Volume 26 (2007), number 2 pp. 131–142 COMPUTER GRAPHICS *forum*

Texture Synthesis using Exact Neighborhood Matching

M. Sabha, P. Peers and P. Dutré

Department of Computer Science Katholieke Universiteit Leuven
{muath.sabha, pieter.peers, philip.dutre}@cs.kuleuven.be

Abstract

In this paper we present an elegant pixel-based texture synthesis technique that is able to generate visually pleasing results from source textures of both stochastic and structured nature. Inspired by the observation that the most common artifacts that occur when synthesizing textures are high-frequency discontinuities, our technique tries to avoid these artifacts by forcing at least one of the direct neighboring pixels in each causal neighborhood to match within a predetermined threshold. This does not only avoid deterioration of the visual quality, but also results in faster synthesis timings. We demonstrate our technique on a variety of stochastic and structured textures.

Keywords: texture synthesis, pixel-based techniques, Markov Random Field

ACM CCS: I.3.3 Computer Graphics: Picture/Image Generation I.3.7 Computer Graphics: *Three-Dimensional Graphics and Realism, color, shading, shadowing and Texture.*

1. Introduction

The goal of texture synthesis is to generate new textures that look similar to a given sample texture. Texture synthesis is used extensively in computer graphics and computer vision applications, such as texture mapping [PFH00], image completion and restoration [DCY03], motion synthesis [WL00], film post-production and the compression of images and video sequences.

Recently, a multitude of texture synthesis techniques has been developed. However, each of these previous techniques are geared towards a specific texture type. For example, pixel-based texture synthesis techniques excel in synthesizing textures of a stochastic nature, whereas patch-based techniques are proficient in synthesizing from near-regular sample textures.

The goal of this paper is to develop a fast texture synthesis technique that is able to generate visually pleasing results from a wide range of texture types. However, when blindly applying any of the previously developed methods to different texture types, the generated results can show high-frequency discontinuities such as cuts and edges (Figure 1). The human eye is particularly sensitive to these high-frequency errors and therefore these kinds of errors should be avoided as much as possible.

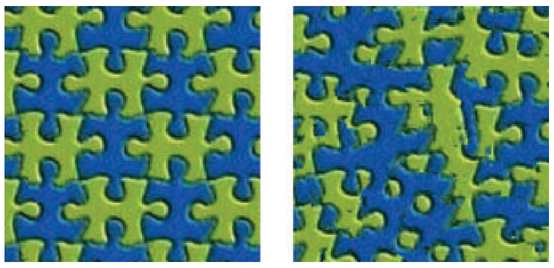

Figure 1: *Left a texture synthesized with the presented method without high-frequency discontinuities and a similar look as the source texture. Right: a synthesized texture distorted by high-frequency discontinuities.*

In this paper we present an elegant pixel-based texture synthesis technique, which is able to generate visually pleasing results from stochastic and near-regular structured sample textures. Key to our method is the reduction of the search space for each pixel such that high-frequency discontinuities are avoided as much as possible. Additionally, since the search space is reduced, a synthesis speed-up is attained. This reduction is achieved by a priori restricting the search space to contain only causal neighborhoods that minimize

Submitted: September 2005
Revised: January 2006
Accepted: June 2006

the probability of introducing high-frequency artifacts in the synthesized results.

2. Previous Work

Texture synthesis has been an active research topic during recent years. Published techniques can be roughly subdivided in two main categories: procedural texture synthesis, and texture synthesis by example.

Procedural texture synthesis is basically a customized *program* that transforms some predefined signal (e.g. Perlin noise [Per85], Worley noise [Wor96], ...) into a desired texture. This produces a high-quality, continuous texture. However, a major limitation of procedural texture synthesis approaches is that creating a new texture requires a new program to be written, usually a trial and error process, to achieve a texture that has the right 'look'. For more information, we refer the reader to [EMP*02].

Texture synthesis by example generates a novel texture that is similar to a given sample texture. There are three classes of algorithms: texture synthesis by analysis, pixel-based texture synthesis and patch-based texture synthesis.

Texture synthesis by analysis usually characterizes a sample texture by a limited number of statistics. A new texture is synthesized such that the statistics of the sample texture are maintained. [HB95] proposed to analyse textures in terms of histograms of filter responses at multiple scales and orientations. [PS00] were able to substantially improve synthesis results for structured textures at the cost of a more complicated optimization procedure. [Deb97] scrambles the input in a coarse-to-fine fashion, preserving the conditional distribution of filter outputs over multiple scales. [NMM*05] separate regular features with the aid of a fractional Fourier analysis on a near-regular texture, which are then tiled. Afterwards, irregular texture detail is added back in to the tiled texture.

Pixel-based texture synthesis generates novel textures by repeatedly selecting and copying a single pixel from the sample texture, based on already synthesized pixels in the novel texture. These algorithms are generally based on the theory of Markov Random Fields, a two-dimensional extension to Markov Chains [PL98].

'Non-parametric sampling' [EL99] synthesizes a novel texture by selecting pixels, with high conditional probability, from a sample texture. The conditional probability between a pixel in the sample texture and a to-be-synthesized pixel is defined by a Gaussian weighted normalized sum of the squared differences of the pixel values in a small neighborhood around each pixel.

Wei and Levoy [WL00] use a fixed causal neighborhood size and interpret all possible neighborhoods in the input texture as a set of vectors that span a high-dimensional search space. Tree structured vector quantization (TSVQ) is used to accelerate searching this space. Furthermore, the algorithm is extended using a multi-resolution synthesis pyramid. Although a number of newer pixel-based techniques have been developed, this technique is still used extensively because of its simplicity and robustness with respect to a wide range of texture types.

