The Visual Computer manuscript No.
(will be inserted by the editor)

Jianwei Han · Kun Zhou · Li-Yi Wei · Minmin Gong
Hujun Bao · Xinming Zhang · Baining Guo

Fast Example-based Surface Texture Synthesis via Discrete Optimization

Abstract We synthesize and animate general texture patterns over arbitrary 3D mesh surfaces. The animation is controlled by flow fields over the target mesh, and the texture can be arbitrary user input as long it satisfies the Markov-Random-Field assumptions. We achieve this by extending the texture optimization framework over 3D mesh surfaces. We propose an efficient discrete solver inspired by k-coherence search, allowing interactive flow texture animation while avoiding the blurry blending problem for the least square solver in previous work. Our technique has potential applications ranging from simulation, visualization, and special effects.

keywords: texture synthesis, texture mapping, flow visualization, texture animation, energy minimization

1 Introduction

We present a technique that allows us to animate a given texture pattern over arbitrary 3D mesh surfaces. The animation is controlled by a (possibly dynamic) flow field over the target mesh, and the texture can be arbitrary user input; the only requirement is that it must satisfy the Markov-Random-Field (MRF) assumptions so that we can perform texture synthesis. In addition, our computation is fast and the rendered animation is frame-coherent.

We achieve this by extending the texture optimization framework [4] over 3D mesh surfaces [14,17]. We propose an efficient numerical solver inspired by k-coherence search [11,5], allowing fast flow texture animation via a GPU implementation (via an approach similar to [5]). We dub our technique *discrete optimization* due to the discrete natural of our k-coherence solver (versus the continuous natural of a least square solver).

Our technique is simple to implement, and has a variety of potential applications such as visualization and special effects rendering.

Jianwei Han, Hujun Bao
Zhejiang University
Tel.: +86-0571-88206681
Fax: +86-0571-88206680
E-mail: hanjianwei,bao@cad.zju.edu.cn

Kun Zhou, Li-Yi Wei, Baining Guo
Microsoft Research Asia
Tel.: +86-10-62617711
Fax: +86-10-88097306
E-mail: kunzhou,lywei,bainguo@microsoft.com

Minmin Gong, Xinming Zhang
University of Science and Technology of China
Tel.: +86-551-3603145
Fax: +86-551-3601013
E-mail: gongminmin@msn.com,xinming@ustc.edu.cn

This work was done while Jianwei Han and Minmin Gong were visiting students at Microsoft Research Asia.

1.1 Background

Textures have long been used for visualizing flow fields over either regular 2D/3D grids or curved 3D surfaces [3,12]. The majority of these methods are based on line integral convolution (LIC) [1], utilizing random noises as textures for conveying flow fields. Most of these visualization methods use static images. [18] extended LIC-based visualization as animation via coherent warping between adjacent frames. Most of these methods, however, utilize random noises as the visualization texture. Even though noise is effective in conveying detailed flow fields, it is not suitable for simulating more general flow fields such as water, smoke, or fire.

The major challenge for animating general texture flows is to ensure frame-to-frame coherence; this is particularly tricky around flow singularities such as sources or sinks where texture pattern appears or disappears. For random noises, frame coherence can be enforced via proper warping [18] and singularities can be trivially handled since noise is inherently chaotic. However, for more general textures, care has to be taken to ensure that the texture patterns evolve smoothly and naturally when animating with the flow fields. [10] proposed procedural methods for synthesizing certain classes of flows over arbitrary mesh surfaces, but since it

is procedural, it cannot be utilized for animating arbitrary user given textures. [4] animates flow fields for any user input patterns; the technique produces stunning visualization effects, but has so far been limited to 2D image grids. In addition, the presented technique is computationally expensive which greatly restricts its applicability.

Our goal is to extend [4] over 3D mesh surfaces to allow animation of user-input textures over arbitrary 3D surface flow fields. Due to the use of pixel-wise optimization in [4], we adopt the mesh neighborhood sampling methods in [17,14] as opposed to some patch-based algorithms [8,9]. In particular, we synthesize textures over mesh vertices as in [17,14], and we adopt their methodology for re-meshing, assigning orientation fields, synthesis ordering, and multi-resolution synthesis.

