

Fast and High Quality Overlap Repair for Patch-Based Texture Synthesis

Andrew Nealen Marc Alexa

Department of Computer Science
Darmstadt University of Technology
{nealen,alex}@informatik.tu-darmstadt.de

Abstract

Patch-based texture synthesis has proven to produce high quality textures faster than pixel-based approaches. Previous algorithms differ in how the regions of overlap between neighboring patches are treated. We present an approach that produces higher quality overlap regions than simple blending of patches or computing good boundaries, however, that is faster than re-synthesizing invalid pixels using a classical per-pixel synthesis algorithm: we use a k -nearest neighbor (knn) data structure, obtained from the input texture in a precomputation step. Results from our implementation show that the algorithm produces high-quality textures, where the time complexity of the synthesis stage is linear in the number of re-synthesized pixels and, therefore, scales well with the size of the input texture.

Keywords: *Patch-based Texture Synthesis, Principal Component Analysis, k -coherence search*

1. Introduction

The field of texture synthesis is concerned with synthesizing, from an input texture sample, an arbitrary amount of *perceptually similar* output texture in 2D image space or on surfaces of 3D models. The term *perceptually similar* in this context means that the user should recognize the result as the *same* texture, yet it must also contain sufficient variation of the input so it is not perceived as *identical*.

Patch-based algorithms tend to preserve the global structure very nicely, however, sometimes introduce local artifacts in the regions of overlap between neighboring patches. The approaches differ in their strategy of generating good regions of overlap. Some of the algorithms [3, 6, 5, 2] use only the pixels coming from the overlapping patches, either blending them or finding a good (non-trivial) boundary between the patches. This leads to potentially fast algorithms that, in some cases, produce visible artifacts. These can be reduced by adaptively varying the patch sizes [8, 7]. A more

flexible method is to explicitly repair bad pixels in the overlap region. This could be done in the spirit of Markov Random Field per-pixel synthesis by exhaustively searching the input texture for similar neighboring valid pixels [4, 10, 7]. Accelerating per-pixel synthesis generally involves intelligently reducing search space [1, 9] or precomputation [11].

In this paper, we introduce a fast, effective and controllable algorithm, which can entirely replace the overlap re-synthesis stage described in our previous paper [7]. Therein, the overlap re-synthesis is of complexity $O(rN \log N)$, (with N = number of pixels in the input texture and r = number of re-synthesized pixels) due to the repeated Fourier Transform of the neighborhood mask, which must be of size $N \times N$ as it is convolved with the entire input (see [7, 8] for details). This obviously does not scale well for large input images with many re-synthesized output pixels (= large values for N and r). We therefore propose an alternative algorithm, inspired by k -coherence search [9], which is linear in the number of re-synthesized pixels.

2. Patch-Based Texture Synthesis with Per-Pixel Repair

We briefly explain the basic idea of patch-based texture synthesis with per-pixel re-synthesis of invalid pixels in the overlap regions. This is essentially a non-adaptive version of Hybrid Texture Synthesis [7]. Note that the ideas we present later apply to all patch sizes or shapes.

Given an input texture and the goal to generate an arbitrarily sized output texture patch-by-patch, the algorithm is informally described as follows:

- (1) For each output texture patch
 - (a) Find the best patch in the input texture, constrained by overlap with the existing synthesis result.
 - (b) Compute the per-pixel error in the overlap region (i.e. the difference between pixels com-

ing from the neighboring patches); mark each pixel exceeding a user defined pixel error tolerance (δ_{max} in $[0,1]$) as invalid.

- (c) Compute a traversal order for these invalid pixels (pixel traversal map).
- (d) Finally, re-synthesize each invalid pixel individually in the order given by the pixel traversal map.

The main benefit of this algorithm is the degree of flexibility in the generation of overlap regions, as in some cases, the pixels coming from neighboring patches are insufficient to generate a visually pleasing overlap region. This goal has been achieved at the cost of additional computational effort, especially when many pixels must be re-synthesized. In this paper, we therefore concentrate on the acceleration of step (d). For details on the other steps, see other work on patch-based texture synthesis [3, 6, 7].

In an exhaustive search scheme for re-synthesizing a single pixel, the entire input texture is compared to the pattern of valid synthesized pixels within a box-shaped neighborhood of the target pixel, employing the same L_2 error metric used for finding best patches. Naive L_2 comparison of two images I_1 and I_2 , each with N pixels, for each of the N possible circular shifts of I_1 (or I_2), results in $O(N^2)$ complexity. This can be reduced to $O(N \log N)$ by elegant reformulation in Fourier Space [8]. Still, performing a few Fourier Transforms for each re-synthesized pixel simply does not scale well with the size of the input texture. Unfortunately, we cannot adopt many of the existing acceleration schemes, as we are confronted with a non-fixed (e.g. non-L-shaped [10]) neighborhood of valid pixels. We have therefore decided to implement a variant of k -coherence search, which we describe in the following section.