Ashikhmin [Ash01] modified the algorithm of [WL00] to encourage verbatim copying of parts of the input sample. Unlike [WL00] where for each pixel all causal neighborhoods in the sample are compared, only *four* neighborhoods per pixel are checked. These four neighborhoods are defined by the corresponding causal neighborhoods in the sample texture of the already synthesized neighboring pixels of the to-be-synthesized pixel. Ashikhmin [Ash01] noted that his algorithm works best on natural textures, such as textures of flower fields, pebbles, forest undergrowth, bushes and tree branches. However it is not suited for textures containing structured features.

Hertzmann *et al.* [HJO*01] introduced an algorithm that handles both texture synthesis and texture transfer. The works of [WL00] and [Ash01] are combined and extended to work on corresponding pairs of images rather than on single textures.

Zelinka and Garland [ZG02] accelerate texture synthesis by using a jump map. Each pixel in the jump map contains a list of pre-calculated references and probabilities for matching pixels. A texture is synthesized in real time by copying a matching pixel, referred in the jump map, from the sample texture.

Tong *et al.* [TZL*02], presented k-coherence, a method for synthesizing bidirectional texture functions. K-coherence can also be used for *normal* texture synthesis and is closely related to [Ash01] and the technique presented in this paper. K-coherence stores for each pixel a set of k nearest causal matches. Similar to Ashikhmin [Ash01], the source pixels in the causal neighborhood are used to define a candidate set from which the best matching pixel is copied. Unlike Ashikhmin, who directly uses the causal neighborhoods around the source pixels, k-coherence creates the candidate list from the k pre-computed best-matching neighborhoods for each source pixel.

Recently, Battiato *et al.* [BPR03] extended the texture synthesis technique of [WL00] by using antipole clustering, instead of TSVQ, to speed up texture synthesis, yielding improved synthesis results.

Patch-based texture synthesis. Efros and Freeman [EF01] point out that pixel-based texture synthesis algorithms like those of Efros and Leung [EL99], Wei and Levoy [WL00] and Ashikhmin [Ash01], all perform excess computations when dealing with structured textures. They propose to synthesize a novel texture by copying whole patches from the sample texture, as opposed to copying a single pixel in the pixel-based techniques. Other early work on patch-based texture synthesis was performed by [XGS00] (Chaos Mosaics), [PFH00] (Lapped Textures), [EF01] (Image Quilting) and [LLX*01].

Cohen *et al*. [CSH*03] propose to use Wang tiles for patch-based texture synthesis. Wang tiles can be used to generate non-repeating tilings of a limited number of tiles by assigning a color to each tile's edge and matching only edges with similar color. To extend Wang tiles for texture synthesis, Cohen *et al*. assign a texture patch to each Wang tile, making sure that tiles with common colors can be matched without introducing artifacts in the synthesized texture. The major advantage of this approach is that the tiled texture is guaranteed non-repeating.

Kwatra *et al*. [KSE*03] copy irregularly shaped patches from a sample image to generate a new texture. The process of copying patches is performed in two stages. First, the best candidate rectangular patch is selected by comparing the pixels in a candidate patch with already synthesized pixels. Second, an optimal portion of the rectangular patch, determined using a graph cut algorithm, is copied to the synthesized texture. The textures synthesized with this technique are of a very good visual quality.

Nealen and Alexa [NA03], presented a hybrid patch-based texture synthesis technique that tries to use as large as possible patches by adaptively splitting them. To stitch the patches of different sizes together, a pixel-based method is used. In [NA04], this method is extended and speeded-up by replacing the pixel-based synthesis technique by a k-coherence- based technique [TZL*02].

Liu *et al*. [LLH04] treat a near-regular texture as a statistical distortion of a regular tiling, possibly with individual variations in tile shape, size, color and lighting. Textures are synthesized by tiling the regularized texture, and applying the statistical distortions afterwards to the synthesized texture. The results of this technique are very good compared to other patch-based techniques.

3. Motivation

Although the observations made by Efros and Freeman [EF01] are still valid, pixel-based texture synthesis techniques are still widely used; for example for directly synthesizing on a 3*D* surface [Tur01] and for image completion [DCY03]. The main reason for this is the ease of use and ease of implementation and its strengths in synthesizing stochastic textures.

Our method is also a pixel-based texture synthesis technique, designed with the following goals in mind:

- **Simplicity**: Pixel-based methods are popular because they are straightforward to implement and easy to use.
- **Wide range of texture-types**: A disadvantage of pixel-based texture synthesis methods is that they are not really good in synthesizing textures from structured sample textures. Our method is able to handle these kinds of textures better.
- **Visually pleasing results**: We are not interested in generating a texture with the lowest mathematical error, but in synthesizing visually pleasing textures. We will therefore relax some constraints in current pixel-based systems to achieve better visual results.
- **Fast**: Pixel-based texture synthesis methods usually trade-off speed to synthesis quality. Our technique is both fast and able to generate high-quality results.

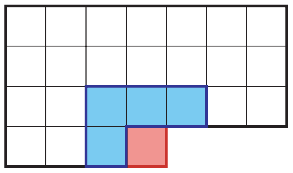
Figure 2: *An L-shaped causal neighborhood of size 7 (width) around a pixel marked in red. The pixels marked in blue are the four* ***direct neighbors****.*

As in many pixel-based texture synthesis techniques, we also assume a Markov random field model. A texture is modeled as a local and stationary random process: each pixel is classified by a vector representing a small set of neighboring pixels (local causality). This classification is similar for all pixels (stationary). The local causality principle allows us to construct a search space in which we would like to find the best-matching vector representing a given pixel's neighborhood. The stationary principle indicates that the search space is the same for all pixels. The dimensionality of the search space equals the required number of pixels in a neighborhood to make a faithful classification. Usually, a large neighborhood is required, and thus resides the search space in a high-dimensional space.