Another related issue is computational speed. Even though most texture synthesis algorithms have been utilized as an offline process (with a notable exception in [5,6]), computation speed is important for our application since we need to synthesize multiple frames of textures over dense polygonal meshes. Unfortunately, despite its high quality, texture optimization in [4] can be slow due to its particular search and minimization algorithms. Fortunately, extensive research has been performed for synthesis acceleration; as reported in [15,5], k-coherence search [11] provides the best tradeoff in terms of quality and speed for an efficient parallel implementation. We adopt k-coherence into our optimization framework, and propose a discrete solver which addresses both the speed and quality issue in the original solver proposed by [4]. As a recent work, [6] extends the GPU synthesis in [5] to arbitrary surfaces, but the underlying algorithm is k-coherence synthesis, different from our optimization algorithm.

The fields of flow visualization and texture synthesis are both vast, and it is beyond our paper to provide a complete coverage; for more detailed surveys, we refer the readers to [18,12] for flow field visualization and [19,4,5,6] for texture synthesis.

1.2 Our Contribution

Our major contributions are as follows.

We combine texture optimization [4] and synthesis by neighborhood sampling [14,17] to achieve controllable, frame-coherent texture animation over general input textures and output meshes. To our knowledge, this combination has not been attempted before, and it requires non-trivial extensions of previous algorithms. In particular, we have to re-cast the energy function in [4] so that the optimization variables lie on irregular mesh vertices instead of regular-grid image pixels.

In addition, the original EM-solver [4] is too computationally expensive for real-time or interactive applications. We propose a novel discrete solver based on k-coherence search [11], allowing a fast, quality neighborhood search in the M-step, while avoiding the blurry blending problem in the E-step. As an added advantage, our k-coherence solver allows a GPU-friendly implementation, resulting in further acceleration of our algorithm.

2 Algorithm

Our algorithm extends [4] over 3D mesh surfaces for synthesizing static textures and dynamic flow visualizations. For clarity of exposition, we begin with a brief review of [4]. We then present our modifications over [4] for surface synthesis.

For easy reference, we summarize [4] and our algorithm as pseudo-code in Table 1. We also highlight the differences of these two algorithms in the table caption.

2D Image Synthesis

$\mathbf{z}_p^0 \leftarrow$ random neighborhood in $Z \; \forall$ p $\in X^\dagger$
for iteration n = 0:N **do**
  $\mathbf{x}^{n+1} \leftarrow \text{argmin}_{\mathbf{x}} \; [E_t(\mathbf{x}; \{\mathbf{z}_p^n\}) + \lambda E_c(\mathbf{x}; \mathbf{u})]$ // *E-step*
  $\mathbf{z}_p^{n+1} \leftarrow \text{argmin}_{\mathbf{z}} \; [|\mathbf{x}_p - \mathbf{z}|^2 + \lambda E_c(\mathbf{y}; \mathbf{u})]$ // *M-step*
  // **z** *is a neighborhood in* Z *and* **y** *is the same as* **x**
  // *except for neighborhood* $\mathbf{x}_p$ *which is replaced with* **z**
  if $\mathbf{z}_p^{n+1} == \mathbf{z}_p^n \; \forall$ p $\in X^\dagger$
    x = $\mathbf{x}^{n+1}$
    break
  end if
end for

3D Mesh Synthesis

$\mathbf{z}_p^0 \leftarrow$ random neighborhood in $Z \; \forall$ p $\in X^\dagger$
for iteration n = 0:N **do**
  $\mathbf{x}^{n+1} \leftarrow \text{argmin}_{\mathbf{x}, \mathbf{x}(p) \in \mathbf{k}(p) \forall p} \; [E_t(\mathbf{x}; \{\mathbf{z}_p^n\}) + \lambda E_c(\mathbf{x}; \mathbf{u})]$ // *E-step*
  $\mathbf{z}_p^{n+1} \leftarrow \text{argmin}_{\mathbf{z}} \; [|\mathbf{W}_p \mathbf{x} - \mathbf{z}|^2 + \lambda E_c(\mathbf{y}; \mathbf{u})]$ // *M-step*
  // $\mathbf{W}_p$ *is the interpolation matrix so that* $\mathbf{x}_p$ = $\mathbf{W}_p$ **x**
  if $\mathbf{z}_p^{n+1} == \mathbf{z}_p^n \; \forall$ p $\in X^\dagger$
    x = $\mathbf{x}^{n+1}$
    break
  end if
end for

Table 1 Pseudocode. The top portion is for [4], while the bottom portion our algorithm. The major differences include (1) The output variables **x** indicates image pixel colors in [4] but mesh vertex colors in our case, (2) the output neighborhood $\mathbf{x}_p$ is from regular image grid in [4] but interpolated from mesh vertex colors in our case, (3) the restriction of each output vertex color **x**(p) to its k-coherence candidate set **k**(p) in the E-step, and (4) we utilize k-coherence as the search algorithm in the M-step.

