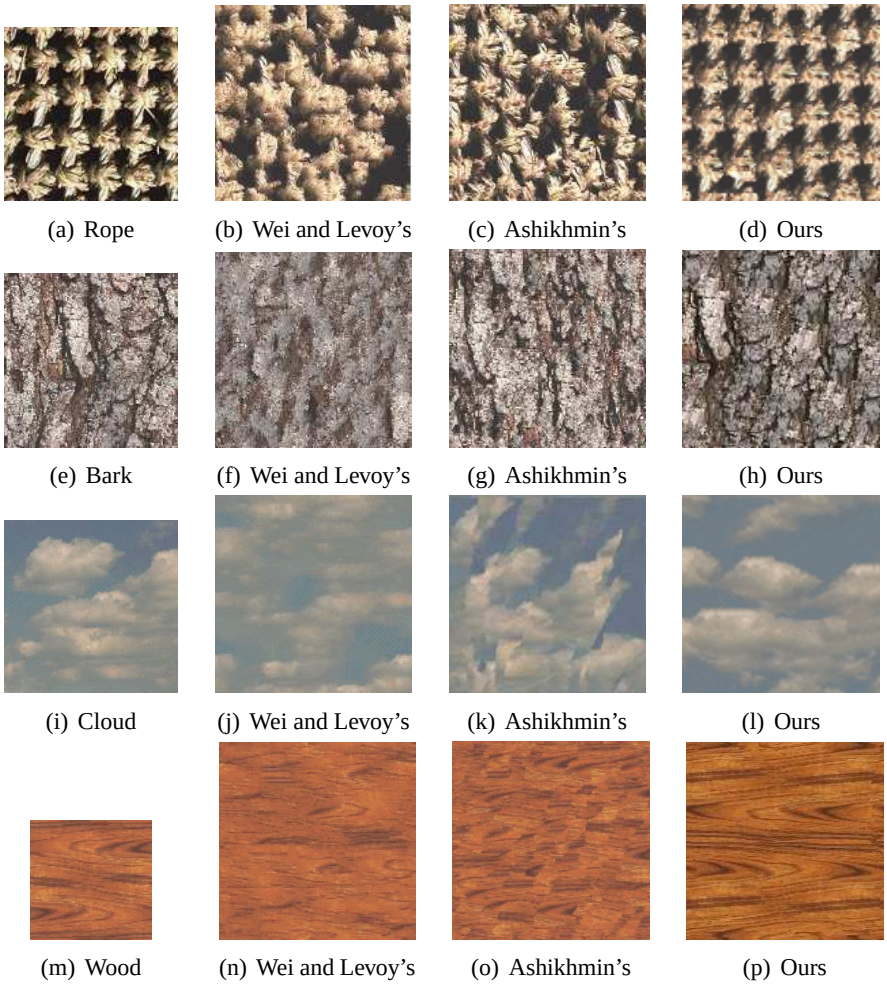
Figure 9: Texture synthesis results. For each pair of images, the left is a texture sample and the right is a result.

Results on various textures are shown in Figures 8-10. In Figure 8, we can observe the evolution of an output texture at each iteration due to iterative refinement. The results in Figure 9 are obtained by applying our method to some free textures available on the Internet. We compare our results with those of Wei and Levoy [18] and Ashikhmin [2] collected from their papers and websites (Figure 10). Timing data for the textures in Figures 8-10 are summarized in Table 1, where statistics for PCA, kd-tree generation, and texture synthesis are given separately in seconds.

The results of texture fusion are provided in the next two figures. In Figure 11, we present our results on texture fusion for two different kinds of beans. The fusion map in Figure 11(c) is automatically generated as follows: A circle is placed at a randomly-chosen pixel with a color, either red (for purple beans) or blue (for yellow beans), based on the following probability density functions:

$$P_r(x) = e^{-x^2/2\sigma^2} \text{ and } P_b(x) = 1 - P_r(x),$$

where x is the distance from the top row of the map, and P_r and P_b denote the probability density functions of red and blue circles, respectively. We set σ to 80, the radius of the red circles to 15, and that of blue ones to 11, where the value of σ is obtained empirically and the radii of circles are taken from the dominant scales of their corresponding textures, respectively. We repeatedly place the circles until all pixels in the fusion map are covered. When a newly-placed circle intersects



