A Hybrid Based Texture Synthesis Approach

Workshop on Multimedia Technology

Chang-Hsing Wu, Yueh-Yi Lai, and Wen-Kai Tai*
Department of Computer Science and Information Engineering
National Dong Hwa University

wkdai@mail.ndhu.edu.tw

+886-3-8662500 ext.22118
Fax: +886-3-8662781
1, Sec. 2,Da Hsueh Rd., Shou-Feng, Hualien,
974 Taiwan, Republic of China

Abstract

We present a texture synthesis approach that works well for a wide variety of textures without any knowledge of their physical information processes, especially well-suited for natural textures. Our approach starts from extending a sample texture and then pastes patches into the output texture, forming an inverted U-shaped for initialization. Finally, we generate synthetic texture of arbitrary size. Also, we accelerate the synthesizing process using the Approximate Nearest Neighbor technique. As the experimental results shown, so the proposed approach is effective and efficient.

Keywords: Computer Graphics, texturing, texture synthesis

1. Introduction

Texture is a ubiquitous visual experience. It describes a wide variety of surface characteristics such as terrain, plants, minerals, fur, skin, and so forth. Since reproducing detailed surface appearance is important to achieve visual realism in rendered-images, textures are commonly employed when we render synthetic images. A texture can be used to modulate various surface properties, including color, reflection, transparency, or displacements. Textures can be obtained from a variety of sources such as hand-draw pictures, scanned photographs, and procedural texture synthesis. Texture synthesis is an alternative way to create textures. Because synthetic textures can be made any size, visual repetition is avoided. Texture synthesis can also produce tileable images by the way we properly handling the boundary conditions.

A texture synthesis approach that we proposed can effectively synthesize a wide variety of textures. First, our approach extends the input sample texture, then adopt Liang's algorithm [6] to past patches into the output texture, forming an inverted U-Shape for initialization. Finally, the output texture is filled using L-Shape method of Ashikhmin's algorithm [2].

The major contributions of our approach are summarized as follows: 1) We combine the advantages of Ashikhmin's algorithm [2] and Liang's algorithm [6] to generate textures with better perceived quality. 2) Our approach eliminates the short horizontal edges appeared in the output texture of Ashikhmin's algorithm [2] by extending the input texture. 3) Pasting an inverted U-shape into the output texture as elementary synthesizing basis, our approach is even more suitable for synthesize Photographic pseudo-periodic textures, because of reserving the global information of the input texture.

The rest of this paper is organized as follows. In Section 2, we review previous related works. In Section3, we present our approach and acceleration technique. We show and discuss the results of our works in Section 4. Finally, in Section 5, we conclude and give some possible extensions of this work.

2. Previous Works

Numerous approaches have been proposed for texture analysis and synthesis. In this section, we briefly review some recent and representative works. There are two mainly strategies used by the current texture synthesis algorithm. The first strategy is to compute global statistics in feature space and thereby sample images from the texture ensemble directly on a large image lattice. The second strategy is to compute local conditional Gibbs distributions and thus synthesize pixels incrementally.

Algorithms using first sampling strategy include De Bonet [3] and Heeger [4]. When using this strategy one samples texture from an ensemble, and usually one may sacrifice the effectiveness for speedy synthesis or for ease of sampling. Algorithms using the second sampling strategy include Wei [9], Ashikhmin [2], Xu [10], and Liang [6]. These methods used L_2 distance to determine the similarity of two neighborhoods $N(p_1)$ and $N(p_2)$. The L_2 distance is a sum over all pixels in the neighborhood of squared differences of pixel values at a particular position.

Wei and Levoy [9] proposed an algorithm, which starts with an input texture sample I_a and a white random noise I_s. Pixels in the output image are assigned in a raster scan order. The value of each output pixel p is determined by comparing its spatial neighborhood $N(p)$

with all neighborhoods in the input texture. The pixel with the most similar neighborhood will be assigned to the corresponding output pixel. Unfortunately, this approach inevitably produces seams between individual texture patches and the technique either uses a fallback algorithm to fill in transition regions or simply blurs the seams.

Ashikhmin [2] proposed a simple texture synthesis algorithm that is well suited for a specific class of naturally occurring textures. The key observation is that at a given step during the [9] synthesis process, pixels in the input sample with neighborhoods similar to shifted current neighborhood in the output image. This algorithms assume that pixels from the input sample that are appropriately “forward-shifted” with respect to pixels already used in synthesis are well-suited to fill in the current pixel. This technique is simple, easy to implement, and efficient and produces good visual results, but is not well-suited for temporal texture synthesis in most cases.