3. Acceleration

The basic principle is to restrict the search space of possibly fitting neighborhoods for each re-synthesized pixel. Ashikhmin [1] notes, that in the case of natural textures it is sufficient to inspect only a reduced set of candidate pixels C_a in the input texture. This set is constructed by gathering the pixels in the (e.g. 3×3) neighborhood around the target pixel (the one dark gray and two light gray pixels in the left image of Fig. 1), locating their original positions in the input texture (Fig. 1, upper right) and collecting the properly shifted pixels, which constitute the set C_a (the two hatched pixels in the upper right image of Fig. 1). For this it is necessary to store the original location (in the input texture) of each output pixel. We name this two-dimensional array (with dimensions identical to the output texture) the *source map*. Note that in Ashikhmin’s paper this data structure is termed the *array of original positions* [1]. Then,

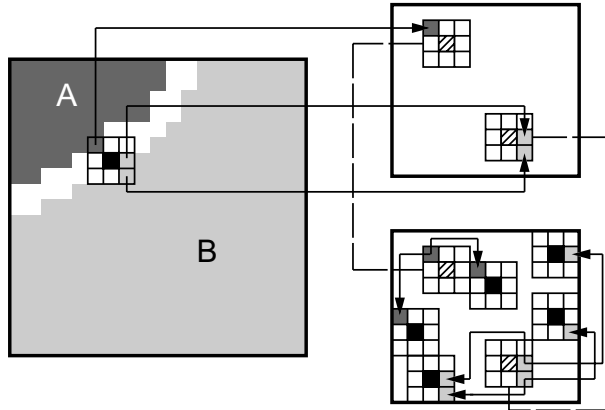


Figure 1. Left: The intermediate synthesis result. Upper right: the input texture and C_a . Bottom right: the input texture and C_{ext} .

when re-synthesizing a single pixel (see Fig. 1, left image), one must only check the neighborhoods of the candidate set C_a against the target neighborhood in the output texture.

In the setting of overlapping patches, this approach tends to produce unsatisfactory results. Pixels in the overlap regions generally come from only two or three patches. Using neighboring pixels from the source texture can, therefore, only extend the existing contiguous pixel-patches to eventually meet along a path in the patch overlap region, similar to Image Quilting (IQ) [3]. The example in Fig. 1 shows this phenomenon: the two hatched pixels in the candidate set C_a are both simply extensions to the already existing, contiguous patches A and B (Fig. 1, left).

Fig. 2 compares this technique with the results of an exhaustive search. Generally, the region growing nature of Ashikhmin’s algorithm is favorable for per-pixel synthesis, but might produce inferior results when combined with patch-based approaches.

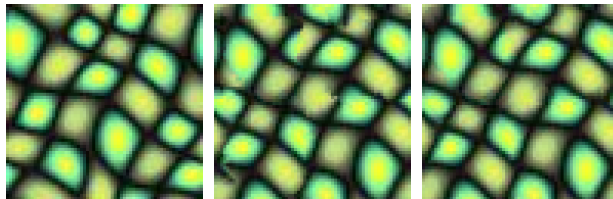


Figure 2. Left: the input texture. Middle and right: 64×64 synthesis results using four 32×32 patches. Using an Ashikhmin-based (middle) and an exhaustive search strategy (right) for overlap re-synthesis.

To increase the number of possible source locations in the input texture, we add pixels that are similar to the regions of the overlapping patches. This is essentially a variant of Tong et al.’s k -coherence search [9]. More precisely, for a given input texture with M pixels and user provided

parameters k (number of nearest neighbors) and n_p (the $n_p \times n_p$ box shaped pixel-neighborhood), we precompute a k -nearest neighbors (knn) data structure, where we store for each of the M input pixels a list of its k -nearest neighbors. First, we construct for each input pixel the associated high dimensional feature vector by ordered concatenation of the $n_p \times n_p$ RGB-triples in the box-shaped neighborhood. To this set of feature vectors we apply Principal Component Analysis (PCA), retaining 95-97% of the original variation. Depending on the input texture’s structure and color variation, this generally reduces the feature vector dimensionality by 75-90%. We then compute the k -nearest neighbors to each pixel in the input using the freely available TSTOOL Matlab package. This knn data structure is computed and stored once for every new texture we wish to synthesize.