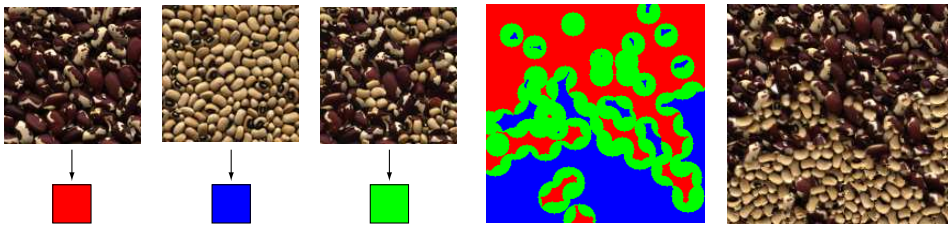
(a) Rope (b) Wei and Levoy's (c) Ashikhmin's (d) Ours

(e) Bark (f) Wei and Levoy's (g) Ashikhmin's (h) Ours

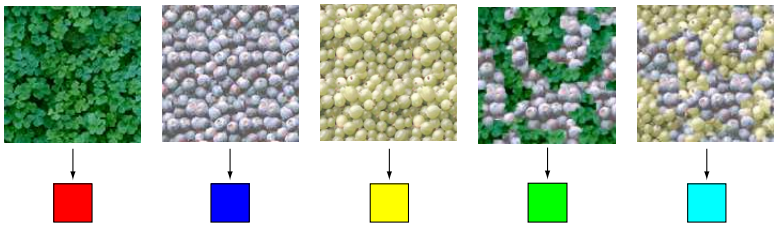
(i) Cloud (j) Wei and Levoy's (k) Ashikhmin's (l) Ours

(m) Wood (n) Wei and Levoy's (o) Ashikhmin's (p) Ours

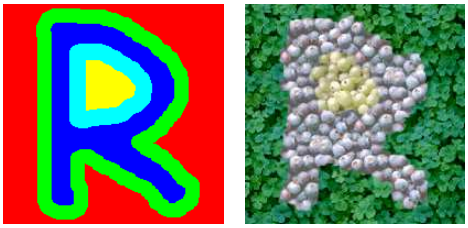
Figure 10: Comparison with previous methods



(a) Purple beans (b) Yellow beans (c) A merged texture (a)+(b) (d) A fusion map (e) An output texture



(f) Clover (g) Grape (h) Plum (i) A merged texture (f)+(g) (j) A merged texture (g)+(h)



(k) A fusion map (l) An output texture

Figure 11: Texture fusion results. Each color box specifies a corresponding region for each texture.

	size	scale	Preprocessing time		Synthesis time	
			PCA	kd-tree generation	total	average
brick	128×128	21	513.4	1.7	287.6	28.76
marigold	128×128	18	359.9	1.3	243.6	24.36
granite	128×128	18	363.0	1.3	265.2	26.52
grass	128×128	19	387.7	1.4	230.9	23.09
frog	128×128	21	511.2	1.6	189.5	18.95
raspberry	128×128	20	480.8	1.5	212.2	21.22
paper	128×128	17	348.7	1.3	217.5	21.75
water	128×128	19	392.8	1.4	259.8	25.98
roof	128×128	21	505.7	1.6	209.2	20.92
stone	128×128	15	287.3	1.2	221.6	22.16
rope	192×192	18	392.2	3.8	258.8	25.88
bark	192×192	14	260.9	2.8	303.9	30.39
cloud	192×192	13	235.2	2.5	239.0	23.9
wood	128×128	23	728.1	1.9	263.2	26.32

Table 1: Timing data for our texture synthesis method

an already-colored region with a different color, the part of the region intersecting the circle is colored green so as to be filled later with the mixed texture. The next example demonstrates a three-way texture fusion provided with a user-drawn fusion map.

As shown in Figure 12, our method can also be used for constrained texture synthesis [18]. Since the pixels on the boundary of the hole are fixed, we fill the hole in a cyclic order (See Figure 12(b)) while moving from the fixed sides toward their corresponding opposite sides, so that every pixel is scanned four times. The visual quality of the resulting texture is slightly better than that of Wei and Levoy [18] which used a spiral order (Figure 12(c)(d)).

The last experiment combines our method with a patch-based method. As explained in Section 2, a patch-based method consists of two stages: patch placement and boundary stitch. We employ our constrained texture synthesis scheme for boundary stitch (See Figure 13). Initially, a texture sample is placed at a randomly-chosen position in the output texture. At each step, the texture is translated to a position where it matches an already-synthesized region (Figure 13(b)). To ensure that the output texture is fully covered, we avoid the previously-chosen positions when placing the texture. The partially-overlapped region is filled by employing our scheme for texture fusion. To make the patch placement efficient, we adopt an FFT-based acceleration technique [11]. Figures 13(c) and (d) show our result and

that of Kwatra et al. [11] for a strawberry texture, respectively.

