Huang HD, Tong X, Wang WC. Accelerated parallel texture optimization. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 22(5): 761~769 Sept. 2007

Accelerated Parallel Texture Optimization

Hao-Da Huang[1,3] (黄浩达), Xin Tong[2] (童　欣), and Wen-Cheng Wang[1] (王文成)

[1] *State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China*

[2] *Microsoft Research Asia, Beijing 100080, China*

[3] *Graduate University of Chinese Academy of Sciences, Beijing 100080, China*

E-mail: haoda.huang@gmail.com; xtong@microsoft.com; whn@ios.ac.cn

Received November 7, 2006; revised June 19, 2007.

Abstract Texture optimization is a texture synthesis method that can efficiently reproduce various features of exemplar textures. However, its slow synthesis speed limits its usage in many interactive or real time applications. In this paper, we propose a parallel texture optimization algorithm to run on GPUs. In our algorithm, k-coherence search and principle component analysis (PCA) are used for hardware acceleration, and two acceleration techniques are further developed to speed up our GPU-based texture optimization. With a reasonable precomputation cost, the online synthesis speed of our algorithm is 4000+ times faster than that of the original texture optimization algorithm and thus our algorithm is capable of interactive applications. The advantages of the new scheme are demonstrated by applying it to interactive editing of flow-guided synthesis.

Keywords texture synthesis, energy minimization, parallel, GPU, flow visualization

1 Introduction

Texture synthesis aims at generating visually similar large textures from a small exemplar texture. It has many applications in computer graphics and computer vision, and thus received much attention in recent years. In general, texture synthesis methods can be categorized as either local ones or global ones. Local texture synthesis methods grow a texture pixel by pixel[1~6] or patch by patch[7~11]. They have achieved much success in either constrained or unconstrained texture synthesis. Global texture synthesis methods evolve a texture as a whole based on some similarity measurement[12~18]. As local methods grow pixels or patches by previously grown pixels and patches, they are order-dependent and thus cannot be implemented in parallel. Moreover, small errors may be accumulated over large distances leading to inconsistencies in the synthesized texture. On the contrary, global methods update pixels or patches simultaneously[14,16~18], so they are order-independent and have the potential to run parallel on GPUs. The error accumulation problem is also avoided in global methods, and the global methods proposed recently have revealed many interesting functions such as controlling the texture variability or allowing flow-guided synthesis[15~18]. Thus, global texture synthesis methods have drawn much attention in recent years.

Among existing global texture synthesis methods, *texture optimization*[15] is especially interesting. It formulates texture synthesis as an energy minimization problem, and uses simple iteration operations to optimize the target texture. Besides producing state-of-the-art texture synthesis quality, texture optimization also affords additional flexibility to perform controllable synthesis like flow-guided synthesis. Unfortunately, this method is slow, limiting its usage in many real time applications, such as drag-and-drop control on texture synthesis and interactive editing of texture flows.

In this paper, we firstly propose a basic parallel scheme to run texture optimization on GPUs, by using k-coherence search for finding nearest neighborhoods and introducing PCA (principle component analysis) to reduce the cost on neighborhood comparison. Then we proposed two acceleration techniques to speed up our basic parallel scheme by avoiding redundant synthesis work. With a reasonable precomputation cost, our accelerated parallel scheme can produce a 256×256 texture from a 64×64 sample texture in 77~87ms, about 4000+ times faster than the original texture optimization[15] and more than two times faster than

Regular Paper

The IOS authors are partially supported by the National High Technology Development 863 Program of China under Grant No. 2006AA01Z306 and the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312102.

our basic parallel scheme.

The remainder of this paper is organized as follows. A brief overview of the related work is given in Section 2. The new scheme is described in details in Section 3 and its application, the interactive flow-guided synthesis, is described in Section 4. After experimental results are given and discussed in Section 5, a summary is drawn in Section 6.

2 Related Work

Texture synthesis methods have been proposed in a large quantity. It is beyond the scope of this paper to survey them all. Here, we mainly discuss the global methods for texture synthesis.

No matter whether a global method or a local method is used, a target texture is produced either pixel-by-pixel or patch-by-patch. Generally speaking, the patch-based methods perform better to reproduce semantic information of a texture[8~12] and can save time on synthesis, while the pixel-based methods are more flexible and thus more suitable for constrained synthesis[2~4].