Assume we have a sequential (e.g. scanline order) synthesis algorithm and assume that the already synthesized pixels are chosen optimally and that we use an L-shaped causal neighborhood to classify each pixel. When synthesizing a new pixel, the only possibility to introduce a high-frequency discontinuity is in the transition between a direct neighbor and the to-be-synthesized pixel. In order to avoid these errors completely, the direct neighbors in the causal neighborhood in the sample texture should exactly match the corresponding pixel values of the direct neighbors around the to-be-synthesized pixel. An example of a causal neighborhood and its direct neighbors is depicted in Figure 2. The probability of finding such a causal neighborhood in the sample texture is very small. We relax this constraint by requiring at least one (instead of all) of the direct neighbors to match exactly. This ensures that for at least one direction no high-frequency discontinuities can occur. The causal neighborhoods in the sample texture, which have at least one matching direct neighbor define a reduced search space. In effect, we are reducing the search space by taking a slice through the complete search space in which a single dimension is fixed. This reduced search space is completely defined by the direct neighbors of the current to-be-synthesized pixel and thus for each pixel a different reduced search space is defined.

The reduced search space might not contain the optimal neighborhood vector in an L^2 sense, but we will show that this results mainly in low-frequency artifacts in the synthesized textures. Since the human visual system is more sensitive to high-frequency discontinuities than low-frequency artifacts, this approach results in visual pleasing results.

In the remainder of this paper, we will discuss how such a system can be efficiently implemented and a thorough analysis is conducted on the generated results.

4. Implementation

In the previous section we argued for reducing the search space for each pixel depending on the pixel values in the direct neighborhood. Instead of reducing the large stationary search space for each pixel separately, we opt for constructing the reduced search space on the fly. Each reduced search space can be seen as the union of the search spaces associated with a single direct neighboring pixel value. The search space associated with a single direct neighboring pixel are the causal neighborhoods that have the same pixel value at an identical position in the full search space. Thus if we know which pixel positions in the sample texture have the same color, then we also know the causal neighborhoods that share this color at a specific position. This reduces the construction of the search space to a simple search space look-up, in which the pixel value of the direct neighbor is the look-up key. This is illustrated in Figure 3.

We opt for using a kd-tree as a data structure to accelerate this look-up. This kd-tree is a 3-dimensional tree, in which each node corresponds to a unique color, represented by an RGB triplet, in the sample texture. Thus, the 3 dimensions of the kd-tree correspond to the red, green and blue channels in the RGB color-space. Associated with each node is a list of pixel positions in the sample texture containing this color. Figure 4 shows a schematic overview of the kd-tree.

The presented texture synthesis method consists of three steps: texture analysis (kd-tree construction), texture synthesis and synthesis initialization.

4.1. Texture analysis

During the analysis step, a kd-tree is constructed. Each pixel in the sample texture is added to the kd-tree. If the pixel value (RGB) is already in the kd-tree, then the pixel's location is added to the associated list of pixel positions. If the kd-tree

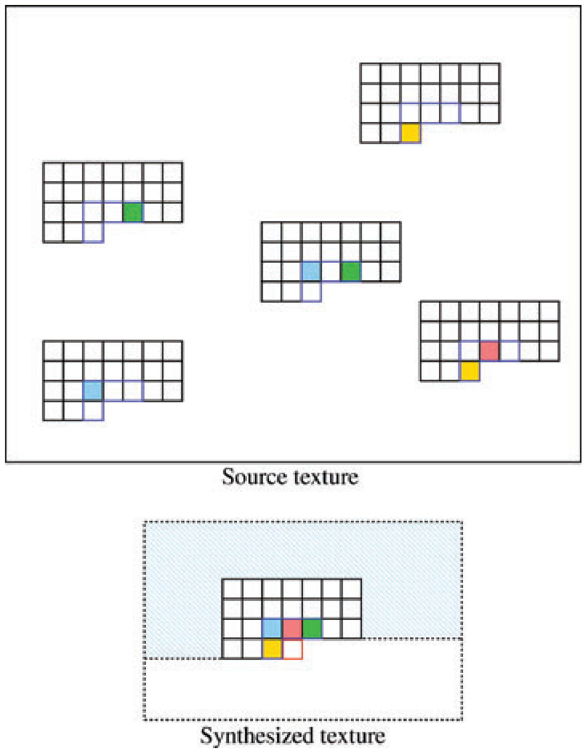

Figure 3: *The causal neighborhoods that need to be verified are uniquely determined by the pixels in the source texture that have the same color as the direct neighbors of the current pixel.*

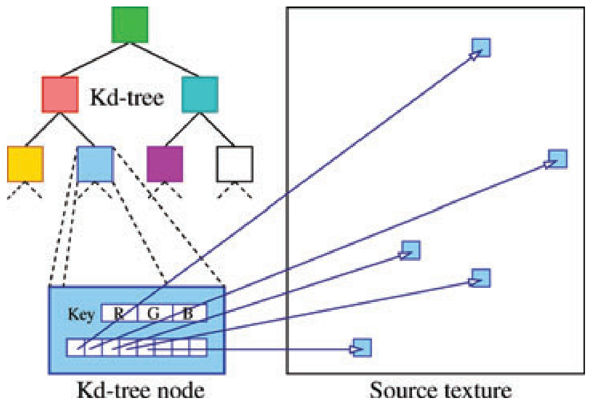

Figure 4: *A schematic depiction of the kd-tree used in our method. Each node of the kd-tree represents a unique color in the source texture. Associated with each node in the kd-tree is a list of pixel positions in the source texture of occurrences of the color of this node.*

does not contain the pixel value, then a new node is created and a list of pixel positions is associated. This list is initialized to contain only the current pixel position. Because pixels on the edge of the sample texture have an incomplete causal neighborhood, only the pixels which are above a predetermined distance from the edge are considered for inclusion in the kd-tree.

