

An Adaptive Texture Synthesis Algorithm

Xuewen Ding^{1, a}, Subramaniam Ganesan^{2, b}, Jing Chen^{3, c}

¹School of Electronic Information Engineering, Tianjin University of Technology and Education, Tianjin, China, 300222

²Department of Electrical and Computer Engineering, Oakland University, Rochester, Michigan, USA, 48309-4478

³Tianjin Tianda Qiushi Electirc Power High Technology Co., Ltd., Tianjin, China, 300384

^adingxw1@126.com, ^bganesan@oakland.edu, ^cchenjing7777@163.com

Keywords: Texture Synthesis, Adaptive texture synthesis, global structure, low computational cost

Abstract. Patch-based texture synthesis has proven to produce reasonable results for a wide variety of texture classes. In this paper, an effective adaptive texture synthesis method is proposed based on the popular Image Quilting and the general framework of Hybrid Texture Synthesis (HTS) algorithm. Our algorithm uses the Fourier domain for finding the best match candidate patches, and adaptively splits them so as to use as large as possible patches to preserve global structure within the input texture. Finally, the newly chosen patch is adaptively composited into the synthesized result to remove seams and discontinuities in the overlap region. Results from our implementation show that our algorithm produces high-quality texture at low computational cost. Our algorithm is also flexible to obtain the trade-off between visible quality and speed.

Introduction

Texture synthesis from example is the process that an input texture chip is taken and used as a basis to generate an arbitrary quantity of 'similar' texture. Copying parts (pixels or patches) from the input and pasting them together as an output image is one way texture synthesis can be accomplished. In this paper, an adaptive texture synthesis method base on the popular Image Quilting algorithm [3] and this general framework of Hybrid Texture Synthesis is proposed [1][2]. The proposed algorithm improves Image Quilting and HTS in several aspects to overcome some of their shortcomings.

Previous work

Texture synthesis approaches most relevant to ours can be classified into two major braches of research with regards to non-parametric synthesis from example.

One set of methods are pixel-based, where a texture is synthesized by repeatedly matching the neighborhood around the target pixel in the synthesis result with the input texture [4][5]. The main problem with pixel-based algorithms is speed owing to initially an exhaustive process. These algorithms also tend to lose global features within the input texture for using only a small neighborhood round single pixel.

Patch-based synthesis algorithms copy whole patches using an overlap region from the source texture and paste them into the synthesized result [1][3][6][7]. If the chosen patch size is large enough to encompass the global features, implementations of these methods tend to preserve features in the image. While solving the issues with the lack of speed common to pixel-based methods, traditional Patch-based texture synthesis algorithms introduced their own set of problems like discontinuities along the seams of the patches.

Some methods which can remove the seams on the boundaries of patches have been implemented [1][2][3]. Efros and Freeman's Image Quilting algorithm aligns adjacent patch boundaries, constrained by overlap, and then performs a minimum-error-boundary-cut (MEBC) within the overlap region to reduce overlap artifacts [1]. In so doing, the MEBC helps to preserve localized high-frequency feature such as edges within the source texture. HTS method proposed by Nealen and

Alexa uses patch sampling for initial synthesis, and re-synthesizing individual pixels in a carefully calculated sequence by a post-process procedure[2][3]. Despite its simplicity, Image Quilting works well, producing synthesis results that are equal or better than the Efros & Leung family of algorithms however at a fraction of the computational cost [1]. But Image Quilting still has a tendency to generate abrupt color changes, termed boundary mismatch [7]. The HTS method does a good job of eliminating overlap artifacts between adjacent patches, but it suffer from a heavy computational expense for the overlap re-synthesis stages [2][3].

Adaptive Texture Synthesis Algorithm

The complete stages of the texture synthesis Algorithm proposed in this paper are as follows:

- 1) Go through the image to be synthesized in raster scan order in steps of one patch.
- 2) For every location, search the input texture to find the best patch, constrained by overlap with the existing synthesis result.
- 3) Split the patch adaptively so as to use as large as possible patches while staying within a user-defined error tolerance Δ_{max} ($\Delta_{max} \in [0,1]$) for the mismatch in the overlap region.
- 4) Paste the block onto the synthesized texture by using Image Quilting or HTS. The overlap error is used to make choice. If overlap error of the newly chosen block is bigger than k ($k \in [0,1]$) times Δ_{max} , HTS with per pixel re-synthesis in the overlap is used. Otherwise Image Quilting is used to stitch together the new patch and the synthesized result. Repeat.