Liang [6] proposed a patch-based method. 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}. The patch-based sampling algorithm is fast and it makes high-quality texture synthesis a real-time process. And it works well for a wide variety of textures ranging from regular to stochastic. But this method could not synthesize the texture have the interrupted object boundaries or any other high-frequency features.

3. Hybrid Based Texture Synthesis Approach

Our hybrid based texture synthesis approach combines the algorithm of Ashikhmin [2] and Liang [6]. First, we extend the input image to eliminate the problem of short horizontal edges. Second, we paste patches into the output texture for initialization, forming an inverted U-Shape. Third, the pixels in the output texture are then filled according to Ashikhmin's algorithm [2]. Moreover we use the ANN technique to accelerate the search of nearest boundary zones.

3.1 EXTENDING THE INPUT IMAGE

Ashikhmin's algorithm [2] makes some short edges obviously appear in the output texture I_o. Because it tends to grow patches starting from some position in the input sample texture I_i and continuing to drop down to the bottom of the input sample texture, as shown in **Figure 1**, and patches them reconnect to the top of the input sample texture, as shown in **Figure 2(a)**. For that, we propose a method to eliminate this obviously boundary edge by connecting the bottom and left ends of the input sample texture to someplace in itself seamlessly, as shown in **Figure 2(b)**. Following steps are performed before we synthesize an output image.

(a) As shown in **Figure 3(a)**, we use the boundary zones [6] of the input sample texture to find the best match patch $B_i(x,y)$ from the input sample texture.

(b) Paste the best match patch $B_i(x,y)$ in such a way that overlays the boundary zones of the input sample texture, as shown in **Figure 3(b).**

(c) Update the array of original position in the input sample texture in accordance with coordinates of pixels of the patch $B_i(x,y)$.

(d) Repeat steps (a), (b) and (c) until the input sample texture is extending completely, as shown in **Figure 3(c)**.

The patch-size of extended sample texture, w_S, must be larger than the size of L-Shape, w_L, because L-Shape maybe slide into the bottom of the input sample texture I_i at the synthesis phase. The patch-size of extended sample texture, w_S, affects how well the synthesized texture keeps the local characteristics of the input sample texture I_i without fracture. For an input texture of size $w_i \times h_i$, the patch size, w_S, should be $w_S = \lambda \min(w_i, h_i)$, where $0 < \lambda < 1$. We always set w_S is the scale of the texture feature. Unless stated otherwise, all examples are generated with λ values between 0.25 and 0.5.

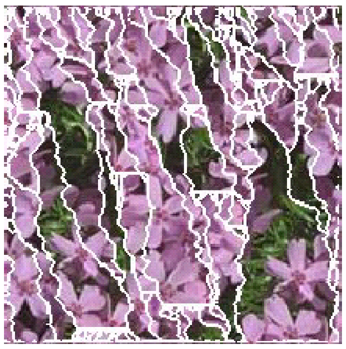
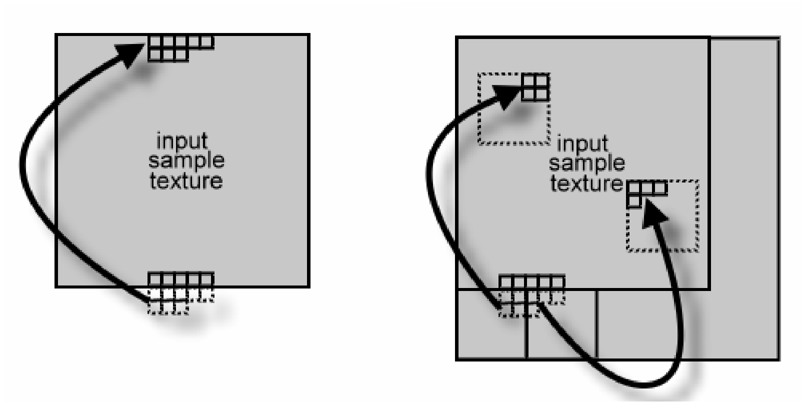
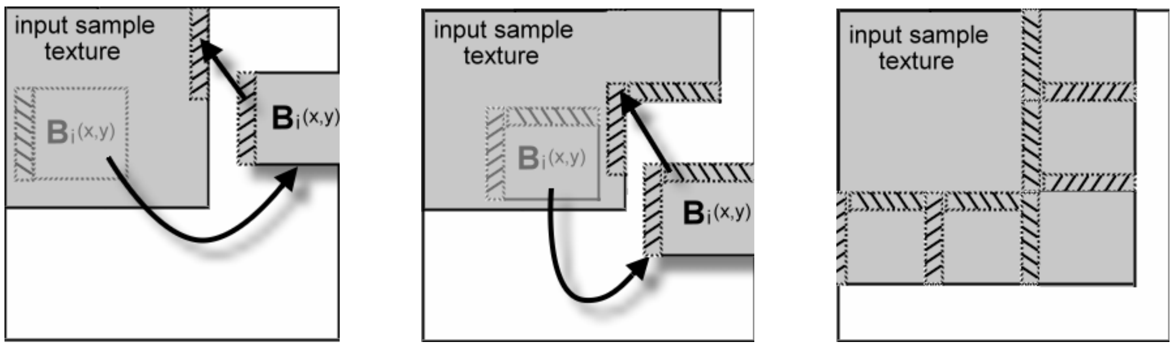