4.2. Texture synthesis

Texture synthesis is performed in scanline order. For each to-be-synthesized pixel, a reduced search space is constructed in the following manner: For each direct neighbor, a list of sample texture pixel positions is retrieved from the kd-tree using the direct neighbor's pixel value. Each position in the retrieved list is adjusted (depending on which direct neighbor was used as key on the kd-tree query) such that the resulting positions now represent the center positions of causal neighborhoods in the sample texture. The union of the four lists is a representation of the reduced search space. Next, for each causal neighborhood in the search space, the L^2-difference is computed with the corresponding causal neighborhood around the current to-be-synthesized pixel. The center pixel of the best matching (smallest L^2-error) causal neighborhood is copied into the current to-be-synthesized pixel.

As in most previous pixel-based texture synthesis algorithms, it is very important to select a 'good' causal neighborhood size. There is a direct correlation between the feature size in the sample texture and the minimum size of the causal neighborhood. However, a too large causal neighborhood size will result in excess computations and unnecessary prolonged synthesis timings.

4.3. Synthesis initialization

The texture synthesis process relies on previously synthesized pixels in order to select a good pixel value for the current pixel. When starting to synthesize a texture, no previously synthesized pixels are available. In order to bootstrap the synthesis process an initialization is required.

Depending on the texture type, two different texture initialization techniques are utilized. In both cases, the upper band of the texture is filled. The height of this band depends on the height of the causal neighborhood. In case the sample texture is stochastic in nature, the band is filled with randomly selected pixel values from the sample texture. In case the sample texture is near-regular, a structured upper band is required. However, randomly copying pixel values destroys the near-regular structure. Therefore, we synthesize the upper band in the following manner: We first rotate the upper band and the sample texture. Next a random block is copied from the rotated sample texture to cover the top of this rotated band. Because the width of the rotated texture is limited and smaller than the sample texture, a large enough block can be copied to the top to ensure a completely filled 'upper band' in the rotated upper band. Finally the band is completed using the texture synthesis technique described in the previous section. After synthesis, the band is rotated back and copied into the target texture. By working in a rotated texture, we ensure that as much as possible of the causal neighborhood overlaps with already synthesized pixels.

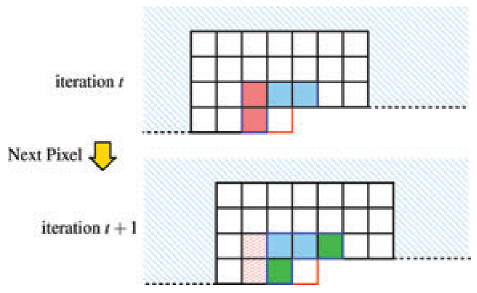

Figure 5: *When advancing to the next pixel, half of the associated lists of pixel positions can be reused. The lists associated with the blue marked pixels can be reused. The lists associated with the red marked pixels are discarded, while the green mark pixels have to be queried in the kd-tree.*

4.4. Optimizations

A number of general optimizations are possible:

- **Minimize kd-tree look-ups.** Traversing the kd-tree can be a costly operation, especially when the kd-tree is large, and the associated lists of sample texture positions are short. It is obvious that a number of the kd-tree look-ups of previously synthesized pixels can be reused, reducing the number of kd-tree queries by 50% (see Figure 5).
- **Avoid checking causal neighborhoods twice.** The intersection of partial search spaces defined by the direct neighboring pixels is not necessary empty. In order to avoid checking causal neighborhoods multiple times for a single pixel, already checked neighborhoods are marked, and subsequently ignored for the remainder of the synthesis of the current pixel.
- **Minimize associated list lengths.** For near-regular textures there is a large probability that multiple sample texture positions define a similar causal neighborhood. Since the causal neighborhoods associated to each sample pixel value are known on beforehand, it is very easy and fast to group similar causal neighborhoods into a single list-entry.

5. Results and Discussions

Figure 6 shows textures synthesized using our technique. The generated results cover a wide range of texture types, ranging from stochastic to near-regular textures. As illustrated, our