In all of our experiments the width of the overlap edge (on one side) was 1/5 of the size of the block. Below describes in detail Steps from 2) to 4).

Best patch Searching

In algorithm step i , a patch is selected from an example texture T which best fits the target region in the existing result. This selection/search procedure is constrained by the overlap error between the newly chosen block and the intermediate result.

The error image is computed by the use of the image mask I_i and binary support function J_i (Fig. 1). Firstly, the current block is simply grown by the pixel overlap, and then the grown region in the current result is checked to search the already synthesized pixels: if it is a valid pixel in the current location, the corresponding pixel of J_i (initially all 0's) is set to 1 and the color value from the synthesized result is stored in I_i .

Given the input texture T , after obtaining I_i and J_i , the weighted error $\Delta_i \in [0,1]$ is computed between the mask I_i and a circular shift \mathbf{x}_0 of T as

$$E_i(\mathbf{x}_0) = \frac{1}{\kappa_i} \sum_{\mathbf{x}} \sum_c \left[J_i(\mathbf{x}) W_c (I_{i,c}(\mathbf{x}) - T_c(\mathbf{x} + \mathbf{x}_0))^2 \right] \quad (1)$$

The error image E_i stores Δ_i for each \mathbf{x}_0 . Where $\kappa_i = \sum_{\mathbf{x}} J_i(\mathbf{x})$, $c = \{R, G, B\}$ (the set of color channels in RGB space) and $\sum_c W_c = 1$. Obviously $J_i(\mathbf{x}) I_i(\mathbf{x}) = I_i(\mathbf{x})$, and define the cross correlation between two images (functions) $f \diamond g$ as $(f \diamond g)(\mathbf{x}_0) = \sum_{\mathbf{x}} f(\mathbf{x}) g(\mathbf{x} + \mathbf{x}_0)$, Equation 1 is computed as

$$E_i(\mathbf{x}_0) = \frac{1}{\kappa_i} \sum_c W_c \left[\sum_{\mathbf{x}} I_{i,c}(\mathbf{x})^2 - 2(I_{i,c} \diamond T_c)(\mathbf{x}_0) + (J_i \diamond (T_c^2))(\mathbf{x}_0) \right] \quad (2)$$

The correlation $f \diamond g$ between two functions can be computed in fourier space as $f \diamond g = F^{-1}(F(f) * F(g))$. In the implementation, the fourier transform for T is pre-computed and only the fourier transforms of I_i and J_i need be recomputed for each new block.

Based on the greater sensitivity of the human visual system to changes in luminance than changes in hue or saturation, $W_{R,G,B} = \{0.299, 0.587, 0.114\}$, which is analogous to the Y component (luminance) of the YIQ color model [8].



Adaptive Patch Sampling

After P_i chosen from the input texture T in algorithm step i , if the weighted error Δ_i between the overlapping pixels of P_i and the already synthesized result exceeds a user-defined error tolerance Δ_{max} , the current patch P_i will be subdivided into four small congruent blocks and recurse. In our implementation, the patch P_i initially is quadrilateral patch with size 2^n by 2^m ($n, m \in \mathbb{N}$). The adaptive sampling split is taken as example shown in Fig.2. For recursive calls, the pixel overlap is halved at every split and limited to a minimal overlap width value of 3.

The value of the user-defined error tolerance Δ_{max} determines directly the amount of block splits. If setting Δ_{max} to 0 leads to always split, it is equal to a traditional per-pixel synthesis method. Whereas, setting Δ_{max} to 1 will not split, it is like the typical patch-based methods. Less extreme values used for Δ_{max} allows the synthesis trades structural inconsistencies for detail artifacts. In our experiment, Δ_{max} usually is set value with $\Delta_{max} \leq 0.1$. The global structure preserving and artifacts removing will be obtained by the following block compositing stage.