Many global texture synthesis methods work by pixels. An order-independent texture synthesis method was proposed in [14] to run by multi-resolution synthesis with several passes of pixel correction in each resolution, where all the pixels could be corrected independently and simultaneously. Based on this work, a parallel texture synthesis method was proposed in [16], which could synthesize new texture in real-time on GPUs. It is also able to control the variation of output textures. Afterwards, the method in [16] was improved by synthesizing textures in a transformed appearance space to promote texture quality[18]. However, because these global methods are pixel-based, they may fail to reproduce some semantic features in example textures, and so lower the synthesis quality.

Texture optimization[15] is a global texture synthesis method which is intermediate between pixel-based and patch-based methods. It formulates the synthesis problem as a minimization problem of an energy function and refines the output texture progressively through an Expectation Maximization (EM)-like algorithm. Here, the neighborhood size used to define texture energy determines the granularity at which synthesis is performed. The use of large neighborhoods gives texture optimization a patch-based flavor, while each pixel value is allowed to change, giving texture optimization a pixel-based flavor. Thus, it can better reproduce semantic structures of an exemplar texture than pixel-based methods, and also better perform constrained synthesis than patch-based methods. Although texture optimization has many advantages, it is not fast enough for interactive applications. So in this paper, a fast parallel scheme is proposed to promote the synthesis speed of texture optimization.

To our knowledge, there is another work to implement texture optimization on GPUs by utilizing its inherent synthesis parallelism[17], but our new scheme is about 20+ times faster than [17]. This is because our scheme has two more improvements: 1) the PCA is adopted to reduce the cost on neighborhood comparison; 2) we utilize local pixel coherency and local neighborhood stability to reduce redundant synthesis work.

3 Accelerated Parallel Synthesis Scheme

This paper is to present a scheme to accelerate texture optimization. Before discussing the scheme, we give a brief overview of texture optimization in Subsection 3.1. Then, in Subsection 3.2, we discuss how to implement texture optimization on GPUs in parallel. After that, two techniques are developed to accelerate the parallel scheme in Subsection 3.3, using two properties in the synthesis process to reduce redundant computation.

3.1 Brief Review of Texture Optimization

For ease reference, we list the pseudo-code of [15] in Fig.1. Here, Z denotes the input exemplar texture, X denotes the output texture, $\boldsymbol{x}$ is the vectorized version of X, $\boldsymbol{x}_p$ (sub-vector of $\boldsymbol{x}$) denotes the pixel neighborhood centered at pixel p, and $\boldsymbol{z}_p$ is the vectorized pixel neighborhood in Z that is most similar to $\boldsymbol{x}_p$. Then the texture energy function is defined as $E(\boldsymbol{x}) = E_t(\boldsymbol{x}; \{\boldsymbol{z}_p\}) + \lambda E_c(\boldsymbol{x}; \boldsymbol{u})$, which measures the similarity between output texture and input texture. The first term $E_t(\boldsymbol{x}; \{\boldsymbol{z}_p\})$ $\left(= \sum_{p \in X^+} |\boldsymbol{x}_p - \boldsymbol{z}_p|^2 \right)$ measures the local neighborhood similarity across X^+ and the second term represents the user-specified constraints. X^+ is a subset of X and it is empirically chosen to consist of the neighborhood centers that are $w/4$ pixels apart, where w is the width of each neighborhood. This is for saving time and preventing the synthesized texture from getting too blurry in regions where there is a mismatch between overlapping neighborhoods.

As texture optimization uses an EM-like algorithm to minimize the texture energy function, it executes its E-step and M-step as follows, also illustrated in Fig.1. In the E-step, $E(\boldsymbol{x})$ is minimized through updating output pixels $\boldsymbol{x}$ with neighborhoods $\{\boldsymbol{z}_p\}$ unchanged, where each pixel of $\boldsymbol{x}$ is set to the average of the corresponding values from the neighborhoods overlapping it and user-specified constraints. In the M-step, $E(\boldsymbol{x})$ is minimized through updating neighborhoods $\{\boldsymbol{z}_p\}$ with output pixels $\boldsymbol{x}$ unchanged, where $\boldsymbol{z}_p$ is found by hierarchical tree search. After several iterations of E-step

and M-step, the energy function would finally converge and the output texture is produced.