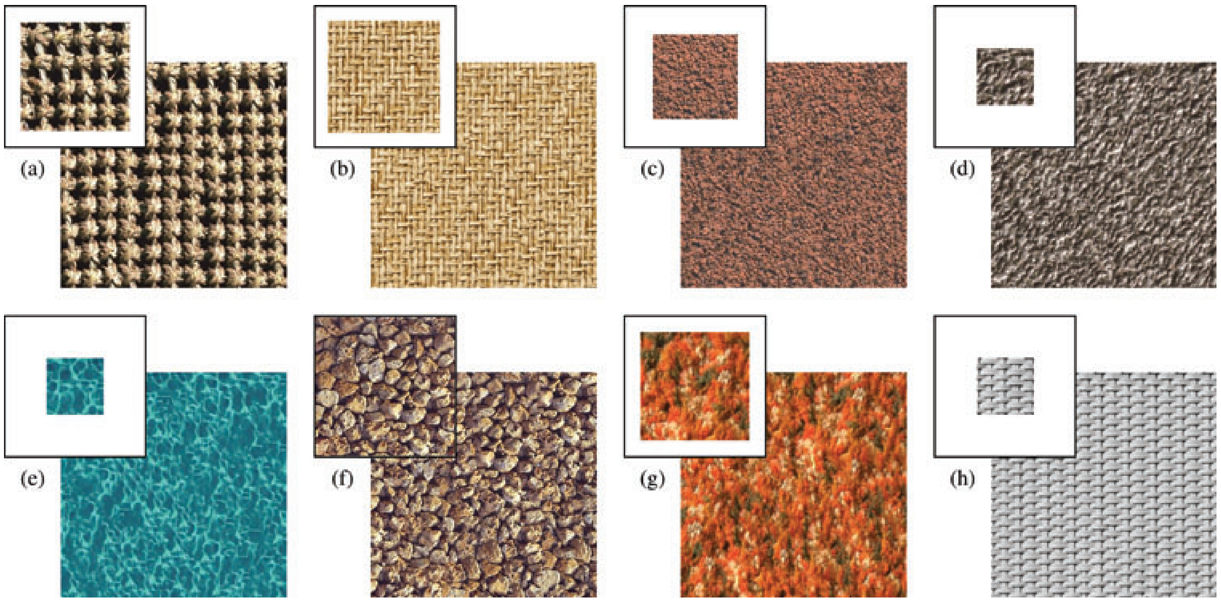

Figure 6: *Examples generated with our technique. The source texture is shown for each example in the top-left corner. Synthesis statistics can be found in Table 1.*

Table 1: *Synthesis statistics for the textures in Figure 6. All texture are generated at a resolution of 400 × 400 and all timings are provided in seconds. For each texture the source texture size, number of colors in the source sample and the causal neighborhood size used during synthesis are provided. The timings are split into three parts: time required for constructing the kd-tree (Analysis Timings), time required to synthesize the upper-band (Initialization Timings) and the time require to synthesize a 400 × 400 texture. The total time is also given. For comparison we also provide the time required for synthesizing the same texture (identical resolution, source texture and causal neighborhood size) using a brute force approach [WL00]. The ratio between the presented method and the brute force method are shown in the last column.*

Texture	Source size	Colors in source	Causal neighb size	Avg. tested neighb/pixel	Analysis timings	Initialization timings	Synthesis timings	Total timings	Brute force timings	Speed-up
a	192 × 192	12895	69	7.58	0.24	1.79	24.97	27.00	2754.00	102.00
b	199 × 199	11718	33	15.40	0.31	0.75	13.85	14.91	9360.00	627.77
c	150 × 150	17607	5	3.69	0.29	0.30	2.84	3.43	536.52	156.42
d	100 × 100	4201	7	17.10	0.07	0.10	2.88	3.05	351.18	115.14
e	100 × 100	6933	7	5.35	0.11	0.14	2.94	3.19	351.85	110.30
f	246 × 246	36772	19	5.40	0.94	1.01	4.85	6.80	7080.00	1041.18
g	192 × 192	28778	11	3.15	0.60	0.54	2.90	4.04	2125.72	526.11
h	100 × 100	234	33	188.63	0.01	3.30	121.94	125.25	1837.42	14.67

method is able to synthesize visually pleasing textures for different kinds of texture types.

Table 1 summarizes the computational costs for generating the results shown in Figure 6, together with various statistics such as sample texture size, number of distinct colors in the sample texture, the size of the causal neighborhood and the average number of tested causal neighborhoods per pixel. The timings are further split up in the time required for constructing the kd-tree, time required for the actual synthesis and the speed-up with respect to a brute force synthesis technique [WL00]. All examples were computed on an AMD64 4000+ processor (2400 Mhz). From this table we can conclude the following: the speed-up achieved compared to a brute force synthesis technique is on average 2 orders of magnitude, especially when using source textures with a large number of colors. The speed-up is less pronounced when using textures with very few different colors (e.g. example Figure 6 (h)). This is caused by the fact that there are more pixels in the source texture than colors, and thus a single color occurs

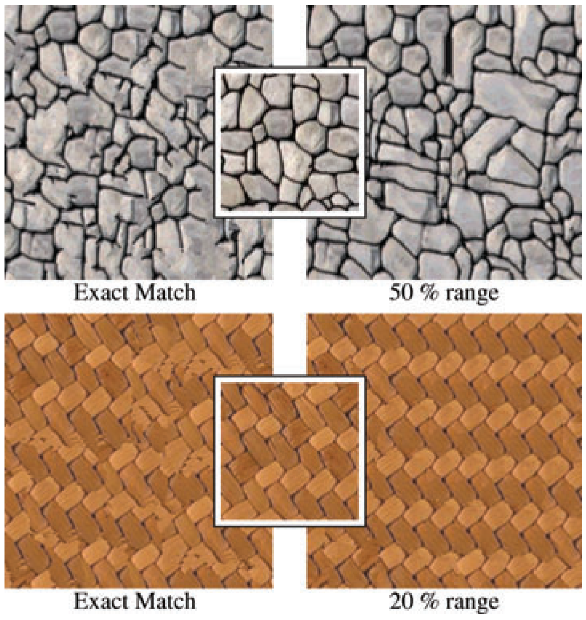

Figure 7: *Matching the direct neighbors exactly does not always result in a search space rich enough to ensure satisfying synthesized textures. By allowing a small error tolerance when matching direct neighbors, better results can be achieved.*

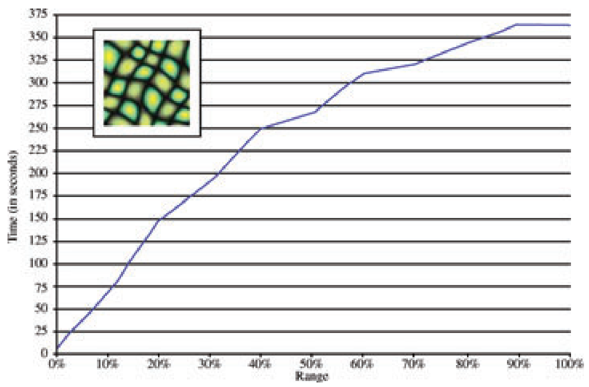

Figure 8: *Increasing the error tolerance when matching the direct neighbors, results in a richer search space. However, this search space is much larger, and thus results in slower synthesis timings. In this graph we plotted the increase in time versus the error tolerance for the source texture shown in the left-top when synthesizing a 400 × 400 texture.*

multiple times at different positions. This results in a large numberof causal neighbors that need to be verified each time a new pixel is synthesized.

