병렬 회전 예제 기반 텍스처 합성

박한욱° 김창헌

고려대학교

{frdlanve, chkim}@korea.ac.kr

Parallel Rotated Exemplar-based Texture Synthesis

Han-Wook Park° Chang-Hun Kim

Korea University

요 약

본 논문에서는 다수의 회전 이미지를 생성, 이용해 결과의 다양성을 추구하고 기존 기법들의 문제인 예제 가장자리 경계면의 Neighborhood를 이용해 생기는 경계선을 완화한 합성 결과물을 생성하는 새 예제 기반 텍스처 합성 방법을 제안한다. 논문에서 제안하는 방법은 구현하기에 따라 공간 결정적인 형태로 구축할 수 있으며, 병렬 처리가 가능한 하드웨어를 이용한 병렬 연산처리로 합성 속도 가속을 하는데도 유리한 구조를 가지고 있다.

Abstract

We present a simple new idea to improve the quality of exemplar based texture synthesis using multiple rotated input exemplars. Our algorithm successfully obtain rotational synthesis feature variations and manages to reduce the artifacts in the results, especially patch seams due to the structure of the exemplars provided which have been inappropriate for previous neighborhood matching synthesis algorithms. Our algorithm is parallel in nature, thus it is possible to implement our algorithm using GPU or multi-core CPU to accelerate synthesis process.

키워드: 텍스처 합성, 회전 예제 이미지, 합성 다양성
Keywords: texture synthesis, rotated exemplar, synthesis variation

1 Introduction

Exemplar-based texture synthesis produces a texture of arbitrary size which is visually similar to the given input exemplar. Texture synthesis can be very useful to provide more detailed surface textures for large objects or terrain, with limited size of texture supplies, or to introduce user control to the given input texture image to provide an interface to design a texture. Generally a texture synthesis algorithm is preferable if the algorithm can produce a result which is aperiodic, infinite, and visually similar to the provided input with no or little texture artifacts. Recent texture synthesis algorithms managed to provide a result of good quality which is infinite, spatially deterministic, aperiodic which is fast enough by using parallel processors such as GPU.

However, despite of these technological improvements there still are some unresolved problems in texture synthesis. One of such problem is repetition in the synthesis product. There were various attempts to eliminate periodical repetitions, like forcing spatial distribution of designated texture feature, or applying multiple perturbations to the result to introduce fine scale variations. Still, even if these methods prevent periodical synthesis result, the synthesis product will show nearly same but slightly different texture features all over the image. Since there are no way to alter an image feature completely this is quite unavoidable, thus most texture synthesis algorithms suggest various methods to alter feature structures to provide more variation. Also, there are some textures which are not very appropriate to be used for an exemplar for conventional texture synthesis algorithms. These textures generally are not toroidal, and may have large structural or color intensity difference along the border transitions when tiled. Since it is hard to find appropriate transitional patch to expand border areas of these textures, there can be texture seams in the synthesized result as shown in Fig. 1. We propose a new simple idea to reduce these problems by generating multiple rotated exemplar images to improve feature variety and reduce patch seam artifacts.

2 Related work

The goal of exemplar-based texture synthesis is to use given input exemplar as much as possible to produce a visually similar image. A kind of approach to achieve the goal is iteratively optimizing number of overlapping patches. In patch optimization approach, an algorithm will repeatedly cut-and-paste a patch of random size to reduce overall patch overlapping errors using dy-