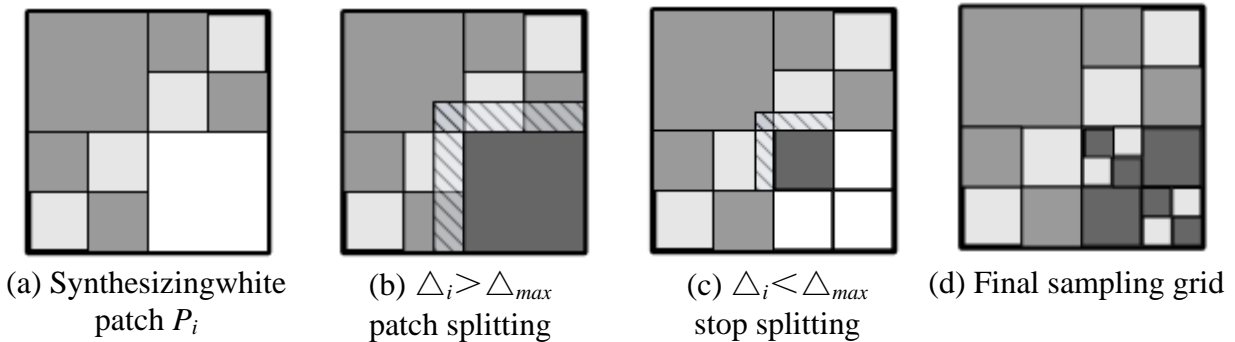


Fig.2. Adaptive patch sampling. Shades of gray represent already synthesized patches, the hatched areas are the overlap regions used for best patch search.

Block Compositing

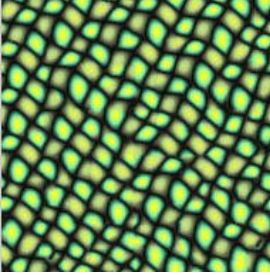
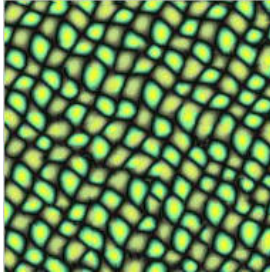
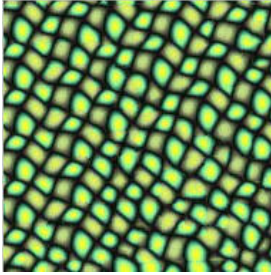
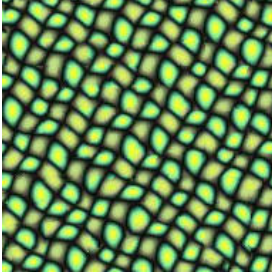
After a patch is picked from T , the newly chosen patch will be adaptively composited into the synthesized result by using the compositing scheme in Image Quilting or HTS. If the overlap error of the current block is smaller than k ($k \in [0, 1]$) times Δ_{max} , Image Quilting is used for compositing. The minimum cost path along the overlap region will be computed for stitching directly together the new patch and the synthesized result.

When the overlap error of the newly chosen block is bigger than k times Δ_{max} , there has a big tendency to produce boundary mismatch if Image quilting is still used for compositing. So the overlap re-synthesis strategy in HTS will be applied. In the overlap regions, a pixel error is firstly computed for each pixel and the pixel with an error exceeding a user defined pixel error tolerance δ_{max} will be marked as invalid. To ensure sufficient valid neighborhoods for these mismatched pixels, a traversal order for them is calculated by using morphological dilation of the valid regions. Then each invalid pixel is re-synthesized individually in the order given by the pixel traversal map. Finally, the newly

chosen block with re-synthesized overlap is copied to the target region in the already synthesized result.

The number k allows a trade-off between quality and speed. The synthesis quality for various settings of k is shown in Table 1. Bigger values for k lead to faster synthesis at the potential cost of visual quality. Note that setting $k=1$ results in an approach which can be called the adaptive Image Quilting. The adaptive Image Quilting can get better visual effect than ordinary Image Quilting, but the boundary mismatching can't vanish. Whereas setting $k=0$ results in entire HTS.

Table 1. Four 192×192 synthesis results on an i5-4210M with varying values for k , using Fig.1 (a) *green scale* 64×64 as input (Initial patch size of 32×32 , $\delta_{max} = 0.02$ and $\Delta_{max} = 0.04$).

$k=0.2$	$k=0.4$	$k=0.6$	$k=0.8$
			
Synth time:65 sec.	Synth time:61sec.	Synth time: 46 sec.	Synth time:25 sec.


