Vision, Modeling, and Visualization (2010)

Instant Texture Synthesis by Numbers

P. Panareda Busto[1,2], C. Eisenacher[1], S. Lefebvre[3] and M. Stamminger[1]

[1]FAU Erlangen-Nürnberg, Germany
[2]Technical University of Catalonia, Spain [3]INRIA Nancy, France

Abstract

Appearance Space Texture Synthesis (ASTS) provides fast texture synthesis by example. Unfortunately, speed is only achieved at the cost of a long pre-processing step. This is tedious for artists and limits the input resolution. In addition, the aggressively pruned search space results in strong artifacts when synthesizing multiple textures under user guidance (Texture-By-Numbers, TBN). In this paper, we replace the k-coherence search step used by most modern synthesis algorithms with a new parallel, coherent random walk. We show that this drastically improves the synthesis quality with TBN, while maintaining the parallel nature, speed, and flexibility of the original ASTS runtime. Since it removes the expensive pre-computation of k-coherent candidates, we are able to use larger inputs, and start synthesis much faster. This is essential for artists designing high-quality exemplars.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Texture Synthesis

1. Introduction

Texture synthesis by example is a convenient way for artists to create a large amount of similar looking texture. The ideal algorithm synthesizes gigatexels of high quality texture onto arbitrary surfaces. It does it in a short time, and with a maximum of optional artist control. By separating synthesis into careful pre-processing and parallel *k*-coherence search on a GPU, Appearance Space Texture Synthesis (ASTS) [LH06] is able to meet most of these requirements.

However, it has two drawbacks, limiting its practical use. First, while the synthesis runtime is incredibly fast, we need to pre-compute *k*-coherent candidates. This takes considerable time, and using tree-based acceleration structures [AMN*98, ML09], we quickly run out of memory as the size of the exemplar increases. This is common for TBN as the input accommodates multiple textures and the transitions between them. Second, the severely pruned search space of *k*-coherence does not provide enough variation for Texture-By-Numbers (TBN) [HJO*01]. This leads to strong transition artifacts as shown in Figure 1(b).

Inspired by Barnes et al. [BSFG09] we design a new parallel, coherent random walk, that nicely integrates into the ASTS synthesis step. It requires no pre-processing and improves the image quality with TBN considerably over *k*-coherence search. Our algorithm provides artists with greater control over the result than plain ASTS, and allows them to try different ideas in shorter time, increasing their productivity greatly.

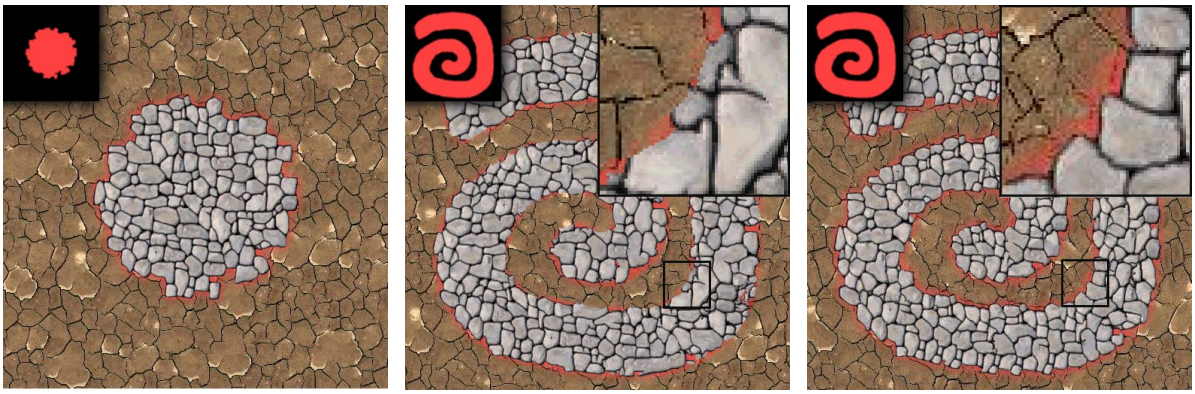
2. Related Work

There is a large body of excellent texture synthesis algorithms, catering different needs [WLKT09]. They can be roughly categorized into pixel-based [EL99], patch-based [EF01] and optimization based [KEBK05, RB07] approaches. The latter two are very successful synthesizing 2D images, and frequently used for image-editing applications.