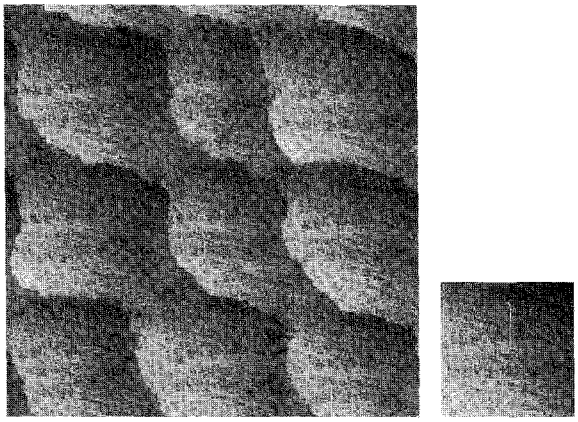


Fig. 1: A synthesis result using the algorithm of Lefebvre et al 2005[5] with a texture exemplar which has great difference along the edges of the image.

namic programming[2] or Graphcut[4]. Although it is good to say the algorithms using patch optimization approach will result above average quality since it pastes a collection of pixels instead of pixel by pixel, patch optimization algorithms are hard to avoid repetitive texture features because it is hard to introduce fine scale feature variations to the image being synthesized. One other approach is neighborhood matching, which fetches a neighborhood around a pixel and finds the best matching exemplar neighborhood to rewrite the pixel in sequential order, or individually. This approach is especially good to introduce fine scale variations by distorting the resulting image a bit during synthesis process, but have some possibility that the result may has some broken structures. It is also good to enforce some constraints over the result, like giving a designed color control image[1]. Hertzmann et al[3] used a pair of images to achieve some effects like super-resolution, texture transfer, artistic filtering. Zhang et al[10] used binary texton masks to deform and make a transition between two textures. Also, by rewriting each pixel individually, a number of algorithms[9, 5, 6] could be implemented with parallel processors such as GPU, to drastically reduce synthesis time. Recently, there was a new application of exemplar-based texture synthesis, which synthesizes a smaller exemplar which best represents the given input texture, which considers orientations of anisotropic textures[7].

3 Synthesis overview

Our goal is to produce an image of arbitrary size which is visually similar to the given input exemplar while introducing some rotational variations. To do this, we extend the algorithm introduced in 'Parallel Controllable Texture Synthesis'[5] to provide some support for rotated input exemplars. For each synthesis pyramid l, we synthesis a coordinate image $\mathbf{S}_l$ which refers to the corresponding exemplar pyramid $\mathbf{E}_l$ by performing upsample, jitter, correction.

Upsample Upsample is a step which prepares a coordinate image $\mathbf{S}_l$ based upon the image $\mathbf{S}_{l-1}$ which was generated by previous step, doubling its coordinates.

Jitter Jitter is to prevent periodic synthesis result and give some fine scale variations by perturbing the coordinate image $\mathbf{S}_l$.

Correction Correction step corrects the overall structure of the image being synthesized by rewriting each pixel to new coordinate which best matches to the pixel being matched. We use 5^2 sized neighborhood to match neighborhoods.

4 Preprocessing

4.1 Preparing rotated exemplar images

We precompute all needed information beforehand to eliminate unnecessary computational cost during actual synthesis step. To introduce rotational variations to the synthesis result, we have to prepare a number of rotated images for each pyramid level **l**. The number of exemplars to be generated is defined as $imax$, which is typically 8 for our implementation. We put an index sequentially to each rotated exemplar and denote them as $\mathbf{E}_l^i$.

$$E_l^i = rotation(E_l, \theta_i), \quad \theta_i = \frac{i}{imax} 360 \tag{1}$$

While performing rotation for each exemplar, there are areas which cannot be sampled from original input exemplar since the location is out of the bound of the input. To fill the empty areas, we sample the image as tiled texture for toroidal exemplar, or mirrored texture for non-toroidal exemplar.

After we produce a collection of rotated exemplars, we flag each of the rotated images as 'toroidal' or 'non-toroidal'. Even if the given exemplar image was non-toroidal, the rotated image cannot be guaranteed as toroidal after being rotated, thus we mark every image which was not rotated with degrees of 90, 180, 270 as non-toroidal. The images rotated with degrees of 90, 180, 270 are toroidal, if the input image was. We perform this operation for every pyramid level to prepare rotated image stacks from level 0 to final level **L**.