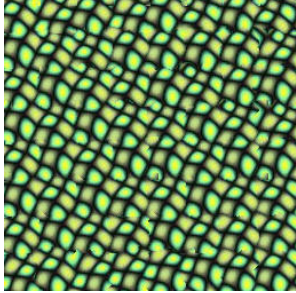
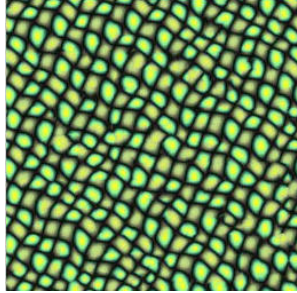
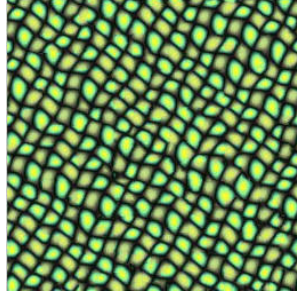

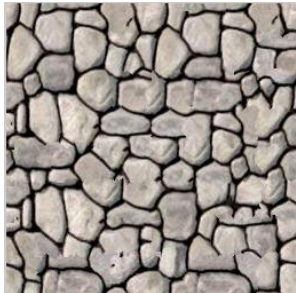


Experimental Results

The algorithm is compared to the original Image Quilting and HTS algorithm using the images provided in [1][2][3]. We were especially interested in comparing the synthesis results of Wei/Levoy *green scales* and Liang et al.'s natural *stone wall* (Table 2). Both of them display a great amount of structure and anisotropy, which generally challenge existing synthesis algorithms. Especially the benchmark *green scales* texture which is compared in most other papers as well. In addition, we also used other complex textures that are a little hard to synthesize with some existing algorithms.

Table 2 shows some synthesis results/timings using *green scales* texture and natural *stone wall* textures based on our Matlab implementation. The results in each row are of size 256×256 with varying input texture sizes from 64×64 to 200×200 . In column 2 of Table 1, we see the results generated by Image Quilting. Although these results preserve global structure and similarity, the blurring and boundary mismatch artifacts are noticeable, thus revealing the regular patch grid. The biggest advantage of Image Quilting is simplicity and fast-speed, with the un-optimized MATLAB code used to generate these results running for less than 10 seconds. Column 3 demonstrates that HTS usually obtain the compelling results but the longest synthesis time because of the cost of additional computational effort, especially when many pixels must be re-synthesized. We produced the column 4 by using the adaptive algorithm proposed in this paper and setting k equal to a medium value 0.5. Although our adaptive algorithm does not entirely nullify the existence of overlap artifacts, it can reduce their overall frequency like HTS, thereby eliminating visible patch boundaries. We found the results of our algorithm are very close to the production of HTS, but even preserve the global structure better than HTS in the result of *stone wall* texture because of using Image Quilting for some patch synthesis. The consuming time of our adaptive algorithm is much less than performing of HTS for both of these two textures and it will reduce continuously when increasing the value of k . The results using HTS and our adaptive algorithm are generated by setting Δ_{max} to the same value, with $\Delta_{max} = 0.04$ for *green scales* and $\Delta_{max} = 0.03$ for *stone wall* texture.

More comparing results are shown in Table 3, with their respective parameters setting. Our algorithm generally produces synthesis results close to HTS, however, better than Image Quilting. Its synthesis time always falls in between the cost Image Quilting and HTS.

Table 2. Results and times on an i5-4210M using *green scales* and *stone wall* texture. Each result is 256×256(Initial patch size of 32×32).

input	Image Quilting	HTS	Our Algorithm $k=0.5$
 <p><i>Green scales</i> 64×64 $\delta_{max} = 0.02$ $\Delta_{max} = 0.04$</p>	 <p>Synth time:5 sec.</p>	 <p>Synth time:179 sec.</p>	 <p>Synth time:125 sec.</p>
 <p><i>Stone wall</i> 200×200 $\delta_{max} = 0.02$ $\Delta_{max} = 0.03$</p>	 <p>Synth time:7sec.</p>	 <p>Synth time:81 sec.</p>	 <p>Synth time:52 sec.</p>

Conclusions

In this paper we proposed an adaptive texture synthesis method based on Image Quilting and HTS. Our adaptive texture synthesis works for any block layout or shape. The results of our algorithm are of better quality comparing to Image Quilting, for limiting the overlap error by splitting adaptively the blocks and searching the best pixel to replace invalid pixels in the overlap region. It is significantly faster than HTS for stitching directly the new block to the synthesized result and has comparable synthesis quality as HTS. Our adaptive texture synthesis is also flexible to obtain the trade-off between visible quality and speed by adjusting the parameter k as mentioned in Section 2.