Synthesizing large amounts of texture onto virtual objects, however, is an area where pixel-based synthesis algorithms excel [LH06]. Their basic operation is to repeatedly replace each synthesis pixel with the most similar exemplar pixel, where similarity is generally the Euclidean distance between the neighborhood vectors around each pixel. With each such *correction pass* the synthesized image becomes more similar to the exemplar.

To guide synthesis [Ash01], the neighborhoods are supplemented by information from a *label map* [HJO*01]. In addition to similar texture, the exemplar neighborhood now has to carry similar labels or *numbers* in order to be chosen.

DOI: 10.2312/PE/VMV/VMV10/081-085



(a) input exemplar with annotation (b) k-coherence search (c) parallel randomized correspondence

Figure 1: *Texture by Numbers takes an exemplar with a source label (left), and synthesizes a texture following a target label. K-coherence search fails due to excessive pruning of the search space, especially at the transitions between textures (middle). Our parallel, coherent random walk drastically improves the result and avoids costly pre-processing (right).*

To speed up the frequent and expensive search for similar neighborhoods, most algorithms employ hierarchical synthesis [WL00], local propagation [Ash01], or dimensionality reduction [HJO*01]. Lefebvre and Hoppe combine all three strategies for an extremely fast, parallel GPU implementation [LH05]. One of their key ideas is to synthesize coordinates into the exemplar instead of colors. This allows them to naturally propagate good matches to adjacent pixels and the next level of the synthesis hierarchy, resulting in excellent synthesis quality.

To further improve speed and quality, they gather neighborhoods for each exemplar pixel and PCA compress them to create the *Appearance Space* [LH06]. Its higher information density allows to define a synthesis neighborhood with only four appearance space points and use it instead of the traditional color neighborhoods to synthesize exemplar coordinates. While it contains information about a very large area of the input, it is considerably faster and the small spatial extent allows anisometric synthesis into a texture atlas.

3. Searching Neighborhoods Efficiently

Most successful search strategies locally propagate good matches, and then add some additional candidates into the mix to avoid the creation of texture seams.

Coherence search: Ashikhmin [Ash01] limits the search space to the *coherent candidates*, i.e. the coordinates of the adjacent pixels plus the respective offset, as shown in Figure 2(a). This requires only 4 comparisons per pixel, and the scanline order propagates good matches, resulting in the growth of coherent patches. While this is beneficial for image quality, it produces disturbing seams where two patches join, especially if the exemplar does not contain enough high frequencies to visually mask those seams.

***K*-coherence search:** To relax the restricted search space, Tong et al. pre-compute the k most similar neighborhoods for each exemplar pixel [TZL*02]. During synthesis they choose from the coherent candidates and their similarity sets. This requires $4 \times k$ comparisons, but is able to recover from seams. For $k = 1$ it is equal to coherence search. Lefebvre and Hoppe [LH05, LH06] use *parallel* k-coherence, as shown in Figure 2(b). This requires $9 \times k$ comparisons, but improves quality for small k without additional memory or pre-computation cost. Typically $k = 2$ suffices.

Patch Match: Barnes et al. [BSFG09] use a simple coherence search in scanline order, and replace the similarity sets by a random search with contracting radius around the best coherent candidate, as shown in Figure 2(c). The basic idea is that while good matches can be anywhere in the exemplar, the probability to find a good one is higher nearby. This avoids pre-computation and provides very good image quality, as it is able to escape from local minima given enough iterations. The number of comparisons per pixel is dependent on the initial search radius and the rate of contraction.

4. Instant Texture Synthesis by Numbers

Our goal is to provide fast, high quality guided texture synthesis within the ASTS framework. To archive this we design an equally efficient, but less restrictive replacement for the k-coherence search.

4.1. Parallel Coherent Random Walk

As shown in Figure 2(d) we determine the best coherent candidate using parallel k-coherence with $k = 1$. This provides an excellent initial guess for a random search. However, we then perform a random search with decreasing radius around the *currently* best matching neighborhood.