4.2 Redefinition of coordinate image

Since we prepared number of rotated exemplar images for each pyramid level, $\mathbf{S}_l[p]$ have to refer to the pixel value of same level exemplar $\mathbf{E}_l^i$, but conventional method to carry only x and y value of the coordinate is not enough to redirect the coordinate to specified exemplar. Therefore, we add additional image index **r** to 3rd channel of the coordinate image from the original 2 channel coordinate image. Sometimes we have to refer to only x and y component, or r component of the image. In this case, we denote each as $\mathbf{S}_l^s[p]$ and $\mathbf{S}_l^r[p]$, thus getting a color value of location p will become $\mathbf{E}_l^{S^r[p]}[S^s[p]]$. We show an example in Fig. 2. The upper left image is input exemplar, and the image next to it is visualization of the corresponding coordinate image in RGB channel. The third image shows only r component of the coordinate image and the last image is synthesis result image using coordinate image and input exemplar.

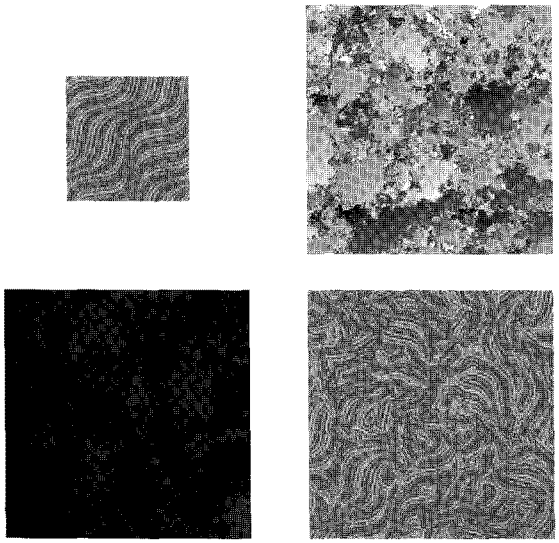


Fig. 2: Visualization of coordinate images and its colorized result. Two coordinate images are modified to fit into the range of 0 to 255 to be easily distinguished.

4.3 Candidate preparation

As introduced in Tong et al[8] and Lefebvre et al 2005[5], we prepare a finite number of synthesis candidates for each pixel of rotated exemplar pyramid. This is very crucial step for our algorithm as by preparing a candidate which has an index of rotated image other than the image being prepared, our synthesizer can use neighbors of other rotated image in same pyramid level during correction step to obtain smooth transition between rotated features and improve synthesis quality at the same time. For each rotated exemplar image $\mathbf{E}_t^s$, The kth component of candidate for location $\mathbf{u}$ on $\mathbf{E}_t^s$is denoted as $\mathbf{C}_k^{t,s}(u)$. We search for total 3 candidates with minimal neighborhood matching error for each pixel in each rotated exemplar pyramid by comparing its neighborhoods, using sum of squared distance of neighbor vectors. Typically, the first candidate searched will be in the same image, but we restrict the search to not find a candidate on the same image being searched from the second candidate and following ones to give more rotational variation to the image being synthesized.

4.4 PCA Projection

To reduce dimension of the neighborhood to be matched, we employ principal component analysis as like Lefebvre et al 2005[5] did. This can greatly reduce computational cost during correction step. As shown in Lefebvre et al 2005[5], we first project 3 channel RGB exemplar images to 2 channel, then cache its neighborhoods to another images, projecting them to 6 channel from 50 dimension. However, this 6 channel image must preserve its transformation matrix to match neighborhoods during correction step, so every rotated neighborhood images must have same projection matrix. Therefore, we create a long strip from rotated images when performing PCA projection to create a single projection matrix for each level then split them into each rotated images afterwards.