Algorithm. Controllable Texture Synthesis
$\boldsymbol{z}_p^0 \leftarrow$ random neighborhood in $Z \forall p \in X^+$
for iteration $n = 0 : N$ **do**
  $\boldsymbol{x}^{n+1} \leftarrow \arg\min_x [E_t(\boldsymbol{x}; \{\boldsymbol{z}_p^n\}) + \lambda E_c(\boldsymbol{x}; \boldsymbol{u})]$
  //E-step
  $\boldsymbol{z}_p^{n+1} \leftarrow \arg\min_v [\|\boldsymbol{x}_p - \boldsymbol{v}\|^2 + \lambda E_c(\boldsymbol{y}; \boldsymbol{u})]$
  //M-step, $\boldsymbol{v}$ is a neighborhood in Z and $\boldsymbol{y}$ is the
  //same as $\boldsymbol{x}$ except for neighborhood $\boldsymbol{x}_p$ which is
  //replaced with $\boldsymbol{v}$
  if $\boldsymbol{z}_p^{n+1} = \boldsymbol{z}_p^n \forall p \in X^+$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}^{n+1}$
    break
  end if

Fig.1. Pseudo-code of texture optimization.

In texture optimization described above, most of the synthesis time is spent on M-steps due to the following two factors: 1) the large number of input neighborhoods for nearest neighborhood search, and 2) expensive neighborhood comparison ($\|\boldsymbol{x}_p - \boldsymbol{v}\|^2$) for large size neighborhoods. Although the hierarchical structure is used to organize the neighborhoods for reducing the complexity on nearest neighborhood search, such search is still time consuming. Moreover, hierarchical tree search involves recursive operations that prevent this texture synthesis method from implementing on GPUs. These two problems will be solved in our GPU-based texture optimization algorithm.

3.2 GPU-Based Texture Optimization

To accelerate texture optimization for interactive applications, we introduce k-coherence search and PCA to efficiently run texture optimization on GPUs. The GPU-based texture optimization is illustrated in Fig.2.

We firstly introduce k-coherence search to substitute hierarchical tree search in M-steps. k-coherence search[5] is fast and amenable to GPU implementation. It has been used in [16] to implement nearest neighborhood search on GPUs and greatly accelerate the texture synthesis speed.

k-coherence search consists of two phrases: precomputing and synthesis. In the precomputing phrase, for each exemplar pixel p a similarity set $C_{1..k}(p)$ of k exemplar pixels with similar $m_s \times m_s$ neighborhoods is precomputed. In the synthesis phrase, the most similar neighborhood regarding to $\boldsymbol{x}_p$ is selected from the candidate set $C(p)$ by considering $m_i \times m_i$ immediate neighbors of p. $C(p)$ could be expressed as

$$C(p) = \{\boldsymbol{u} | \boldsymbol{u} = C_i(S(p+d) - d), \quad i = 1..k, \\ d = (-m_i/2, -m_i/2)..(m_i/2, m_i/2)\}.$$

Algorithm. Parallel Texture Synthesis
$\boldsymbol{z}_p^0 \leftarrow$ random neighborhood in $Z \forall p \in X^+$
for iteration $n = 0 : N$ **do**
  $\boldsymbol{x}^{n+1} \leftarrow \arg\min_{\boldsymbol{x},\, \boldsymbol{x}(p) \in D(p)} [E_t(\boldsymbol{x}; \{\boldsymbol{z}_p^n\}) + \lambda E_c(\boldsymbol{x}; \boldsymbol{u})]$
  //E-step
  $\boldsymbol{z}_p^{n+1} \leftarrow \arg\min_{\boldsymbol{v},\, \boldsymbol{v} \in C(p)} [\|\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{v}}\|^2 + \lambda E_c(\boldsymbol{y}; \boldsymbol{u})]$
  //M-step
  if $\boldsymbol{z}_p^{n+1} = \boldsymbol{z}_p^n \forall p \in X^+$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}^{n+1}$
    break
  end if
end for

Fig.2. Pseudo-code of Parallel Texture Optimization. There are three major differences between Fig.1 and Fig.2: 1) in the E-step the pixel $\boldsymbol{x}(p)$ is restricted to the candidate set $D(p)$; 2) in the M-step the neighborhood $\boldsymbol{v}$ is restricted to the candidate set $C(p)$; 3) neighborhood similarity is approximated by $\|\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{v}}\|^2$ using PCA.

where $S(p)$ stores the source position of pixel p in the exemplar texture. From the above analysis, it could be concluded that the time complexity of k-coherence search here is $O(k \times m_i \times m_i \times m_s \times m_s)$. In our experiment, we set $k = 2$, $m_i = 7$, and $m_s = 16$ or 8 to obtain high quality textures.