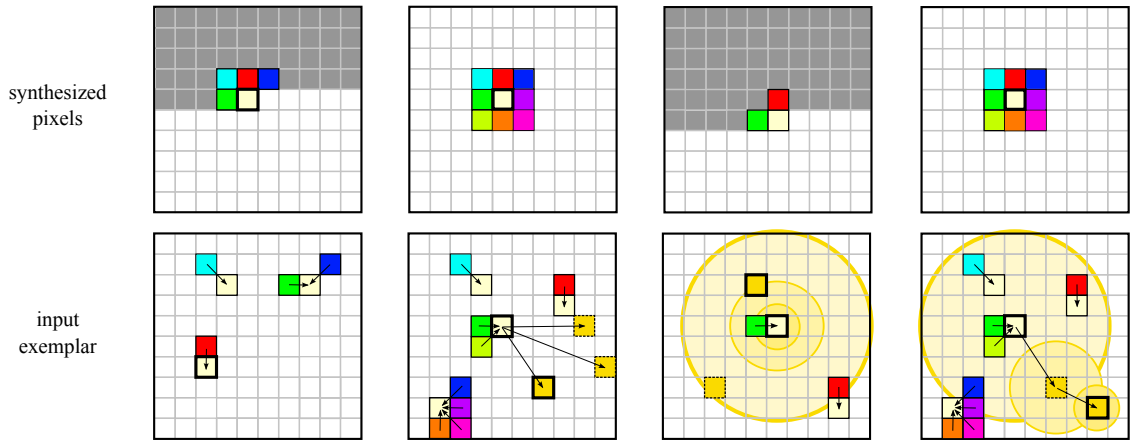


Figure 2: *Comparison of successful strategies for searching the most similar neighborhood for a synthesized pixel (yellow): Coherence search only tests the coherent candidates suggested by adjacent pixels (a). (Parallel) k-coherence also considers the pre-computed similarity set of each coherent candidate (b). PatchMatch performs a random search with decreasing radius around the best coherent candidate (c). We perform a random walk, updating the search center while decreasing the radius (d).*

I.e., in contrast to PatchMatch we update the center of the search, if we found a better match. We thus in fact perform a random walk with a fixed number of steps, seeking a region of the exemplar that is a better source of candidates. This improves the convergence speed without additional cost, and is especially useful for TBN, as it provides opportunities to walk toward proper transition areas. All examples in this paper use only two iterations of our random walk.

As shown in Algorithm 1, we half the search radius for each step. This is fast, simple, and provides a very good speed-quality tradeoff for a wide range of inputs.

4.2. Analysis

As our search strategy does not require k-coherent candidates, the only task left from the original ASTS analysis stage is to gather 5x5 neighborhoods from the exemplar, and PCA project them to 8D appearance space. If the input size is larger than a certain threshold, we only use a pseudo-random subset of the neighborhoods to compute the principal components. This limits memory and computational cost. For all shown examples we use a threshold of 128^2 and a simple linear congruentional random number generator [PTVF07].

The source and target label maps are treated similarly, but are projected to 4D, as they tend to contain less information than the exemplar. Further we add a small Gaussian blur to the labels. This is only a rough approximation of feature distance, but generally improves synthesis quality.

Finally, we construct the 4-point runtime neighborhood, append the projected label, and project the result to 8D.

Algorithm 1 randomWalk($coord_{best}$)

```
1: radius = resolution / 2
2: while (radius >= 2) do
3:     cand_r = getRandomAround(coord_best, radius)
4:     NBH_r = getExNeighborhood(cand_r)
5:     if (isMoreSimilar(NBH_r)) then
6:         coord_best = cand_r
7:     end if
8:     radius /= 2
9: end while
```

4.3. Synthesis

Due to the hierarchical nature of ASTS we already have a good propagation mechanism, so we require only minimal changes: as shown in Algorithm 2 we simply perform our random walk instead of comparing k-coherent candidates.