5 Synthesis iteration

5.1 Initialization

Before getting into actual synthesis process, we have to initialize the initial coordinate image $\mathbf{S}_0$. We assign 0 for each of all coordinates for $\mathbf{S}_0^c$. However, to present rotated features we have to assign different image index for each pixels for $\mathbf{S}_0^r$, thus we assign a random index value to each pixels, or if a user wants, we may assign a specified value for specified pixel or initialize all pixels with same specified value. If we assign all 0 to the image index values, the result will be very similar to that of conventional neighborhood matching texture synthesis algorithm as shown in Fig. 3. Still, even if we do so, the result can be very different if the given exemplar has great difference along the edges when tiled. (See Sec. 6)

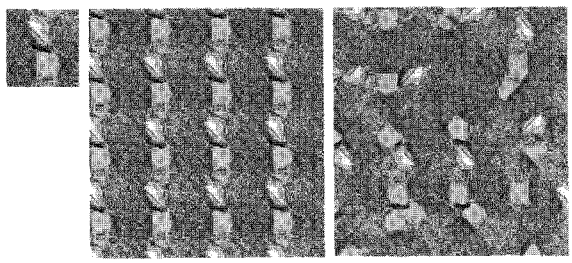


Fig. 3: The leftmost image is input exemplar. The image next to it is a result initialized with random rotated image index values, while the rightmost image is initialized with all 0.

5.2 Synthesis process

As mentioned in Sec. 3, we perform upsample, jitter, and correction for each level iteratively. In upsample step, we simply upsample the coordinates from the coordinate image of previous step. We can do this by assigning four children coordinates of parent coordinate from previous image to each pixel of the image of current level.

$$\begin{aligned} S_l^c[2p+\Delta] &:= (2S_{l-1}^c[p]+\Delta)\ mod\ m \\ S_l^r[2p+\Delta] &:= S_{l-1}^r[p] \\ \Delta &\in \left\{\binom{0}{0},\binom{0}{1},\binom{1}{0},\binom{1}{1}\right\} \end{aligned} \tag{2}$$

After performing upsample step, we perform jitter to give some variations to the coordinate image by perturbing its coordinates. We define a jitter function which produces a random value between -1 to 1. Since we do not consider spatial determinism in our algorithm, it is generally enough to have a random value generator for jitter function $\mathbf{J}()$. We also give a user specified jitter constant $\mathbf{b}_l$ for each level to provide some jitter control interface. Upsample and jitter step does not alter image index r of coordinate image.

$$S_i^c[p] := (S_i^c[p] + J_i(p)b_i), \quad 0 \le b_i \le 1 \tag{3}$$

Finally, we perform correction step to correct the image being synthesized to have similar structures that of the rotated exemplar images. For each pixel p in coordinate image $\mathbf{S}_i$, we fetch a neighborhood $\mathbf{N}S_i(p)$, and try to find most well matching exemplar location u by measuring error between $\mathbf{N}S_i(p)$ and exemplar neighborhood $\mathbf{N}E_i^i[u]$. The overall process is very similar to that of Lefebvre et al 2005[5], with some modifications to consider image index r. Since we prepared our candidate set to consider rotated image indexes, correction step can successfully perform transitions between rotated images. We also use penalty constant k to keep the coordinates under control to prevent too much coordinate jumping.

$$S_i[p] := C_{j_{min}}^{i,i_{min}}(S_i^c[p + \Delta_{min}] - \Delta_{min}),\ where$$

$$j_{min}, \Delta_{min} = \underset{\substack{j \in 1..k \\ \Delta \in \{\binom{-1}{-1}, \binom{-1}{0}, \ldots, \binom{+1}{+1}\}}}{argmin} \|NS_i(p) - NE_{i,n}(u)\|\psi(j)$$

$$in\ which\ \psi(j) = \begin{cases} 1, & j = 1 \\ 1 + k, & j > 1 \end{cases} \tag{4}$$

$$i_{min} = S_i^r[p + \Delta_{min}],\ i = S_i^r[p + \Delta],$$

$$n = \text{r component of } C_j^{i,i}(S_i^c[p + \Delta] - \Delta),$$

$$u = \text{coordinate of } C_j^{i,i}(S_i^c[p + \Delta] - \Delta)$$

We also do subpass just like as introduced in Lefebvre et al 2005[5]. In our implementation of the correction step, we perform 2 correction pass and 2^2 subpass for each level.