In Figure 7 some less successful results are shown. The reduced search space is sometimes not diverse enough to ensure good synthesized results. In order to expand the search space for these cases, we extend our algorithm. Until now we constructed the search space by taking the union of the search spaces defined by the direct neighboring pixels. This implies that at least one of the direct neighbor's pixel values is exactly matched in each vector of the constructed search space. It is possible to relax this constraint of having at least one exact match by considering all partial search spaces for which the key (color) lies within some predetermined range or alternatively the n nearest colors to the exact key value. The rational is that as long as the pixel values are perceptually close in appearance, high-frequency discontinuities are avoided. A disadvantage is that the search space increases in size, and thus requires more time to be searched, resulting in a slower synthesis (see Figure 8). This extension can be easily incorporated in the original algorithm since we already used a kd-tree, which is a suitable data structure for doing range queries on.

To show the effect of using a reduced search space, we synthesized a texture with different error tolerances on the key value ranging from 0% (exact match) upto 100% (exhaustive search). A selection of the generated textures can be seen in Figure 9. For this particular example it is clear that the reduced search space (0% range) performs at least as good as an exhaustive search. This shows that although the diversity of the search space has been reduced, the results still look visually pleasing.

Finally, we investigated the effect of using a reduced search space on the 'randomness' of the generated textures by creating false color images that encode where each pixel in the generated texture originates from in the sample texture (Figure 10). Examples 10(a) and 10(b) were generated from a stochastic sample texture. The false color images clearly show that the generated textures consist of random irregularly shaped blocks from the sample texture. Examples 10(c) and 10(d) are generated from near-regular sample textures. As expected, the false color images show a regular pattern. Note that in Figure 10(c) the same tile is repeated in the synthesized texture. This is caused by the fact that the sample texture is an exact tiling of patterns. Since our method is deterministic, it will always copy the same pixel when multiple sample pixels have the same error. In Figure 10(d) another regular structured texture is shown, but not an exact tiling, resulting in some randomness in the color coding.

Some additional results can be found at the end of this paper. All of the results in this figure have been created by matching at least one of the direct neighbors directly. As discussed before, this can still lead to some discontinuities in the synthesized texture (e.g. bolt image and window image in Figure 13). Using a range search would solve this problem at the cost of prolonged synthesis timings.

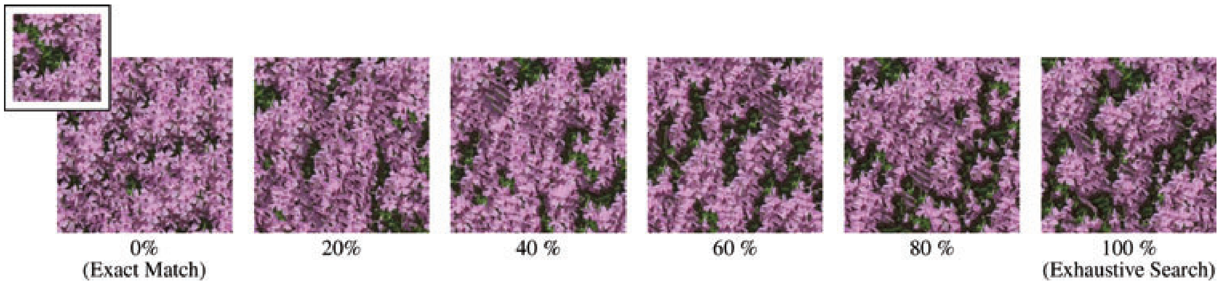

Figure 9: *A comparison of the synthesized results' visual quality with respect to an increase in error tolerance on matching the direct neighbors.*