Figure 1. *Region-growing: boundaries of texture pieces are marked white.*



(a) (b)

Figure 2. *(a) The L-Shape wraps to the top of the input sample texture. (b) The L-Shape reconnects to some where in the input sample texture.*



(a) **(b)** **(c)**

Figure 3. *(a) Find the best patch which search from the input sample texture to connect the input image. (b) Continue finding the best match patch to paste. (c)The extended input sample texture.*

3.2 SYNTHESIZING PROCESS

Generally, as Wei [9] and Ashikhmin [2], the first step is to initialize the output image to random noise with a histogram which is equal to the input image. However, random noise leads output image to lose more global information. Because the patches can provide more correct global information for growing pieces, we instead use [6] to paste patches into the output texture I_o. The initialization steps of our algorithm proceeds as follows.

a) Randomly choose a $w_B \times w_B$ texture patch $B_i(x,y)$ from the input sample texture I_i, and paste $B_i(x,y)$ at the top left corner position *(0,0)* of the output texture I_o. Patch $B_i(x,y)$ becomes the patch $B_o(0,0)$ in the output texture.

b) As shown in **Figure 4(a)**, $B_o(0,0)$ has a boundary zone $E\{B_o(0,0)\}$. We use $E\{B_o(0,0)\}$ to find the best match in the input sample texture I_i.

c) The best match $B_i(x,y)$ has a boundary zone $E\{B_i(x,y)\}$ that has the minimum distance from $E\{B_o(0,0)\}$. We paste patch $B_i(x,y)$ into the output texture I_o.

d) Repeat steps (b) and (c) until the top row of the output texture is fully covered, as shown in **Figure 4(b)**.

e) As shown in **Figure 4(c)** and **(d)**, we paste the left and right columns using the same process like steps (b) and (c).

The patch-size, w_B, must be larger than the size of L-Shape, w_L to ensure that the L-Shape is fully inside the inverted U-Shape. The width of boundary zone should be sufficiently large to avoid mismatching features across patch boundaries. The width of the boundary zone is typically four pixels wide in our experiment.

When the initialization process is done, we have an instance of output textures as shown in **Figure 5**. We also update the array of original position in accordance with the output texture I_o. After pasting an inverted U-Shape in the output texture I_o for initialization, we will fill the output texture according to Ashikhmin [2] which slides an L-Shape neighborhood of a specific (fixed) size pixel by pixel in the raster scan order. The synthesizing process proceeds as follows.