2.1 Brief Review of [4]

Unlike most previous work based on greedy heuristics, [4] synthesizes textures by optimization. Specifically, the set of output pixel colors **x** is treated as a high-dimensional variable, and its value is determined by energy minimization. The energy function $E(\mathbf{x})$ measures the perceptual difference between the input and output based on a simple local

neighborhood metric [2,16]. For constrained synthesis such as frame-coherent animation, the energy function also incorporates output color constraints **u**. This energy function $E(\mathbf{x})$ can be summarized as follows:

$$E(\mathbf{x}) = E_t(\mathbf{x}; \{\mathbf{z}_p\}) + \lambda E_c(\mathbf{x}; \mathbf{u})$$
$$E_t(\mathbf{x}; \{\mathbf{z}_p\}) = \sum_{p \in X^\dagger} |\mathbf{x}_p - \mathbf{z}_p|^2 \quad (1)$$

, where E_t measures local neighborhood similarity across the current active subset $X^\dagger$ of the output ($\mathbf{z}_p$ indicates the most similar input neighborhood to each output neighborhood $\mathbf{x}_p$), E_c imposes color constraints as detailed in [4] (Section 4), and λ weighs these two energy terms differently according to user preference.

[4] solves the energy function via an EM-like algorithm; in the E (expectation) step, the set of matching input neighborhoods $\{\mathbf{z}_p\}$ remains fixed and the set of output pixels **x** is solved via least-square method; in the M (maxization) step, the set of output pixels **x** remains fixed and the set of matching input neighborhoods $\{\mathbf{z}_p\}$ is found by searching. These two steps are iterated multiple times until convergence, or a maximum number of iterations is reached. Please refer to Table 1 for math details of these two steps.

This energy minimization framework blends the flavor of both pixel and patch based algorithms; while the neighborhood metric is pixel-centric, the global optimization considers multiple pixels together, bearing resemblence to patch-based algorithms.

2.2 Our Approach

In our approach, we synthesize textures as vertex colors directly over the target mesh surface (similar to [14,17]); as a result, our output variable **x** is defined over mesh vertices rather than image pixels. We adopt the surface neighborhood sampling idea from [14,17] so that we could utilize the pixel-grid optimization in [4] to mesh surfaces. Below, we detail the necessary extensions and modifications.

2.2.1 Resampling $\mathbf{x}_p$ from **x**

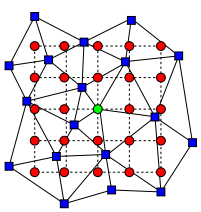

Fig. 1 Resampling mesh neighborhoods. The green circle indicates the current vertex p to be synthesized, and the blue mesh indicates the local region on the target surface around p. The red grid indicates resampled neighborhood. In our implementation, we utilize [14] for resampling.

Since output colors **x** are defined over irregularly sampled mesh vertices, we have to resample each output neighborhood into a regular grid $\mathbf{x}_p$ in order to perform pixel-wise comparison as shown in the E_t term in Equation 1. This can be done by the neighborhood flattening and resampling idea from [14,17]. Specifically, each resampled output neighborhood can be expressed as a linear combination of nearby vertex colors:

$$\mathbf{x}_p = \mathbf{W}_p \mathbf{x} \quad (2)$$

, where $\mathbf{W}_p$ is a per-vertex sparse interpolation matrix relating $\mathbf{x}_p$ to **x**. For static meshes, the set of interpolation weights $\mathbf{W}_p$ can be pre-computed and stored with each output vertex, to reduce run-time computation cost.