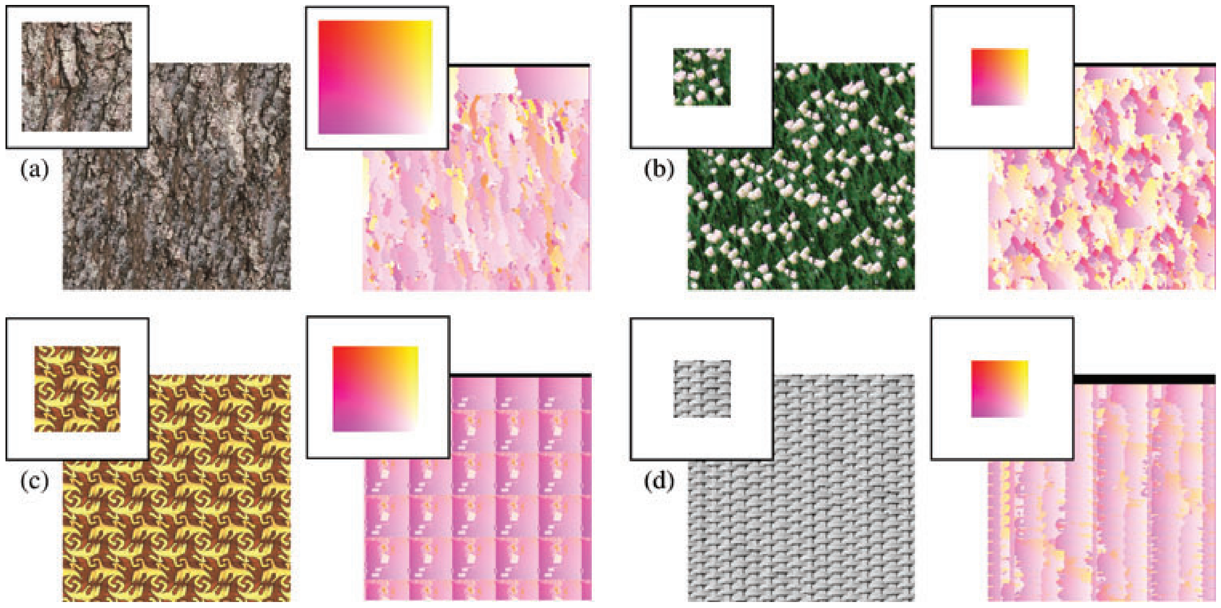

Figure 10: *Some synthesized textures together with false color images that indicate from where each pixel in the synthesized texture originates from in the source texture. The black region in each false color image indicates the part which was handled by the initialization process (section 4.3).*

6. Comparison

The presented technique shares some resemblance to [WL00], [Ash01], [ZG02] and [TZL*02]. As in the TSVQ technique of Wei and Levoy [WL00], our method reduces the search space in order to speed-up the synthesis process. However, with TSVQ a low-dimensional approximation of the complete search space is constructed, whereas our method uses highly detailed slices of the full search space. This implies that the diversity of our search space is less than the TSVQ search space, but has more detail. This surplus in detail allows smoother transitions in the search space from one point to another, resulting in more detailed and visually pleasing results. Ashikhmin [Ash01] also uses a reduced search space defined by the direct neighboring pixels. However, the search space used by Ashikhmin is much smaller (maximum 4 vectors), and is only able to generate good results for natural textures. Our method considers a larger search space and is able to handle a wider range of texture types. The method of Ashikhmin [Ash01] is further generalized in [TZL*02]. Instead of using the search space defined directly by the direct neighbors, an extra level of indirection is added; the *k* most resembling neighborhoods to the 'forward shifted' direct neighbor's causal neighborhoods are used. K-coherence reduces the search space to similar size as the presented method, but due to the extra indirection, can introduce high-frequency discontinuities (i.e. there is no guarantee that the *k* best matches of a forward shifted direct neighbor have a low error on direct neighbors). Finally, our methods bares resemblance to the jump map technique of Zelinka and Garland [ZG02]. Both methods reduce the run-time cost by precomputing a (partial) search space. However, the jump map works with precomputed probabilities and does not take in account the current state of the synthesized texture, whereas our method does.

A visual comparison between the reported results of [WL00] and [Ash01] are shown in Figure 11. Our method clearly preserves the overall structure of the sample textures better and outperforms either in terms of synthesis speed and visual quality. In Figure 12 we compared our pixel-based texture synthesis technique to the reported results of [LLX*01], [KSE*03] and [LLH04], which are all patch-based texture synthesis techniques. As can be seen, our method performs at least as good as the patch-based techniques on structured textures. Note, that our technique can also handle textures of a stochastic nature, whereas patch-based techniques usually cannot.

7. Conclusion and Future Work

We presented a pixel-based texture synthesis algorithm that is able to create high-quality textures very fast. The key to our method is that at least one of the direct neighboring pixels is forced to match within a controlled error tolerance. By forcing such a match a large number of undesirable matches (i.e. neighborhoods with a low L^2-error contribution on distant pixels and a high L^2-error contribution on nearby pixels) are removed from the search space, avoiding undesirable matches that can lead to cuts and discontinuities in the synthesized texture.

Since a large number of causal neighborhoods are a priori ignored, a synthesis speed-up is achieved. If we have N pixels and C different colors in the sample texture, then the reduced search space size is on average $4N/C$ large. Thus, if C is large, then the average size of the reduced search space is small and thus very few causal neighborhoods need to be compared. As a result, a good matching neighborhood is found almost immediately. Even if the number of different colors is low (e.g. 100) then our method is still significantly faster than an exhaustive search.

For future work we would like to further improve the result and synthesis speed by exploiting the fact that large irregular blocks are copied. Unlike patch-based techniques, we would like to impose no restrictions on the shape of these blocks. Currently, our system requires two user-determined parameters: the causal neighborhood size and the maximum error range tolerated when doing a nearest neighbor query in the kd-tree. Ideally, we would like the system to propose a 'good' initial guess, which the user can refine if desired. Finally, we would like to incorporate multiresolution synthesis. Initial experiments yield encouraging results (Figure 13).

Acknowledgments

The first author is supported by a grant from the Belgian Technical Cooperation (BTC) and the second author is supported by Katholieke Universiteit Leuven grant GOA/2004/05.

References

[Ash01] Ashikhmin M.: Synthesizing natural textures. In *SI3D '01*: *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM Press (2001), 217–226.

[BPR03] Battiato S., Pulvirenti A., Reforgiato D.: Antipole clustering for fast texture synthesis. In Proceedings of the 11th WSCG 2003, Plzel-Bory, Czech Republic (2003), 97–105.

[CSH*03] Cohen M. F., Shade J., Hiller S., Deussen O.: Wang tiles for image and texture generation. *ACM Transactions on Graphics 22*, 3 (2003), 287–294.

[DCY03] Drori I., Cohen-or D., Yeshurun H.: Fragment-based image completion. *ACM Transactions on Graphics 22*, 3 (2003), 303–312.

[Deb97] Debonet J. S.: Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. (1997), 361–368.

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press (2001), 341–346.

[EL99] Efros A., Leung T.: Texture synthesis by nonparametric sampling. In *International Conference on Computer Vision* (1999), 1033–1038.

[EMP*02] Ebert D., Musgrave F., Peachey D., S. Perlin Worley: *Texture and Modeling*, *A Procedural Approach*. Morgan Kaufmann, 2002.