Using the precomputed knn list, for each re-synthesized pixel, we now extend the candidate set of pixels in the source texture related to the valid neighbors (C_a) by the k -nearest neighbors of each pixel in this set, resulting in the extended candidate set C_{ext} (the two hatched and five black pixels in Fig. 1, bottom right). Depending on the input texture, the value for k and the neighborhood size, we can synthesize textures equal in quality to those generated by exhaustive search during per-pixel re-synthesis, but with a significant speedup: the runtime synthesis algorithm is relieved of the burdening Fourier Transforms in the overlap re-synthesis stage. Now we only need to perform a small amount of L_2 -comparisons between the re-synthesized pixel neighborhood and candidate set neighborhoods, selecting the best pixel therein.

The only question that remains is: how to maintain/update the source map(s) in our setting. Valid pixels in the overlap region are the result of linearly blending the pixel from neighboring patches (using an alpha mask with values in the range $[0,1]$). Strictly speaking, valid pixels lack a reasonable location in the source texture. A possible (but expensive solution) is to maintain multiple source maps, one for each contributing pixel. By experimentation, we have discovered that maintaining a single source map (based on the pixel with the largest contribution) ensures a sufficiently large candidate set. In practice, each time we copy a new patch into the synthesis result, if the alpha mask value is greater than 0.5 (= more contribution from the newly added pixel), we store the source index of the new pixel, otherwise we retain the existing index. The results from the single source map algorithm are visually indistinguishable from experiments with multiple source maps.

4. Tradeoffs

The box-shaped neighborhood sizes n_p and n during pre-computation and at runtime, and the numbers k_p and k of nearest neighbors for precomputation and at runtime allow

a trade-off between quality and speed. Smaller values for k and n lead to faster synthesis at the potential cost of visual quality. Note that setting $k = 1$ results in an approach comparable to Ashikhmin’s search strategy (Fig. 2, middle). For values of $n_p = n = 7 \times 7$ we show the synthesis quality for various settings of k in Fig. 3. We find that while setting $k = 1$ results in a visible loss in quality, using $k = 4$ is acceptable, whereas setting $k = 11$ is identical to the results of exhaustive search.

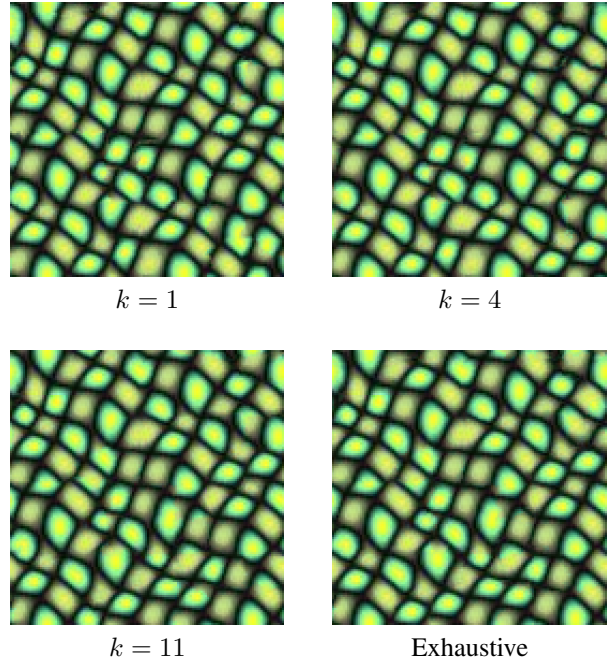


Figure 3. Four 128×128 synthesis results with varying values for k , using Fig. 2 (left) as input (initial patch size = 32×32).

We furthermore allow the user to precompute the knn data structure using grayscale instead of RGB feature vectors, which works fine for images with sparse histograms.

5. Results

As expected, offloading the exhaustive search procedure to a precomputation step relieves the runtime synthesis algorithm of this effort. Table 1 shows some results/timings using our Matlab implementation including adaptively subdivided patches to control the initial amount of error in the overlap regions (see [7]). As mentioned in Section 2, an exhaustive search during overlap re-synthesis scales badly with the input texture size. This can be verified from the synthesis times in column 2 of Table 1, where the result in each row is of size 192×192 with varying input texture sizes. Columns 3 and 4 demonstrate that the use of a fixed size candidate list leads to synthesis times, which are linear