6 Results

6.1 Speed

Our implementation of the algorithm shown that it takes about 10 seconds to preprocess 64^2 exemplar and 70 seconds for 128^2 exemplar. For actual synthesis process, it had no much difference with the GPU implementation of the algorithm from Lefebvre et al 2005[5], since our algorithm considers only one additional candidate during correction step than [5].

6.2 Synthesis results and comparison

We show some synthesized textures using our algorithm in Fig. 6. As you can see in the figure, our algorithm obtains results with rotational variations, which greatly riches the look of texture features. Also, our algorithm produces no or much less undesirable seam artifacts than previous algorithms(See Fig.4. This kind of texture has very clear edge seams when tiled due to great difference along the edge areas, previous algorithms produce practically unusable textures with a lot of texture patch seams because the algorithms cannot find proper neighborhoods to produce smooth transition between texture boundaries. However, since we extensively use rotated image features to match neighborhoods, our algorithm can find proper texture neighborhoods to obtain smooth transition between texture boundaries, resulting much smoother and natural synthesis results.

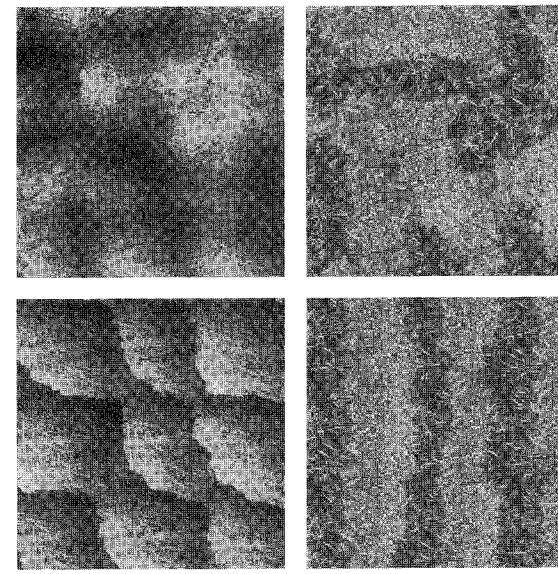


Fig. 4: Comparison of results from our algorithm and Lefebvre et al 2005[5].

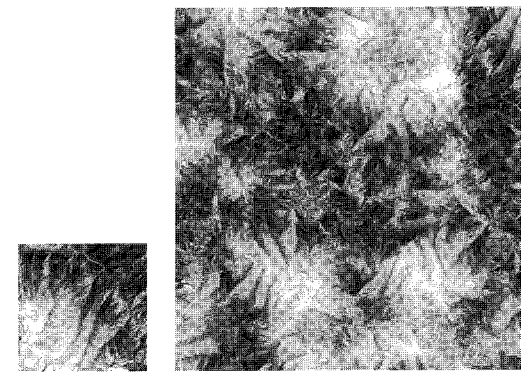


Fig. 5: A failure case of our algorithm.