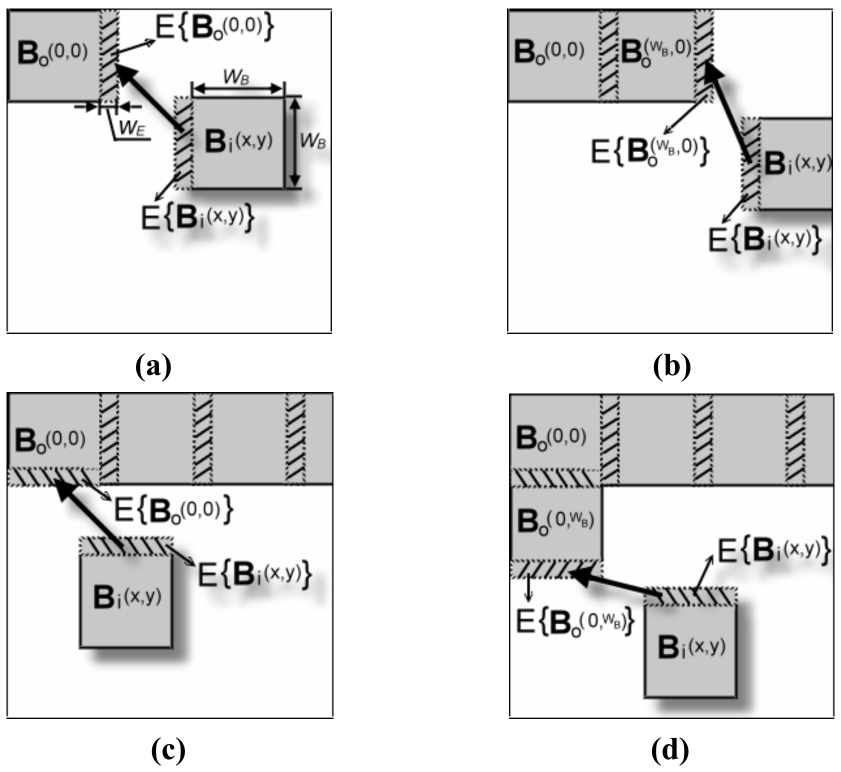


Figure 4. *synthesizing an inverted U-Shape. The grey area is already synthesized. (a)Randomly choose a patch, and pastes it into I_o at the top left corner. Then, find a patch $B_i(x,y)$such that the boundary zones $E\{B_i(x,y)\}$ is the best match $E\{B_o(0,0)\}$,and the best match patch $B_o(x,y)$ is pasted and overlapped boundary zone $E\{B_o(0,0)\}$ of the patch $B_o(0,0)$. (b)The best patch $B_i(x,y)$ is chosen from the input sample texture. The patch $B_i(x,y)$ pastes into the output texture until the top row of the output texture is fully pasted. (c)(d) The best patch $B_i(x,y)$ is chosen from the input sample texture, and pastes into the output texture for left most column.*

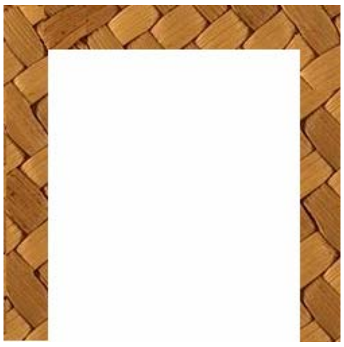


Figure 5. *An instance of inverted U-Shape in the output image.*

(a) For the pixel that centers on the L-Shape to be filled, we use each pixel of the L-Shaped neighborhood in the output texture I_o to find the candidate L-Shapes in the input sample texture I_i using the array of original position. Location of the candidate pixel is appropriately shifted. All candidate L-Shapes generate the candidate pixels as shown in **Figure 6**.

(b) Remove duplicate candidates if candidate pixels are the same pixel in the input sample texture.

(c) Compute the L_2 distance for every candidate L-Shape with respect to the current L-Shape in the output texture I_o, and choose the best candidate L-Shape that has the minimum distance.

(d) Copy the best candidate pixel that centers on the best candidate L-Shape into the current pixel in the output texture I_o.

(e) Finally, save original coordinate of the best candidate pixel in input sample texture to the array of original position.

(f) Repeat this process, (a) - (f), until all pixels of the output texture are fully filled.

The size of L-Shape, w_L, affects how well the synthesized texture. It is set according to the feature of the input sample texture. In this article, all the experimental results are generated with $w_B = 11 \times 11$.