Based on this representation of re-sampled $\mathbf{x}_p$, we can re-write E_t as follows:

$$E_t(\mathbf{x}; \{\mathbf{z}_p\}) = \sum_{p \in X^\dagger} |\mathbf{W}_p \mathbf{x} - \mathbf{z}_p|^2 \quad (3)$$

Note that this equation is still a quadratic energy function, allowing fast least-square solvers as in the original algorithm [4].

In our mesh synthesis algorithm shown in Table 1, we have replaced all occurances of $\mathbf{x}_p$ with $\mathbf{W}_p\mathbf{x}$.

2.2.2 Discrete solver based on k-coherence

The solver in [4] utilized hierarchical tree search for the M-step and least squares for the E-step; however, tree search has an average time complexity of $O(log(N))$ where N is the total number of input neighborhoods, and this step can easily becomes the bottleneck of the solver as reported in [4].

In our solver, we adopt an alternative search method for the M-step. Specifically, we have chosen k-coherence search [11] due to its constant time complexity per search; in addition, its quality is satisfactory as reported in [5]. The k-coherence algorithm is divided into two phases: analysis and synthesis. During analysis, the algorithm builds a similarity-set for each input pixel, where the similarity-set contains a list of other pixels with similar neighborhoods to the specific input pixels. During synthesis, the algorithm builds a candidate-set by taking the union of all similarity-sets of the neighborhood pixel/vertex for each output pixel/vertex, and then searches through this candidate-set to find out the best match. The size of the similarity-set, K, is a user-controllable parameter that determines the overall speed/quality.

Unfortunately, a direct adoption of k-coherence into the solver in [4] is impossible due to the inherent incompatibility between k-coherence (in M-step) and least square (in E-step). In particular, k-coherence requires the bookkeeping of the source pixel locations (x, y) for each output pixel/vertex, and this location information is lost during least square solver (and in general, any method beyond direct pixel copying).

We overcome this problem by adopting a different approach for the E-step other than least squares. The algorithm can be considered as a discrete optimization, as follows. To

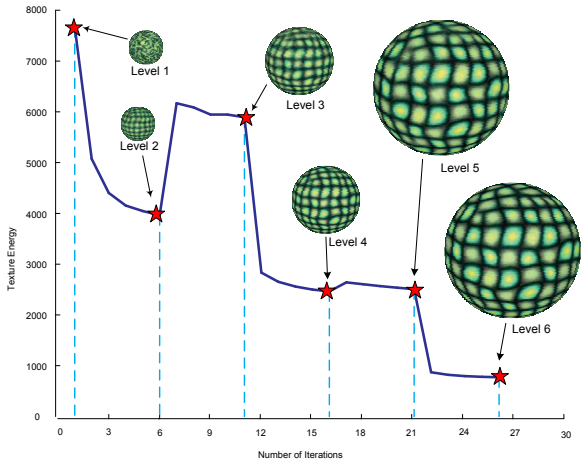

Fig. 2 Texture energy plotted as a function of number of iterations. The energy is normalized with respect to output resolutions as in [4]. Also shown is the synthesized texture after each resolution and scale (neighborhood size) level.

compute $\mathbf{x}^{n+1}$ in the E-step, each one of its values $\mathbf{x}$(p) at vertex p is determined independently from each other. In particular, $\mathbf{x}$(p) for the next iteration is chosen from its k-coherence candidate set $\mathbf{k}$(p), as the one that most reduces the energy function. Since now each $\mathbf{x}$(p) is copied direct from some input pixel, we can retain the input location information to conduct k-coherence search in the M-step.

As an added benefit, since our solver does not blend pixels, we do not suffer from the blurry blending problem as reported in [4].

Figure 2 illustrates the evolution of a synthesis result throughout multiple iterations of our algorithm.

2.3 Further Details

Here, we describe further implementation details beyond our basic algorithm.

Multi-resolution synthesis As demonstrated in [16], a multi-resolution framework allows us to capture large scale texture structures without the need for large neighborhoods, which can cause efficiency and stability issues.