Since k-coherence search requires the source position of each pixel of X (the source pixel positions are stored in S, see the expression of $C(p)$), and the original E-step does not store $S(p)$, we have to modify E-step in the following steps. 1) Firstly a candidate set $D(p)$ for pixel p is constructed from overlapping neighborhoods; 2) then the average color value is calculated from $D(p)$ and user-specified constraint to minimize the energy function $E(\boldsymbol{x})$; 3) finally the pixel in the candidate set $D(p)$ and closest to the average color value is selected as the result of the E-step: its color value is stored into X and its source position is stored into S. Steps 1 and 2 are as the original E-step while Step 3 is specifically designed for k-coherence search in an M-step.

Although with k-coherence search, the texture optimization method could run in parallel on GPUs, the evaluation of the similarity between pixel neighborhoods in the search is still time consuming. To reduce this cost, we introduce PCA for fast neighborhood comparison as in [16]. In the precomputing phrase, PCA is run on all the neighborhoods $\boldsymbol{x}_p$ and the PCA projection matrix $\boldsymbol{P}_{16}$ is calculated. Then, both $\boldsymbol{x}_p$ and $\boldsymbol{v}$ are projected to be 16-dimensional vectors $\tilde{\boldsymbol{x}}_p = \boldsymbol{P}_{16}\boldsymbol{x}_p$ and $\tilde{\boldsymbol{v}} = \boldsymbol{P}_{16}\boldsymbol{v}$, and their similarity distance could be evaluated as 16-dimensional distance $\|\tilde{\boldsymbol{x}}_p - \tilde{\boldsymbol{v}}\|$. PCA has afforded a trade off between the quality and speed for nearest neighborhood search. In

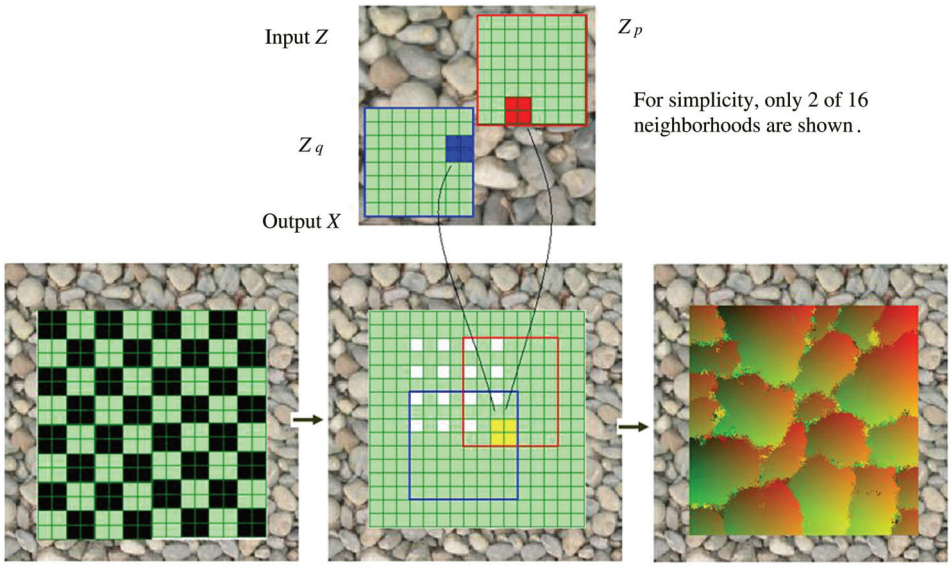


Fig.3. Local pixel coherency. In the left image, according to neighborhood placement of X^+, the output texture could be divided into blocks of $(w/4) \times (w/4)$ pixels, where w is 8. In the middle image, the 4 pixels of the yellow block are covered by the same 16 neighborhoods which are marked respectively with their top-left corner pixel in white. These 16 neighborhoods including $\boldsymbol{z}_p$ and $\boldsymbol{z}_q$ are placed $w/4$ pixels apart. In E-step, the yellow pixels are likely to be updated by pixels completely from one of these 16 neighborhoods. So after the E-step, pixels of the output texture are locally coherent, as illustrated in right image.

our experiment, high quality results could be guaranteed when the neighborhoods are compared in the projected 16-dimensional space. The synthesis speed of our basic parallel scheme is illustrated in Table 1. Compared with the original scheme for texture optimization, our basic parallel scheme has achieved much acceleration.

3.3 Accelerated Parallel Scheme

Although the basic parallel scheme proposed in the last section is much faster than the original texture optimization, it is yet not competent for interactive or real-time applications. Our observation shows that the basic parallel scheme exhibits local pixel coherency and local neighborhood stability in the synthesis process. Ignoring these two properties leads to much redundant work in the texture synthesis process. In the following two subsections, we exploit these two properties to develop two acceleration techniques for speeding up texture optimization further.