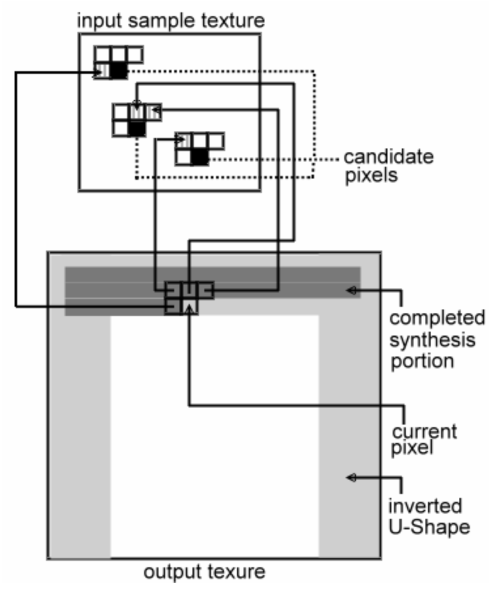


Figure 6. *Creating candidate pixels. Each pixel of the current L-Shaped neighborhood generates candidate L-Shapes of the input sample texture I_i according its original position (hatched). Most pixels of the current L-Shaped neighborhood always point to the same candidate L-Shaped of the input sample texture I_i. We create candidate list and compute distance between current L-Shape and candidate L-Shape. Choosing a candidate pixel which has the minimum distance between two L-Shapes, and copy it into the current pixel.*

3.3 ACCELERATION

Searching for nearest boundary zones is the major computation in our synthesis algorithm. We treat the boundary zone of a patch as high-dimensional points, and the boundary zone matching process is considered as a nearest-point searching problem. So we can accelerate the search by using the Approximate Nearest Neighbor (ANN [7]) technique. We build a ***kd***-tree for each one of the two boundary zone configurations in the input texture. In order to find approximate nearest neighbors, we search the ***kd***-tree, and obtain some candidates. We choose the best match of high-dimensional point from the candidates, and paste it into the output texture.

3.4 DISCUSSIONS

We use Liang [6] to find the starting growing point of every piece without shift. Our algorithm created output texture by merging irregular pieces, and it merges every one of irregular pieces seamlessly. Therefore, our algorithm is especially well-suited for naturally occurring textures and can be applied to synthesize a wide variety of textures effectively. We will classify sample textures and demonstrate results by our algorithm in next section. However, because Liang's [6] is unable to find the best patches while extending the input sample texture, our method in few cases may appear seams at obvious feature in the output texture.

4. Experimental Results

We have tested our approach using many kinds of images from standard texture sets (web site http://www.cns.nyu.edu/~eero/texture/). All the experimental results are computed on a 667MHz Pentium III with 256 MB SDRAM. To compare quality with current representative texture synthesis methods, we have implemented Wei and Levoy's method [9] , Ashikhmin's method [2] and Liang's Method [6]. Two synthesis results are shown in Figure 7. These figures show the input texture (first row), the synthesized results of our hybrid method (second row), Liang's method (third row), Ashikhmin's method (fourth row), and Wei and Levoy's method (fifth row). In these figures, input textures are 192×192, and all results are 250×250.

As shown in Figure 7, 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. Wei and Levoy's method have a tendency to blur out the finer details. As shown in the left column of Figure 7, Liang's algorithm has no explicit model for these objects. And these objects in the synthesis texture always do not resemble those in sample texture. Some boundary edges of the patch appear in this result. In the right column, Liang's algorithm appears some blades of plants.

From these figures, we notice that our algorithm achieves good results, at least as good as those from the Liang's method, and these result are better than those from Ashikhmin's method and Wei and Levoy's method.

Table 1 and Table 2 summarize the performance of our method. Table 1 lists timings of our algorithm in seconds. The pre-process time includes the time of building ***kd***-tree and the time of extending the input sample texture. This table also provides the timing for synthesizing textures of various sizes from an 128×128 input texture sample. For each input texture, we test 5 times and take the average of them.

Table 2 compares the speed of our algorithm with Liang's algorithm [6] which is one of the fast general-purpose texture synthesis algorithms. It shows the timing for synthesizing 200×200 textures from 128×128 samples in seconds. The spending time of our method is slower than Liang's.

		(a)	(b)	(c)	(d)
Input Texture					
Pre-process Time	KD-Tree	3.514085	1.654426	1.713418	1.808752
	Texture Extended	3.306193	3.331661	3.320084	3.305694
Synthesis Time (200×200)		1.244877	1.184464	1.145471	1.522517
Synthesis Time (250×250)		1.854303	1.847851	1.756317	2.252837
Synthesis Time (400×400)		4.648415	4.434351	4.280608	5.508948