In our approach, we build multi-resolution pyramids for both the input image and output mesh, and we synthesize the output from lower to higher resolutions. At the lowest resolution, we simply randomly copy sample colors from input to output since at this low resolution the texture is essentially random. (For texture images satisfying the MRF assumptions, this is always achievable with a deep enough pyramid.) When synthesizing a higher resolution, we first initialize it by up-sampling from the already synthesized immediate lower resolution. We then perform synthesis on this level via our algorithm as described in above.

case	input size	output size	[4]	CPU	GPU
a	64^2	48K	7.87	2.74	0.61
b	128^2	48K	53.66	2.97	0.66
c	64^2	48K	12.39	2.14	0.59
d	127×94	48K	30.84	1.91	0.62
e	64^2	48K	8.02	2.46	0.61
f	64^2	48K	10.72	2.82	0.61
g	128^2	48K	55.36	2.45	0.70
h	128^2	48K	79.79	1.83	0.68

Table 2 Statistics of synthesis results in Figure 3. The input size is measured in pixels whereas the output size in vertices. The right-most three columns demonstrate total synthesis time per M+E steps in seconds via [4], our technique on CPU, and our technique on GPU. For each case, we use a 3-level pyramid and within each level we perform $1\sim3$ iterations of our algorithm with neighborhood size 17^2 followed by $1\sim3$ iterations with neighborhood size 9^2. All performance timings are measured on the following platform: CPU (Pentium 4 3.2 GHz) and GPU (NVIDIA Geforce 7800 GT). We utilize kd-tree in ANN [7] for implementing the search algorithm in [4].

Output mesh retiling As a pre-process, we retile the output mesh using [13]. This allows us to control the texture density with a more uniform vertex distribution. As observed in [14, 16], the retiling is indispensable for such vertex-coloring synthesis techniques. (This retiling process can be skipped if the texture is synthesized into an atlas [20] rather than individual vertices.)

For multi-resolution synthesis, we build a mesh hierarchy via simplification (via [13]) and retile each resolution independently. The retiling density is controlled so that it is roughly 4:1 between adjacent pyramid levels. In addition, we pre-compute and store correspondences between each vertex and its parent triangles at the lower resolution, in order to accelerate the run-time up-sampling process as described above.

GPU acceleration Since both our E-step and M-step utilizes k-coherence search as the core algorithm, our entire synthesis process can be implemented on GPU in a method similar to [5]. Specifically, we store the input $\mathbf{z}$, the output $\mathbf{x}$, and the match $\mathbf{z}_p$ as textures, and implement each E and M step as a separate fragment program. The entire synthesis process is iterated via multi-pass rendering, where the new values are written into proper render targets for $\mathbf{x}$ and $\mathbf{z}_p$.

During the n^{th} E-step, our E fragment program reads from $\mathbf{z}$ and $\mathbf{x}^n$ textures, performs the discrete optimization, and writes the new result $\mathbf{x}^{n+1}$ into the proper render target, which serves as the input for the n^{th} M-step.

During the n^{th} M-step, our M fragment program reads from $\mathbf{z}$ and $\mathbf{x}^{n+1}$, performs the necessary k-coherence search, and writes the new result $\mathbf{z}_p^{n+1}$ into the proper render target.

3 Results

We have applied our algorithm over a variety of input textures and output mesh models, as shown in Figure 3. (Please

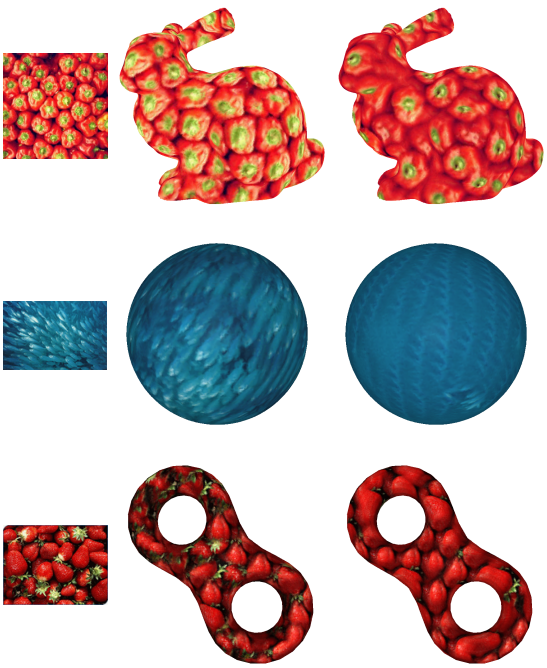

Fig. 4 Comparison of our solver with the least square one in [4]. For each group of images, the input is on the left, our result is in the middle, and the result via least square solver [4] is on the right.