Acknowledgements

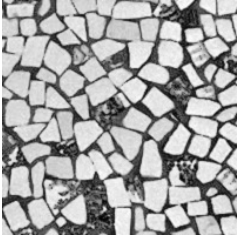
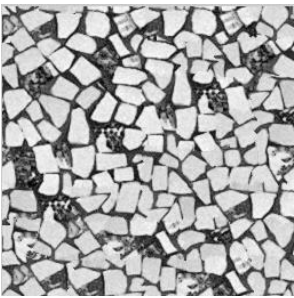
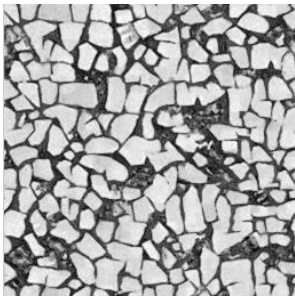
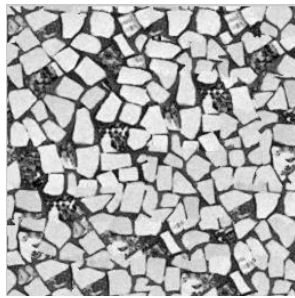


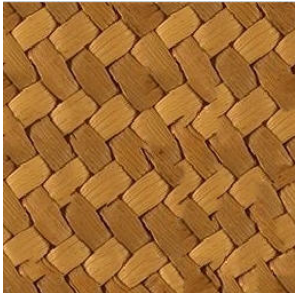

We would like to thank Andrew Nealen and Marc Alexa for making their HTS code freely available for research purposes. This framework provided an excellent background for the rapid prototyping of our method.

This research work is supported by the National Natural Science Foundation of China (No. 61271412), by Tianjin Science and Technology plan project (No.14ZXCXGX00594), by Tianjin City High School Science & Technology Fund Planning Project (No.20130711 and No.20110710), by the Foundation of Tianjin University of Technology and Education (No.RC14-46 and No.KJY12-04).

References

- [1] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In Proceedings of the 28th annual conference on Computer Graphics and Interactive techniques, 2001. 341-346.
- [2] Andrew Nealen and Marc Alexa. Fast and high quality overlap repair for patch-based texture synthesis. In CGI '04: Proceedings of the Computer Graphics International (CGI'04), 2004. 582-585.
- [3] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In Proceedings of the 14th Eurographics workshop on Rendering, 2003. 97-105.
- [4] Seunghyup Shin, Tomoyuki Nishita and Sung Yong Shin. On pixel-based texture synthesis by non-parametric sampling. Computers & Graphics, 2006:30 (4):767-778.
- [5] Steve Zelinka and Michael Garland. Jump map-based interactive texture synthesis. ACM Trans. Graph., 2004:23(4):930-962.
- [6] Krzysztof Ślot, Łukasz Kornatowski and Piotr Dębiec. Fast texture synthesis with cellular neural network-based patch stitching. International Journal of Circuit Theory and Applications, 2012:40(12): 1265-1275.
- [7] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. In ACM Transactions on Graphics (TOG), 2001 (20) 127-150.
- [8] J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes Computer Graphics. Principles and Practice. Addison Wesley, 1990.

Table 3. Results and times on an i5-4210M using *Mosaic* and *Mats* texture. Each result is 256×256(Initial patch size of 32×32).

input	Image Quilting	HTS	Our Algorithm $k=0.5$
 <p><i>Mosaic</i> 200×200 $\delta_{max} = 0.02$ $\Delta_{max} = 0.02$</p>	 <p>Synth time:7 sec.</p>	 <p>Synth time:172 sec.</p>	 <p>Synth time:145 sec.</p>
 <p><i>Mats</i> 200×200 $\delta_{max} = 0.02$ $\Delta_{max} = 0.03$</p>	 <p>Synth time:7sec.</p>	 <p>Synth time:35 sec.</p>	 <p>Synth time:27 sec.</p>