Table 1. The pre-process time and synthesis time of our algorithm in seconds

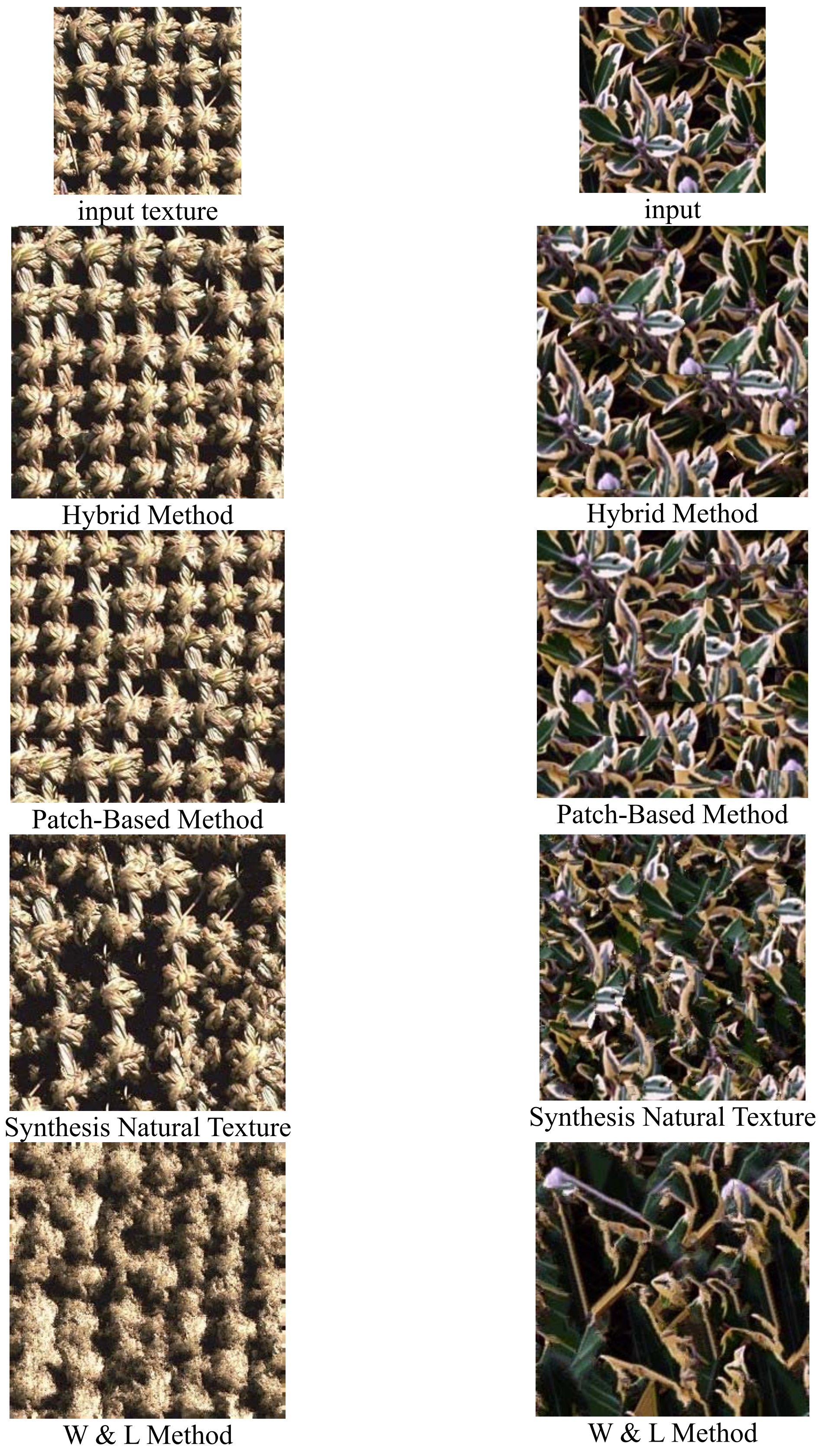


Figure 7. The result of texture synthesis of different synthesis algorithms

Method		Analysis Time	Synthesis Time
Our Method		5.389	1.303
Patch-Based Algorithm	Exhaustive*	0.0	1.415
	QTP**	0.017	0.256
	QTP + KD-Tree**	0.338	0.044
	QTP + KD-Tree + PCA**	0.678	0.020

Table 2. Timing comparison: Our method and Patch-Based Sampling
*Exhaustive means no acceleration in used.
**QTP, KD-Tree, and PCA are the methods of acceleration [6]

Furthermore, we synthesize many textures to prove that our method worked very well. **Figure 8** to **Figure 12** are results of our experiments. These figures show that our method performs on various textures very well. All the synthesized results are of 250×250 textures corresponding 128×128 samples with parameter λ=0.25.