3.3.1 Local Pixel Coherency

Local pixel coherency is referred to that the space-coherent pixels covered by the same set of neighborhoods in the output texture are likely to have coherent source positions in the exemplar texture after E-step. Since neighborhoods of X^+ are set to $w/4$ pixels apart (see Subsection 3.1), every $(w/4) \times (w/4)$ pixels in the output texture are covered by same 16 neighborhoods, so the output texture could be divided into blocks of $(w/4) \times (w/4)$ pixels, as illustrated in Fig.3. Before E-step, pixels of each block may not be coherent. However, because pixels of each block are covered by a same set of neighborhoods, they are likely to be updated by pixels completely from one of the neighborhoods. As a result, pixels of each block are likely to have coherent source positions after E-step. Please see Fig.3 for the illustration of local pixel coherency.

A straightforward improvement based on local pixel coherency is to update only the top-left pixel of the $(w/4) \times (w/4)$ space-coherent pixels in E-step while keeping M-step unchanged. According to local pixel coherency, source positions of other pixels could be induced respect to the source position of top-left corner without running E-step. By this improvement the pixels updated in E-step will be reduced to only $1/((w/4) \times (w/4))$ of the original number.

In practice, we propose a more efficient technique using local pixel coherency. Firstly, according to local pixel coherency, each block of pixels could be regarded as one super pixel to update in E-step. Like the treatment in Subsection 3.2, we apply PCA to project each block to a 4-dimensional super pixel. Then in M-step, each neighborhood of $w \times w$ pixels could be changed as a super neighborhood consisting of 4×4 super pixels. In this way, the cost of M-step is also reduced because we project only 4×4 4-dimensional pixels to 16-dimensional vector now, instead of projecting $w \times w$ 3-dimensional pixels to 16-dimensional vector in the

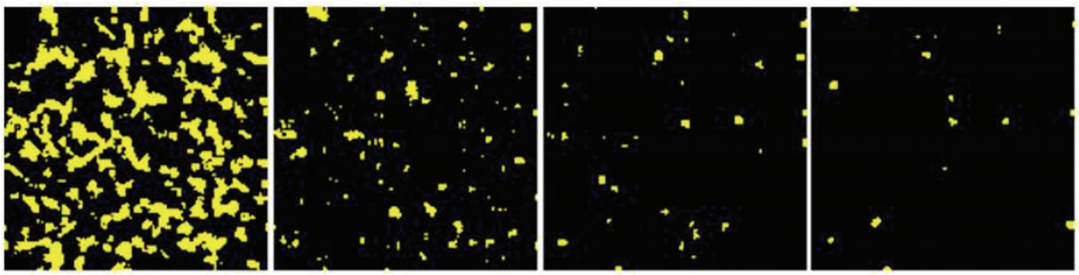


Fig.4. Local neighborhood stability. Stability maps before the 2nd, 3rd, 4th and 5th iterations are displayed with black pixels denoting stable neighborhoods and yellow pixels denoting unstable neighborhoods. Most of the neighborhoods become stable after 1∼2 iterations. Here, the threshold is 0.1.

basic parallel scheme.

3.3.2 Local Neighborhood Stability

Local neighborhood stability is referred to that most output neighborhoods $\{\boldsymbol{z}_p\}$ may reach stable status after few iterations in the synthesis process. We will utilize local neighborhood stability to reduce the cost of M-step. Local neighborhood stability is an inherent property of the EM-like synthesis algorithm. As reported in [15], the EM-like algorithm need to perform 3∼5 iterations at each synthesis level (in a multi-level synthesis framework) to achieve a high quality result. In fact, we found that most output neighborhoods $\{\boldsymbol{z}_p\}$ reach stable status after 1∼2 iterations, as illustrated in Fig.4. So it is not necessary to perform 4∼5 iterations of E-step and M-step for all neighborhoods. Considering this, an adaptive strategy by the stability of neighborhoods is used for performing E and M steps.

In this paper, the stability of single neighborhood centered at p is measured as $\|\boldsymbol{z}_p - \boldsymbol{z}'_p\|^2$, where $\boldsymbol{z}'_p$ is the result of the last iteration. Before M-step, the stability of each neighborhood is calculated. When the stability of a neighborhood centered at pixel p is below a set threshold (like 0.1 in our experiments), we let it inherit its old $\boldsymbol{z}_p$ instead of searching for a new $\boldsymbol{z}_p$. In our experiments this adaptive strategy may reduce nearly half of the M-step cost.

3.4 Implementation Details

In this subsection, we provide some implementation details to complement the above discussion.