Algorithm 2 parallelCorrectionPass()

```
1: S_in = getPreviouslySynthesizedCoordinates()
2: S_out = allocNextSynthesizedCoordinates()
3: for each x, y of S_out in parallel do
4:     NBH_syn = gatherSynNeighborhood(x, y, S_in)
5:     coord_best = coherenceSearch(x, y, S_in)
6:     coord_best = randomWalk(coord_best)
7:     S_out(x, y) = coord_best
8: end for
```

As neighborhoods are ill defined near the edges of non-tilable exemplars, we ensure that candidates are from within the interval $[2^{lvl}, resolution - 2^{lvl}]^2$. To further improve

quality with TBN, we give more weight to the label map at coarse levels: at these levels the actual textures are hardly discernible, and candidates are thus preferably selected due to matching labels. With coordinate upsampling their children naturally inherit that label, and at finer levels the coherent candidates match the target label more frequently.

5. Results and Discussion

We compare k-coherence to our approach using a multithreaded CPU implementation of ASTS derived from the tutorial code of Lefebvre [Lef09]. We cannot hope to reach the absolute timings of the original, highly optimized GPU implementation of the ASTS synthesis runtime, but it allows us to experiment with different strategies quickly.

5.1. Quality

As Figure 3 illustrates, both k-coherence search and our parallel, coherent random walk produce excellent results for unguided synthesis. As the exemplar only contains one texture, the assumption that nearby pixels provide good matches holds, and the 9 coherent candidates often suffice. The k-coherent or random candidates rarely need to override them.

However, as Figures 1, 4 and 5 demonstrate, our strategy clearly outperforms k-coherence for TBN. The problem is that coherence search only tests exemplar pixels that are compatible with adjacent synthesis pixels. I.e., in a transition area a stone pixel will mostly suggest coherent candidates from a stone area, and a grass pixel is likely to suggest pixels from a grass area. As k-coherence only prepares candidates that are similar to the pixels making the poor coherent suggestions, transition pixels – or even pixels from a different label – will rarely be introduced. In contrast to this, our random walk searches through the complete image. It can find a transition pixel and suggest it to adjacent pixels.

As our strategy can ”switch labels“ during the random walk, it converges faster for TBN than the concentric search of PatchMatch. Figures 4(c) and 4(d) compare the results after only two iterations.

5.2. Performance

Analysis: For the pre-computation of k-coherence search, we use ANN [AMN*98] and one thread for each level of the exemplar stack. The latter trades memory for time, and is an interesting test for multicore systems. As Table 1 illustrates, searching the k best matching 7x7 neighborhoods [LH05] is prohibitively expensive, and runs out of memory quickly. Using 5x5 neighborhoods is faster, but still runs out of memory. Using the 8D appearance space vectors eases memory consumption, but also becomes expensive for large inputs. Our search strategy does not need pre-processing at all, and hence exhibits none of the memory, performance, or quality problems resulting from small and projected vectors.

input size	PCA	k-coh pre-process 7x7	5x5	8D	Synthesis k-coh	we
64^2	<1	<1	<1	<1	12.1	10.6
128^2	2	37	7	<1	12.6	11.3
256^2	2	413	97	1	12.9	12.0
512^2	3	(-)	(-)	7	13.9	13.1
1024^2	7	(-)	(-)	28	16.6	16.1

Table 1: *Synthesizing a 1024^2 image with varying exemplar size. Time in seconds, (-) means not enough memory.*

Synthesis: The timings in Table 1 also show, that both search strategies have comparable cost during synthesis. Synthesis times for both increase with exemplar size, as larger exemplars can perform correction on more levels of the synthesis pyramid. For small exemplars we have a slight advantage, as the random walk starts with a smaller search radius, and we have less neighborhood comparisons in total.

6. Conclusion and Future Work

We have presented a simple modification to Appearance Space Texture Synthesis that maintains the speed and quality of the original synthesis runtime, but drops most of the preprocess and allows it to be used with the Texture-By-Number control metaphor. This gives artists greater control over the synthesized textures and allows them to design interesting exemplars much more quickly.

Since k-coherence is used in many texture synthesis algorithms, we believe our approach will benefit several state-of-the-art techniques. As it only requires small code changes, it allows to update existing implementations easily. We intend to map our algorithm to a GPU, to give more meaningful performance comparisons to the original ASTS. We will further investigate local updates to the appearance space, in order to allow the truly interactive design of exemplars.