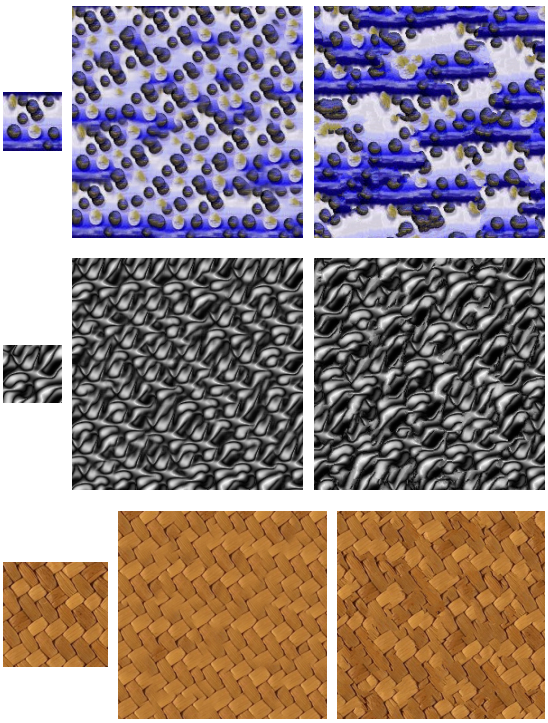

Fig. 5 2D synthesis quality comparison. For each group of images, the input is on the left, our result is in the middle, and the result via [5] is on the right.

note that all of these images are screen shots of texture animations; please refer to our accompany video for full dynamic animation effects.)

For quality comparison, in Figure 4 we have attached two results generated by our k-coherence solver and the least square solver [4]. Notice that our k-coherence solver produces crispier image quality due to its use of copying rather than blending. A further advantage of our technique is that it is much faster than [4] when both running on CPU with further speed improvement via out GPU implementation; see our timing measurements in Table 2,

Despite our speed and quality improvement over texture optimization [4], one inherent limitation of optimization is that it will always be slower than [5], which utilizes a local greedy search. However, the advantage of our approach is that we produce superior synthesis quality than [5] due to our use of optimization, as demonstrated in Figure 5.

4 Conclusions and Future Work

We have presented a surface texture synthesis and animation algorithm based on optimization. Our basic idea is to combine neighborhood-based surface synthesis [17, 14] with optimization [4] to achieve high quality, frame coherent texture animation over arbitrary 3D object surfaces. On top of this basic idea, we have proposed a variety innovations for quality and speed improvements, among which a discrete solver based on k-coherence that allows both faster computation speed and crispier image quality than the original solver in [4]. Our discrete solver further enables a GPU friendly implementation, with even more speed improvement over our CPU-only implementation.

Our initial goal of this project was to simply combine surface synthesis [17, 14] and optimization [4] to allow high quality, offline surface texture animation. Our discovery of the discrete solver as both a quality and speed improvement came only as a second thought near the end game of our project. For future work, we plan to investigate other possibilities for optimization solvers in order to achieve greater quality and speed. In particular, both our current CPU and GPU implementations are not yet real-time, and we envision a future technique that combines the quality of our work with the speed in [5, 6] will be invaluable for a variety of applications, ranging from offline movie production to real-time gaming.

Acknowledgement We would like to thank the anonymous reviewers for their comments. The ZJU authors were partially supported by 973 Program of China (No. 2002CB312104), NSFC (No. 60021201) and Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20030335083). Dr. Zhang's work is partially supported by the International Scholar Exchange Fellowship (ISEF) of the Korea Foundation for Advanced Studies 2005-2006.