7 Conclusion

We introduced a new algorithm which uses multiple rotated input exemplars to synthesize a texture of good quality. Our algorithm successfully generates fine texture results by redefining coordinate image to reflect an index of rotated exemplar pyramid and preparing candidates of rotated images for each pixel. Our algorithm can give rotational variations to the synthesis result, and seamlessly synthesizes textures which were not very appropriate

to be an exemplar for previous synthesis methods due to great difference along the texture boundaries when tiled, thus opening up some amount of textures which were not very good to be synthesized previously. The framework of our algorithm is not only limited to synthesizing a texture with multiple rotated input exemplar, but also have some possibilities to be used with two or more input exemplars. We can consider this as a work to be done in the future. Although our algorithm successfully generates nice results with most exemplars, there are some cases that our algorithm does not. Sometimes a texture can have a structure which shows very clear shades of lighting, which may get very confusing when we mix various rotated version of the texture. One such case is shown in Fig. 5. Although the result does not produce visible seams, it does not look very similar to the input exemplar due to messed up lighting condition of the result. This is a limitation of our algorithm and a problem to be solved.

8 Acknowledgements

This work was supported by the Second Brain Korea 21 Project.

References

[1] ASHIKHMIN, M., Synthesizing natural textures, Symposium on Interactive 3D Graphics, 217-226 (2001)

[2] EFROS, A., AND FREEMAN, W., Image quilting for texture synthesis and transfer, ACM SIGGRAPH, 341-346 (2001)

[3] HERTZMANN, A., JACOBS, C., OLIVER, N., CURLESS, B., AND SALESIN, D., Image analogies, ACM SIGGRAPH, 327-340 (2001)

[4] KWATRA, V., SCHODL, A., ESSA, I., TURK, G., AND BOBICK, A., Graphcut textures: image and video synthesis using graph cuts, ACM SIGGRAPH, 277-286 (2003)

[5] LEFEBVRE, S., AND HOPPE, H., Parallel controllable texture synthesis, ACM SIGGRAPH, 777-786 (2005)

[6] LEFEBVRE, S., AND HOPPE, H., Appearance-space texture synthesis, ACM SIGGRAPH, 541-548 (2007)

[7] WEI, L.-Y., HAN, J., ZHOU, K., BAO, H., GUO, B., SHUM, H.-Y., Inverse texture synthesis, ACM SIGGRAPH, 52 (2008)

[8] TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y., Synthesis of bidirectional texture functions on arbitrary surfaces, ACM SIGGRAPH, 665-672 (2002)

[9] WEI, L.-Y., AND LEVOY, M., Order-independent texture synthesis, http://graphics.stanford.edu/papers/texture-synthesis-sig03/. (Earlier version is Stanford University Computer Science TR-2002-01.) (2003)

[10] ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y., Synthesis of progressively-variant textures on arbitrary surfaces, ACM SIGGRAPH, 295-302 (2003)