Multi-Level Synthesis. As reported in [15], multi-level synthesis is combined with the basic texture optimization algorithm to capture large scale texture structures with relative small neighborhood sizes. We also perform multi-level synthesis and generally use 3 or 4 resolution levels and successive neighborhood sizes of 16×16 and 8×8 pixels at each resolution.

Precomputation. In our algorithm, two kinds of precomputed data are required: 1) for k-coherence search, the similarity set $C_{1..k}(p)$ is precomputed for each neighborhood, as done in [5]; 2) for fast neighborhood comparison and using local pixel coherency, the neighborhood PCA projection matrixes with related projected neighborhoods are precomputed like [16]. All these precomputed data are stored as 2D textures and preloaded to GPU before the synthesis algorithm works.

GPU Implementation. Similar to [16], the basic parallel scheme could be divided and implemented in three fragment programs: E-step program, M-step program and upsampling program. The input of E-step program is neighborhood texture Y (storing $\{\boldsymbol{z}_p\}$) and related precomputed data, and its output is pixel texture X (storing $\boldsymbol{x}$). The input of M-step program is X and related precomputed data, and its output is Y. Upsampling program is needed when applying multi-level synthesis. Either X or Y could be up-sampled, and we found upsampling Y is helpful for keeping the stability of neighborhoods.

In the accelerated parallel scheme, we avoid redundant M-steps for stable neighborhoods by branching. However, the current GPU pixel shader does not support efficient branching on per pixel granularity, so we use early-Z culling to terminate M-steps for the stable neighborhoods. Thus two fragment programs are added: a backup program and a stable program. The backup program stores current $\boldsymbol{z}_p$ for next comparison, and the stable program sets the depth information according to the neighborhood stability and enables early-Z culling for M-steps.

All fragment programs are carefully designed so that there is no need to transfer data between CPU and GPU except loading precomputed data into GPU in the beginning.

4 Interactive Flow-Guided Synthesis

As our synthesis scheme is fast enough, it can be applied to interactive editing for the flow-guided synthesis. We follow [19] to define several user-specified constraints. Each user-specified constraint represents a kind of vector field, and they could be accumulated

to form a new combinatorial vector field. Users can interactively add, delete or modify these constraints to guide the flow-guided synthesis.

For instance, a constraint corresponding to an isotropic source at location $\boldsymbol{p}_0 = (x_0, y_0)$ with strength $k > 0$ is defined as

$$\boldsymbol{V}(\boldsymbol{p}) = \begin{pmatrix} k & 0 \\ 0 & k \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \Big/ ((x - x_0)^2 + (y - y_0)^2).$$

Other isotropic singular constraints include a sink, a saddle, a counter-clockwise center, and a clockwise center, whose matrices are as follows:

$$\begin{pmatrix} -k & 0 \\ 0 & -k \end{pmatrix}, \begin{pmatrix} k & 0 \\ 0 & -k \end{pmatrix}, \begin{pmatrix} 0 & -k \\ k & 0 \end{pmatrix}, \begin{pmatrix} 0 & k \\ -k & 0 \end{pmatrix}.$$

A regular constraint at location $\boldsymbol{p}_0$ with vector value $\boldsymbol{V}_0$ is define as

$$\boldsymbol{V}(\boldsymbol{p}) = ((x - x_0)^2 + (y - y_0)^2)\boldsymbol{V}_0.$$

After a new vector field is decided by users, the warp-correct scheme[15] is adopted for flow-guided synthesis. The new texture flow could be synthesized at interactive speeds by our accelerated parallel synthesis scheme, demonstrated in appended videos.

5 Results and Discussion

We made experiments to test our new scheme for texture optimization, in comparison with the original scheme for texture optimization[15] and the parallel controllable texture synthesis method[16]. Some results of unconstrained texture synthesis are illustrated in Fig.6, and some results of flow-guided synthesis are illustrated in appended videos.