References

1. Cabral, B., Leedom, L.C.: Imaging vector fields using line integral convolution. In: SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp. 263–270 (1993)
2. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2, p. 1033 (1999)
3. Interrante, V., Grosch, C.: Visualizing 3d flow. IEEE Comput. Graph. Appl. **18**(4), 49–53 (1998)
4. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. ACM Trans. Graph. **24**(3), 795–802 (2005)
5. Lefebvre, S., Hoppe, H.: Parallel controllable texture synthesis. ACM Trans. Graph. **24**(3), 777–786 (2005)
6. Lefebvre, S., Hoppe, H.: Appearance space texture synthesis. ACM Trans. Graph. (2006). To appear
7. Mount, D.M., Arya, S.: Ann: A library for approximate nearest neighbor searching (2005). `http://www.cs.umd.edu/~mount/ANN/`
8. Praun, E., Finkelstein, A., Hoppe, H.: Lapped textures. In: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 465–470 (2000)
9. Soler, C., Cani, M.P., Angelidis, A.: Hierarchical pattern mapping. In: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 673–680 (2002)
10. Stam, J.: Flows on surfaces of arbitrary topology. ACM Trans. Graph. **22**(3), 724–731 (2003)
11. Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B., Shum, H.Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. In: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 665–672 (2002)
12. Tong, Y., Lombeyda, S., Hirani, A.N., Desbrun, M.: Discrete multiscale vector field decomposition. ACM Trans. Graph. **22**(3), 445–452 (2003)
13. Turk, G.: Re-tiling polygonal surfaces. In: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, pp. 55–64 (1992)
14. Turk, G.: Texture synthesis on surfaces. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 347–354 (2001)
15. Wei, L.Y., Levoy, M.: Order-independent texture synthesis. `http://graphics.stanford.edu/papers/texture-synthesis-sig03/`. (Earlier version is Stanford University Computer Science TR-2002-01.)
16. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 479–488 (2000)
17. Wei, L.Y., Levoy, M.: Texture synthesis over arbitrary manifold surfaces. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 355–360 (2001)
18. van Wijk, J.J.: Image based flow visualization. In: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 745–754 (2002)
19. Zhang, J., Zhou, K., Velho, L., Guo, B., Shum, H.Y.: Synthesis of progressively-variant textures on arbitrary surfaces. ACM Trans. Graph. **22**(3), 295–302 (2003)
20. Zhou, K., Du, P., Wang, L., Shi, J., Guo, B., Shum, H.Y.: Decorating surfaces with bidirectional texture functions. IEEE Transactions on Visualization and Computer Graphics **11**(5), 519–528 (2005)

Jianwei Han received his B.S. degree in computer science from Zhejiang University in 2003. Currently he is a Ph.D. candidate in the State Key Laboratory of CAD&CG of Zhejiang University as well a visiting student at Microsoft Research Asia. His research interest is texture synthesis.

Kun Zhou is a researcher/project lead of the graphics group at Microsoft Research Asia. He received his B.S. and Ph.D. in Computer Science from Zhejiang University in 1997 and 2002 respectively. His current research focus is geometry processing, texture processing and real time rendering. He holds over 10 granted and pending US patents. Many of these techniques have been integrated in Windows Vista, DirectX and XBOX SDK.

Li-Yi Wei is a researcher at Microsoft Research Asia. Prior to that, he has been architecting NVIDIA's graphics chips from 2001 to 2005. He received a Ph.D. in Electrical Engineering from Stanford University, and a B.S. in Electrical Engineering from National Taiwan University.

Minmin Gong is a second year master student of Institute of Software, University of Science and Technology of China. Before that, he received his B.S. Degree in Computer Science from Beijing Technology and Business University in 2004. His research interests include real-time rendering, natural phenomena simulation and texture compression.

Hujun Bao received his Bachelor and PhD in applied mathematics from Zhejiang University in 1987 and 1993. His research interests include modeling and rendering techniques for large scale of virtual environments and their applications. He is currently the director of State Key Laboratory of CAD&CG of Zhejiang University. He is also the principal investigator of the virtual reality project sponsored by Ministry of Science and Technology of China.

Xinming Zhang was born in 1964 and received his PhD in Computer Science from USTC. He is currently an Associate Professor at USTC and a Visiting Professor at KAIST. His primary research area is Wireless Networks.

Baining Guo is the research manager of the Internet Graphics group at Microsoft Research Asia. Before joining Microsoft, Baining was a senior staff researcher in Microcomputer Research Labs at Intel Corporation in Santa Clara, California, where he worked on graphics architectures. Baining received his Ph.D. and M.S. from Cornell University and his B.S. from Beijing University. Baining is an associate editor of IEEE Transactions on Visualization and Computer Graphics. He holds over 30 granted and pending US patents.

Fig. 3 Surface synthesis results. For each group of images, the input is on the left and our result is on the right. Please refer to our accompany video for animation effects.