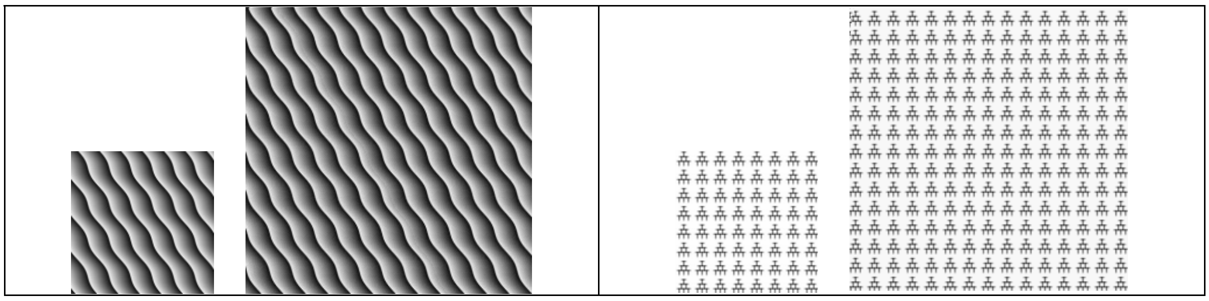


Figure 8: *Experimental results for type Ⅰ textures, artificial periodic texture*

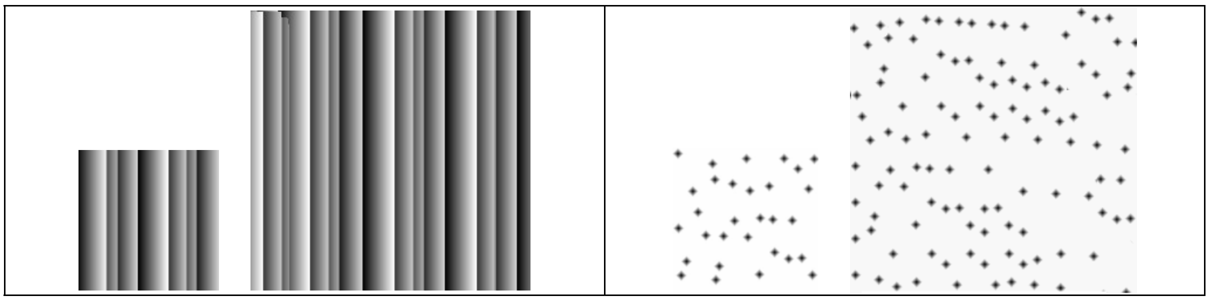


Figure 9. *Experimental results for type Ⅱ textures, artificial non-periodic textures*

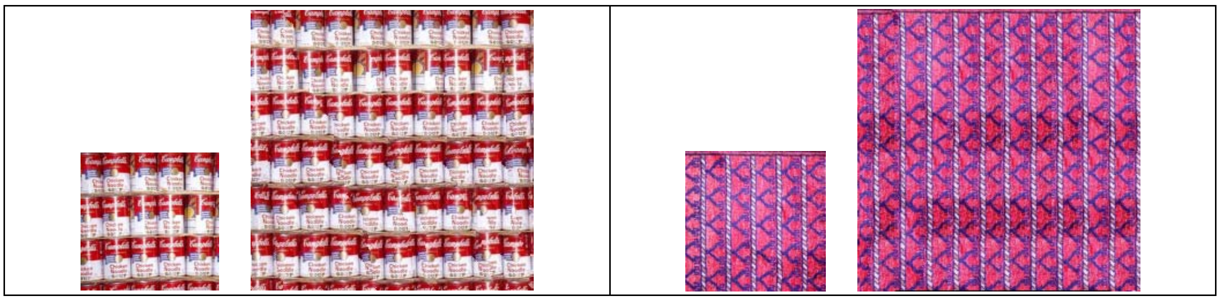


Figure 10. *Experimental results for type Ⅲ textures, photographic pseudo-periodic textures*