References

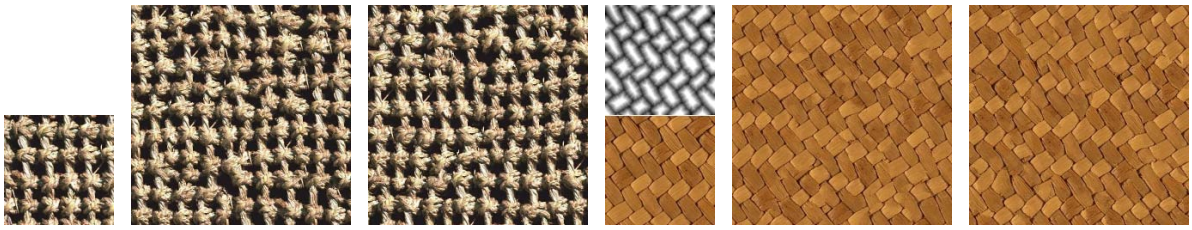
[AMN*98] ARYA S., MOUNT D., NETANYAHU N., SILVERMAN R., WU A.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM) 45*, 6 (1998), 891–923.

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *Proceedings of ACM Symposium on Interactive 3D Graphics* (2001), pp. 217–226.

[BSFG09] BARNES C., SHECHTMAN E., FINKELSTEIN A., GOLDMAN D. B.: Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. on Graphics 28*, 3 (2009), 24:1–24:11.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH* (2001), pp. 341–346.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision* (1999), pp. 1033–1038.

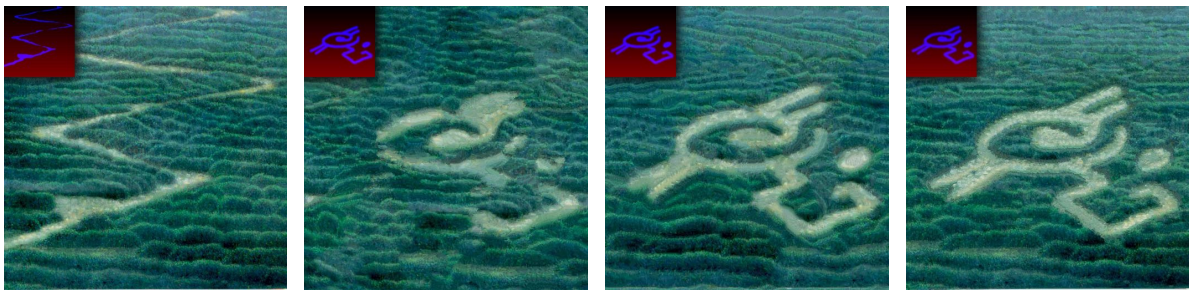


(a) exemplar (b) *k*-coherence (c) our strategy (d) exemplar (e) *k*-coherence (f) our strategy

Figure 3: *For unguided synthesis our parallel, coherent random walk produces similar quality without pre-processing.*

(a) exemplar (b) *k*-coherence (c) concentric search (d) random walk

Figure 4: *For guided synthesis random search strategies (c, d) clearly beat the quality of k-coherence. At identical cost our strategy (d) converges faster than the concentric search of PatchMatch. Generally we see good quality after only two iterations.*



(a) exemplar (b) *k*-coherence (c) our strategy (d) original TBN [HJO*01]

Figure 5: *Our quality is close to that of Hertzmann [HJO*01], but does not have the size restrictions imposed by a search tree.*

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *SIGGRAPH* (2001), pp. 327–340.

[KEBK05] KWATRA V., ESSA I., BOBICK A., KWATRA N.: Texture optimization for example-based synthesis. *ACM Trans. on Graphics 24*, 3 (2005), 795–802.

[Lef09] LEFEBVRE S.: Tutorial code. http://www-sop.inria.fr/members/Sylvain.Lefebvre/wiki/Main/TSynEx, June 2009.

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Trans. on Graphics 24*, 3 (2005), 777–786.

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Trans. on Graphics 25*, 3 (2006), 541–548.

[ML09] MUJA M., LOWE D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of VISSAPP* (2009), pp. 331–340.

[PTVF07] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P.: *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2007.

[RB07] RAMANARAYANAN G., BALA K.: Constrained texture synthesis via energy minimization. *IEEE Trans. on Visualization and Computer Graphics 13* (2007), 167–178.

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.-Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH* (2002), pp. 665–672.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH* (2000), pp. 479–488.

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report* (2009), EG Association.