Table 1. Synthesis Times for Compared Synthesis Schemes

Target	Texture Synthesis ($64^2 \to 256^2$)	Flow-Guided Synthesis ($64^2 \to 256^2$)
Original Texture Optimization	420∼600s	20∼60s
Discrete Texture Optimization	$>$ 3.6s	$>$ 1.2s
Basic Parallel Scheme	170∼190ms	140∼150ms
Accelerate Parallel Scheme	77∼ 87ms	55∼65ms

Besides producing high quality textures as the original scheme for texture optimization[15], our accelerated parallel scheme could run at interactive speeds, which is 4000+ times faster than the original scheme. In our experiments, precomputation for PCA takes less than 1 second on a 64^2 or 128^2 exemplar texture, and precomputation for k-coherence search takes about 15∼20 seconds on a 64^2 exemplar texture and 1∼2 minutes on a 128^2 exemplar texture. Various schemes for texture optimization are compared in Table 1 by the synthesis time, including original texture optimization[15], discrete texture optimization[17], our basic parallel scheme and accelerated parallel scheme. Time of our algorithms was measured on a personnel computer installed with a Pentium 4 3.2GHz CPU and an NVIDIA Geforce 7900 GT GPU. Time of the original texture optimization is extracted from their paper. As for the time of discrete texture optimization in [17], it is briefly and reasonably estimated from the results of the paper on surface texture synthesis.

Although our method is a little slower than [16], it is better to handle the textures with semantic structures by utilizing neighborhoods of relatively large sizes. As shown in the results in Fig.5, our scheme can better reproduce various features of exemplar textures such as the consistent feature of curves, the shape feature of red balls and the uniform distribution of knots in the upper, middle and below exemplar textures respectively.

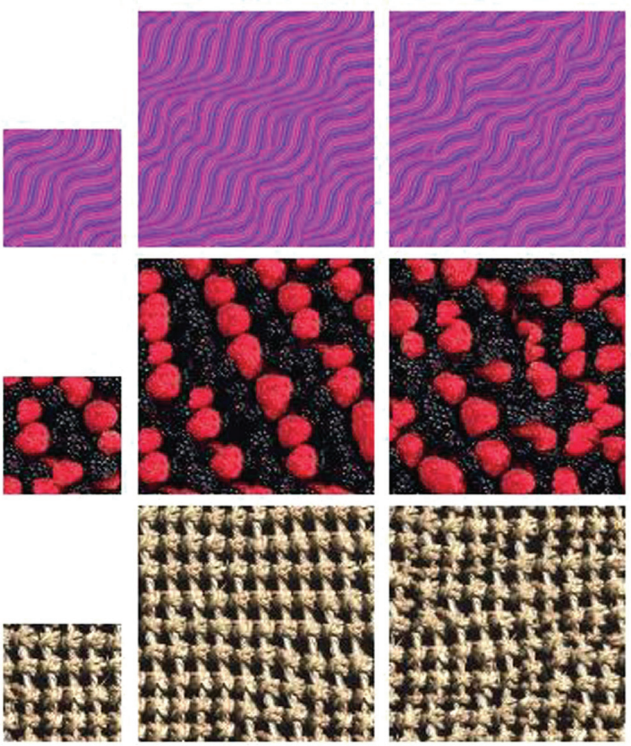


Fig.5. Synthesis quality comparison. For each group of images, input is on the left, our result is in the middle, and the result of [16] is on the right.

Results of unconstrained texture synthesis (Fig.6 and Table 1) show that our accelerated parallel scheme is able to produce high quality texture in high speeds. And it also inherits the controllability and flexibility of texture optimization, which is illustrated by the

Fig.6. Results by our new scheme. For each group of images, input is on the left and output is on the right. Please refer to accompanied videos for texture animation. (Please contact authors if you need the videos.)

interactive flow-guided synthesis in the accompanied videos.

As discussed above, k-coherence search and PCA together could afford a good trade-off between quality and speed, but broken of feature lines and periodic phenomena could still be observable in some of our results. It might be due to following reasons: 1) texture optimization adopts an EM-like algorithm. It is well-known that EM algorithm itself is a local optimization algorithm, which is much dependent on initial value and could not reach the global minimum in general, so local minimum is unavoidable and the broken of feature lines might appear; 2) neighborhood similarity is defined by L^2 distance of neighborhood colors, which has not explicitly considered the feature lines of the exemplar texture, so broken feature lines may be produced in synthesized texture; 3) no mechanism is explicitly considered to control the randomness of synthesized result, so in some results periodic phenomena might appear.

We might try to incorporate semantic information into our neighborhood similarity to avoid the broken of feature line, and design randomness-controlling mechanism to avoid periodic phenomena in the future.

6 Summary and Future Work

A fast parallel scheme is proposed for texture optimization. By using k-coherence search for finding nearest neighborhoods, the inherent synthesis parallelism of texture optimization is successfully exploited to implement parallel synthesis on GPUs. Meanwhile, much acceleration is obtained by taking full advantages of local pixel coherency and local neighborhood stability in the texture optimization process. Our new scheme is capable of interactive applications like interactive editing for flow-guided synthesis. There are several directions to extend our scheme, such as applying it to other interactive applications or anisometric synthesis. We will study these in the near future.