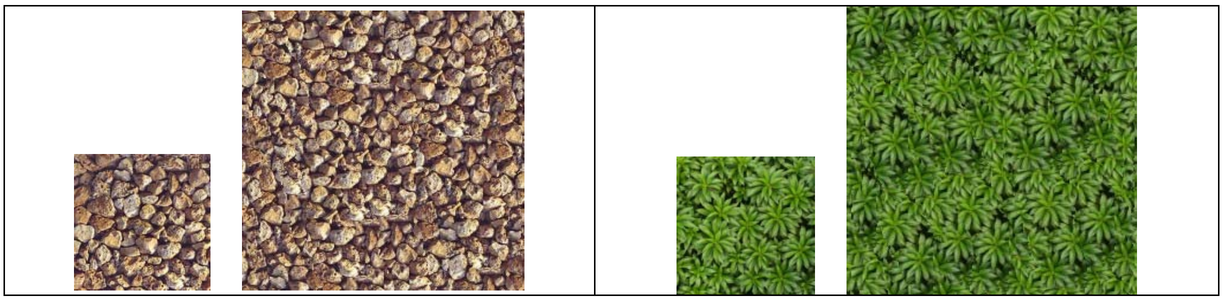


Figure 11. *Experimental results for type Ⅳ textures, photographic random textures*

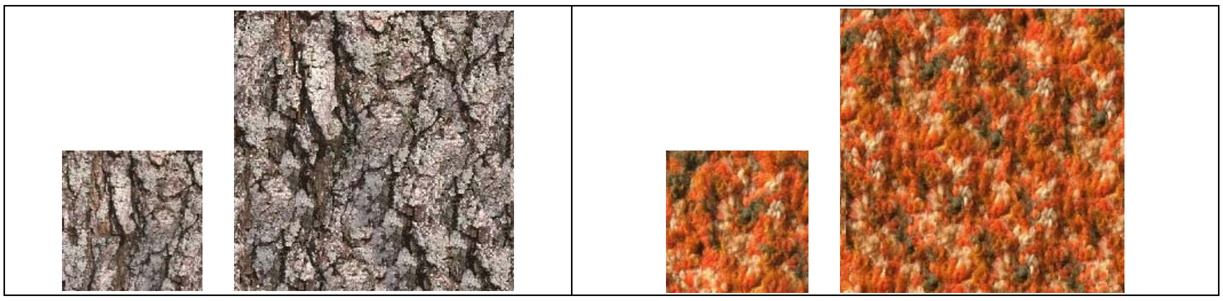


Figure 12. *Experimental results for type V textures, photographic structured textures*

5. Conclusions and Future Works

Textures are important for a wide variety of applications in computer graphics and image processing. Texture synthesis has been an active research area for many years. We propose an approach that can effectively synthesize a wide variety of textures and especially well-suited for naturally occurring textures. Our hybrid method extends the input sample texture in the first step, and then we adopt Liang's algorithm [6] to paste patches formed an inverted U-Shape for initialization. Finally, our approach fills the output texture using L-shape method of Ashikhmin's algorithm [2] to synthesize texture.

In few cases our approach may appear seams at obvious feature in the output texture. Since [6] is unable to find the best patches while extending the input sample texture without seam, it is necessary that we should further propose an optimal method that can produce the best patches. In addition, we are interested in extending the ideas presented here for texture synthesis on surfaces of 3D objects, texture mixtures, and temporal texture synthesis.

References

[1] V. I. Arnold and A. Avez. *Ergodic Problems of Classical Mechanics.* Benjamin, 1968.

[2] Michael Ashikhmin. Synthesizing Natural Textures. *ACM Symposium on Interactive 3D Graphics*, March 2001.

[3] J. S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Image. *Computer Graphics Proceedings, Annual Conference Series*, pages 361-368, August 1997.

[4] D. J. Heeger and J. R. Bergen. Pyramid-Based Texture Analysis/Synthesis. *Computer Graphics Proceedings, Annual Conference Series*, pages 229-238, July 1995.

[5] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. *ACM SIGGRAPH 2001*, 12-17, August 2001.

[6] L. Liang, C. Liu, Y. Q. Xu, B. Guo, and H. Y. Shum. Real-Time Texture Synthesis by Patch-Based Sampling. *Microsoft Research Technical Report MSR-TR-2001-40*, April 2001.

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

[8] R. Szeliski and H. Y. Shum. Creating Full View Panoramic Mosaics and Environment Maps. *Proceedings of SIGGRAPH*, pages 251-258, August 1997.

[9] L. Y. Wei and M. Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Computer Graphics Proceedings, Annual Conference Series*, pages 479-488, July 2000.

[10] Y. Q. Xu, B. Guo, and H.Y. Shum. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis. *Microsoft Research Technical Report MSR-TR-2000-32*, April 2000.