6 Discussion

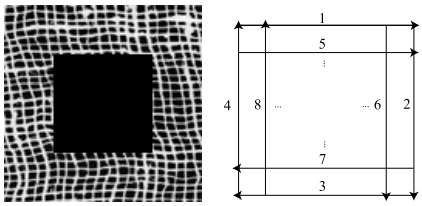
Sampling Density and Window Size: The sampling density plays a key role in collecting the structural information distributed in a texture sample. A rule of thumb to decide it would be to follow the Nyquist limit, that is, less than one half of the dominant scale. However, a dense sampling rate restricts the freedom of pixel sampling, which may cause a verbatim copying artifact. Moreover, due to the randomness in constructing the seed points, the sampling dense rate may also introduce conflicting information to confuse the pixel sampling process, causing a slow convergency in texture refinement. Our compromise was to make the sampling density the same as the dominant scale based on our empirical observations. We instead made the window size twice as large as the dominant scale to hopefully capture the global arrangement of dominant structural components. The efficiency degration due to the larger window size was addressed by employing PCA.

Texture Domain: We have modelled textures as MRFs (Markov Random Fields) adopting the line of other similar approaches. MRFs have been proven to cover a wide variety of useful textures, which are somewhat in-between regular and stochastic. As characterized well in [18], those textures exhibit both locality and stationality. That is, each pixel of a texture is characterized by a small set of spatially neighboring pixels, and this characterization is the same for all pixels. Our method can be applied to such textures (not images) which can be modelled as MRFs.

Temporal Texture Synthesis: We have focused on 2D texture synthesis, in particular, how to enhance the quality of a synthesized 2D texture. However, our method can also be used for temporal texture synthesis, like those in [18, 11]. In this case, the iterative refinement should be avoided for time efficiency.

7 Conclusions

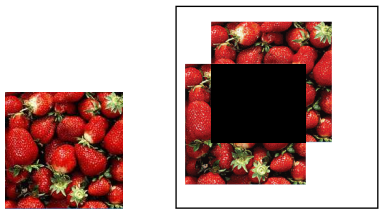
Returning to the question posed in Section 1, we conclude that the problems such as texture blurring and garbage growing are not really inherent in pixel-based schemes, as shown in our experimental results. We claim that the pixel-based iterative refinement strategy yields the best quality with a good initial guess as long



(a) A texture sample (b) Cyclic order

(c) Ours (d) Wei and Levoy's

Figure 12: Constrained texture synthesis



(a) Strawberry (b) Patch placement

(c) Ours (d) Kwatra et al.'s

Figure 13: Combination with patch-based method

as the window size is properly chosen. In this sense, patch-based approaches can be regarded as approximation schemes for acceleration. The most serious disadvantage of our method is efficiency degradation as Ashikhmin [2] pointed out. We believe that our method could be accelerated by incorporating spatial coherency.

References

[1] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

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

[3] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *ICCV (1)*, pages 377–384, 1999.

[4] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.

[5] Jeremy S. DeBonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics*, 31(Annual Conference Series):361–368, 1997.

[6] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 341–346. ACM Press / ACM SIGGRAPH, 2001.

[7] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.

[8] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH*, pages 229–238, 1995.

[9] I. T. Jollife. Principal component analysis. *Springer-Verlag, New York*, 1986.

[10] S.L. Kilthau, M.S. Drew, and T. Moller. Full search content independent block matching based on the fast fourier transform. In *ICIP02*, pages I: 669–672, 2002.

[11] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, July 2003.

[12] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[13] D. M. Mount. Ann programming manual. *Department of Computer Science, University of Maryland, College Park, Maryland*, 1998.

[14] Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000.

[15] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 465–470. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[16] Fumiaki Tomita and Saburo Tsuji. Computer analysis of visual textures. *Hingham, MA: Kluwer Academic*, 1990.

[17] Li-Yi Wei. Texture synthesis from multiple sources. *SIGGRAPH 2003 Sketch and Applications*, July 2003.

[18] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 479–488. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[19] J.S. Weszka, C.R. Dyer, and A. Rosenfeld. A comparative study of texture measures for terrain classification. *IEEE Trans SMC*, 6:269–285, 1976.

[20] Ying-Qing Xu, Baining Guo, and Harry Shum. Chaos mosaic: Fast and memory efficient texture syntheeis. *Microsoft Research Technical Report MSR-TR-2000-32*, April 2000.

[21] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph.*, 22(3):295–302, 2003.