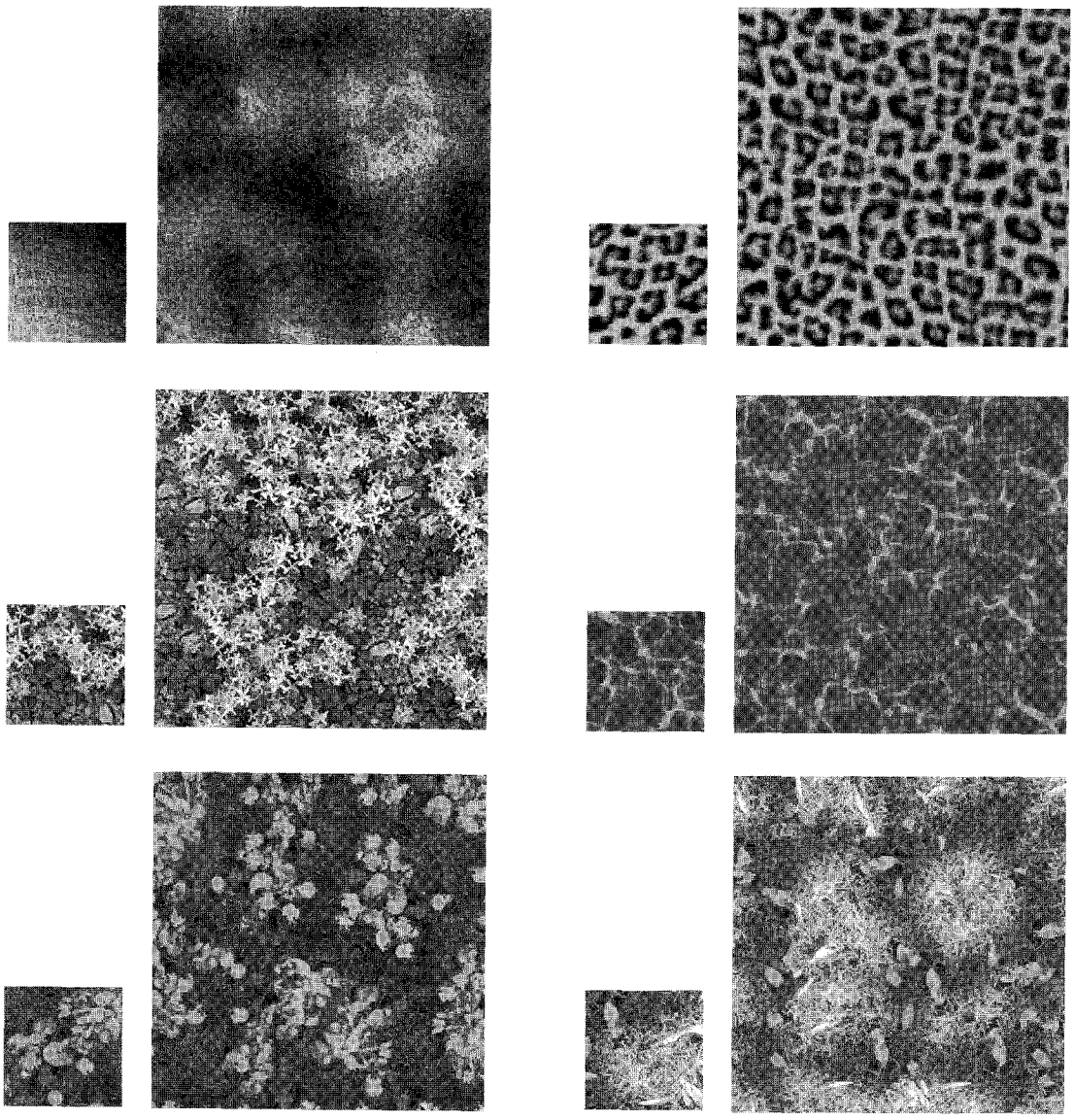
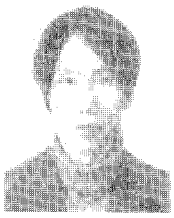


Fig. 6: Synthesis results of our algorithm.

〈저자소개〉



Han-wook Park
2003.03 ~ 2007.02 Bachelor's degree in Dept. of Computer Science and Engineering, Korea University, Korea
2007.03 ~ 2009.02 Master's Degree in Dept. of Computer Science and Engineering, Korea University, Korea
His current research interests include texture synthesis and image processing.

Chang-hun Kim
1975.03 ~ 1979.03 B.S. in Dept. of Economics, Korea University, Korea
1990.04 ~ 1993.02 Ph.D. in Dept. of Computer Science, University of Tsukuba, Japan
1979.01 ~ 1987.02 Senior Researcher, Korea Institute of Science and Technology(KIST)
1993.02 ~ 1995.02 Principal Researcher, SERI, KIST
2003.02 ~ 2004.01 Visiting Professor, UCLA
1995.03 ~ Now Professor, Dept. of Computer Science & Engineering, Korea University, Korea
2000.10 ~ Now Vice-Chairman, The Korea Computer Graphics Society(KCGS)
2005.01 ~ Now The trustee of Korea Information Science Society
2005.11 ~ Now President of Information and Communication University, Korea University, Korea
2005.11 ~ Now President of Computer Science and Technology graduate school, Korea University, Korea
His current research interests include fluid animation and mesh processing.