References

[1] Efros A, Leung T. Texture synthesis by non-parametric sampling. In *Proc. International Conference on Computer Vision*, Corfu, Greece, 1999, pp.1033∼1038.

[2] Wei L Y, Levoy M. Fast texture synthesis using tree-structured vector quantization. In *Proc. ACM SIGGRAPH 2000*, New Orleans, Louisiana, USA, 2000, pp.479∼488.

[3] Ashikhmin M. Synthesizing natural textures. In *Proc. ACM Symp. Interactive 3D Graphics*, Chapel Hill, NC, USA, 2001, pp.217∼226.

[4] Hertzmann A, Jacobs C E, Oliver N *et al.* Image analogies. In *Proc. SIGGRAPH*, Los Angeles, California, USA, 2001, pp.327∼340.

[5] Tong X, Zhang J, Liu L *et al.* Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proc. SIGGRAPH 2002*, San Antonio, Texas, USA, 2002, pp.665∼672.

[6] Zhang J, Zhou K, Velho L, Guo B, Shum H Y. Synthesis of progressively-variant textures on arbitrary surfaces. In *Proc. SIGGRAPH 2003*, San Diego, California, 2003, pp.295∼302.

[7] Efros A A, Freeman W T. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH 2001*, Los Angeles, California, USA, 2001, pp.341∼346.

[8] Liang L, Liu C, Xu Y Q, Guo B, Shum H Y. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 2001, 20(3): 127∼150.

[9] Cohen M F, Shade J, Hiller S, Deussen O. Wang tiles for image and texture generation. In *Proc. SIGGRAPH 2003*, San Diego, California, 2003, pp.287∼294.

[10] Kwatra V, Schödl A, Essa I, Turk G, Bobick A. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphic*, 2003, 22(3): 277∼286.

[11] Wu Q, Yu Y. Feature matching and deformation for texture synthesis. In *Proc. ACM SIGGRAPH*, Los Angeles, California, 2004, pp.364∼367.

[12] Heeger D J, Bergen J R. Pyramid-based texture analysis/synthesis. In *Proc. ACM SIGGRAPH*, Los Angeles, CA, USA, 1995, pp.229∼238.

[13] Debonet J S. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proc. ACM SIGGRAPH*, Los Angeles, CA, USA, 1997, pp.361∼368.

[14] Wei L Y, Levoy M. Order-independent texture synthesis. Tech. Rep. TR-2002-01, Stanford University CS Department, 2002.

[15] Kwatra V, Essa I, Bobick A, Kwatra N. Texture optimization for example-based synthesis. *ACM Transactions on Graphic*, 2005, 24(3): 795∼802.

[16] Lefebvre S, Hoppe H. Parallel controllable texture synthesis. *ACM Transactions on Graphic*, 2005, 24(3): 777∼786.

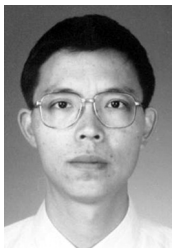
[17] Han J, Zhou K, Wei L, Gong M, Bao H, Zhang X, Guo B. Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer*, 2006, 22(9): 918∼925.

[18] Lefebvre S, Hoppe H. Appearance-space texture synthesis. In *Proc. SIGGRAPH*, Boston, Massachusetts, USA, 2006, pp.541∼548.

[19] Zhang E, Mischaikow K, Turk G. Vector field design on surfaces. Tech. Rep. 04-16, Georgia Institute of Technology, 2004.

Hao-Da Huang received his B.S. degree in computer science from University of Science and Technology of China in 2004. Currently he is a master candidate of Institute of Software, Chinese Academy of Sciences. His research interests include texture synthesis and realistic rendering.

Xin Tong is a researcher/project leader of Internet graphics group, Microsoft Research Asia. After receiving his Ph.D. degree in computer graphics from Tsinghua University, Beijing in July 1999, he joined Microsoft Research China as an associate researcher. Before that, he received his B.S. degree and Master's degree in computer science from Zhejiang University in 1993 and 1996, respectively. His research interests include appearance modelling and rendering, image-based rendering, texture synthesis and natural phenomena simulation.



Wen-Cheng Wang is a research professor in Institute of Software, Chinese Academy of Sciences. He received the B.S. degree in computer science from Xiangtan University in 1988, Master's degree and Ph.D. degree in computer science from Institute of Software, Chinese Academy of Sciences in 1993 and 1998 respectively. His current research interests include visualization/volume rendering, image-based rendering, texture synthesis and virtual reality.