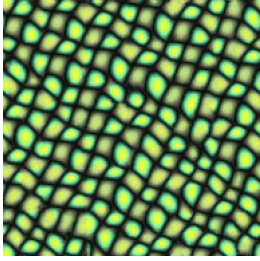
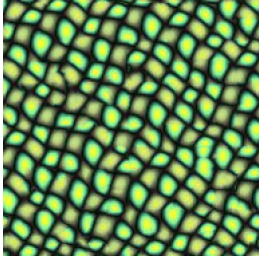
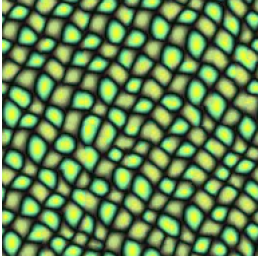
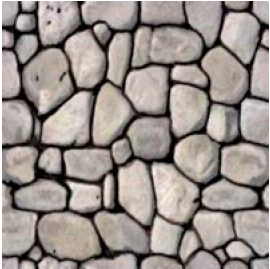

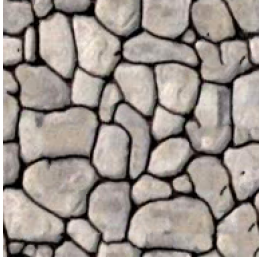
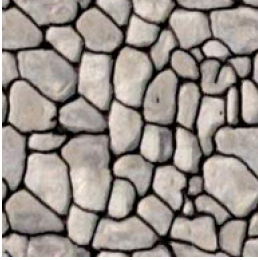
input	exhaustive search, neighborhood $n = 7 \times 7$	k -coherence search, neighborhood $n = 3 \times 3$, $k = 5$	k -coherence search, neighborhood $n = 5 \times 5$, $k = 11$
 <p>scales 64×64 $\delta_{max} = 0.02$ (see [7]) $\Delta_{max} = 0.05$ (see [7])</p>	 <p>pre: 0 sec. synth: 238 sec.</p>	 <p>pre: 6+3 sec. synth: 226 sec.</p>	 <p>pre: 6+4 sec. synth: 427 sec.</p>
 <p>stone wall 200×200 $\delta_{max} = 0.02$ (see [7]) $\Delta_{max} = 0.03$ (see [7])</p>	 <p>pre: 0 sec. synth: 985 sec.</p>	 <p>pre: 247+28 sec. synth: 178 sec.</p>	 <p>pre: 247+37 sec. synth: 350 sec.</p>

Table 1. Some timings on an Athlon xp2100 using Matlab. Each result is 192×192 (initial patch size of 32×32). The precomputation time is pre: $f + s$, where f is the time spent assembling the feature vector set, and s is the time for applying PCA to the RGB feature vectors and then performing the actual k nn search.

in k and n and nearly independent of the input texture size.

6. Conclusions and Future Work

Our proposed method works for any patch layout or shape. The results are of equal quality when compared to exhaustively searching for the best pixel to replace invalid pixels in the overlap region, however, is significantly faster for large input textures (i.e. the synthesis time is independent of input texture size). Given this improvement, we see several avenues for further research.

During precomputation we currently use all feature vectors for PCA, which, depending on the input texture's variation, is possibly not necessary for good results.

Our acceleration is analyzed using an existing, prototype Matlab implementation. An equivalent C/C++ implementation of the algorithm would achieve significant speedup.

Zelinka and Garland's Jump Map [11] is exceptionally fast, so using their algorithm for overlap re-synthesis could result in even greater acceleration.

References

- [1] M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.
- [2] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. In *Proceedings of ACM SIGGRAPH 2003*, pages 287–294. ACM Transactions on Graphics, 2003.
- [3] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 341–346. ACM Press / ACM SIGGRAPH, 2001.
- [4] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *IEEE Conference on Computer Vision*, pages 1033–1038, 1999.
- [5] V. Kwatra, A. Schdl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of ACM SIGGRAPH 2003*, pages 277–286. ACM Transactions on Graphics, 2003.
- [6] L. Liang, C. Liu, Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, 2001.
- [7] A. Nealen and M. Alexa. Hybrid texture synthesis. In *Rendering Techniques 2003, 14th Eurographics Workshop on Rendering*, pages 97–105. Eurographics Association, 2003.
- [8] C. Soler, M.-P. Cani, and A. Angelidis. Hierarchical pattern mapping. In *SIGGRAPH 2002 Conference Proceedings*, July 2002.
- [9] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH 2002 Conference Proceedings*, pages 665–672. ACM Press, 2002.
- [10] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 2000 Conference Proceedings*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.
- [11] S. Zelinka and M. Garland. Towards real-time texture synthesis with the jump map. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 99–104. Eurographics Association, 2002.