[HB95] Heeger D. J., Bergen J. R.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM Press (1995), 229–238.

[HJO*01] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press (2001), 327–340.

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3 (2003), 277–286.

[LLH04] LIU Y., LIN W., HAYS J.: Near-regular texture analysis and manipulation. In *SIGGRAPH Proceedings of the annual conference on Computer graphics and interactive techniques*. ACM Press (2004), 368–376.

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics 20*, 3 (2001), 127–150.

[NA03] NEALEN A., ALEXA M.: Hybrid texture synthesis. In *Eurographics Symposium on Rendering 2003*. Leuven, Belgium, 2003, 97–105.

[NA04] NEALEN A., ALEXA M.: Fast and high quality overlap repair for patch-based texture synthesis. In *Computer Graphics International* (2004), IEEE Computer Society (2004), 582–585.

[NMM*05] NICOLL A., MESETH J., MÜLLER G., KLEIN R.: Fractional Fourier texture masks: Guiding near-regular texture synthesis. In *Eurographics '05: Proceedings of the Eurographics 2005*, Eurographics Association, 2005.

[Per85] PERLIN K.: An synthesizer. image In *SIGGRAPH '85*: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. ACM Press (1985), 287–296.

[PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. (2000), 465–470.

[PL98] PAGET R., LONGSTAFF I.: Texture synthesis via a noncausal nonparametric multiscale Markov random field. In *IEEE Transactions on Image Processing 7*, 6, IEEE (1998), 925–931.

[PS00] PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 40*, 1 (2000), 49–70.

[Tur01] TURK G.: Texture synthesis on surfaces. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press (2001), 347–354.

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press (2002), 665–672.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. (2000), 479–488.

[Wor96] WORLEY S.: A cellular texture basis function. In *SIGGRAPH '96*: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM Press (1996), 291–294.

[XGS00] XU Y. Q., GUO B., SHUM H.: Chaos mosaic: Fast and memory efficient texture synthesis. In *Tech. Rep. MSR-TR-2000-32, Microsoft Research* (2000).

[ZG02] ZELINKA S., GARLAND M.: Towards real-time texture synthesis with the jump map. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association (2002), 99–104.

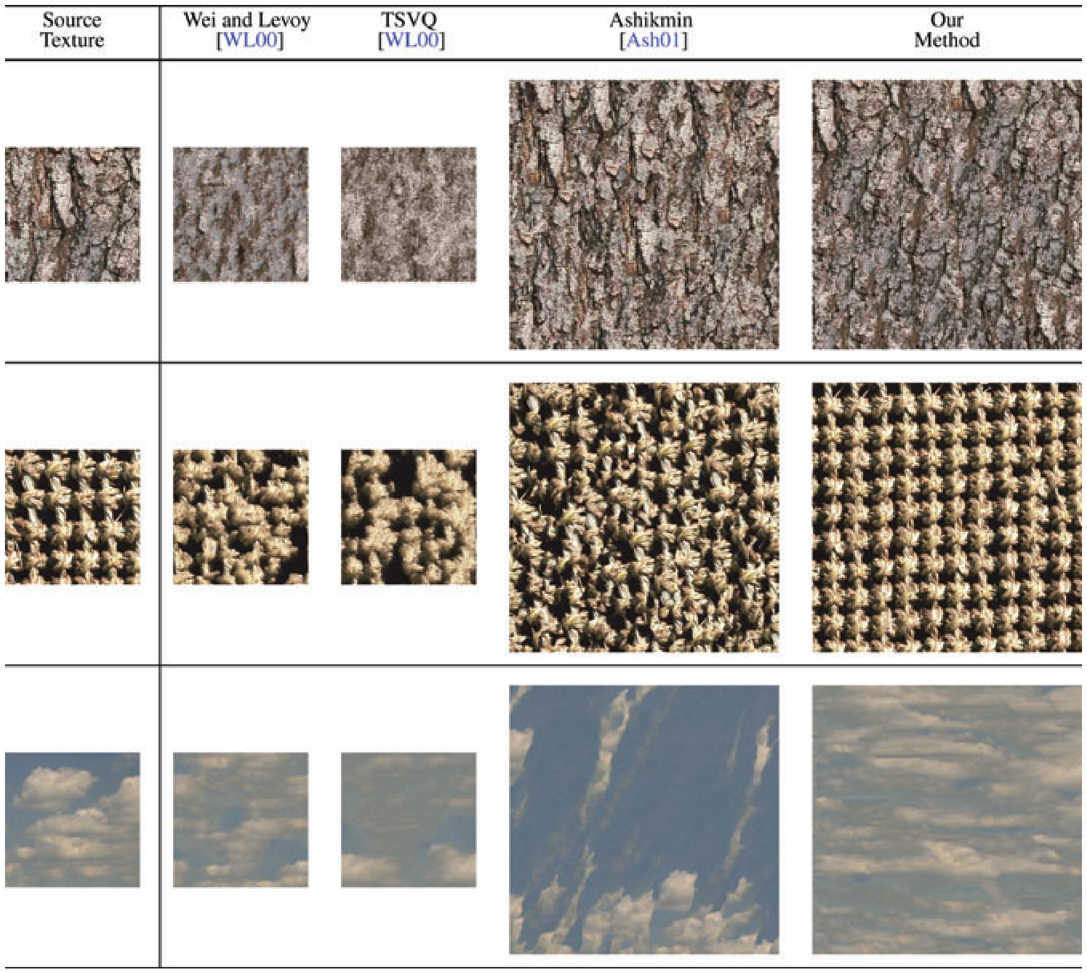

Figure 11: *A comparison between the reported results of [WL00] and [Ash01] and the presented technique.*

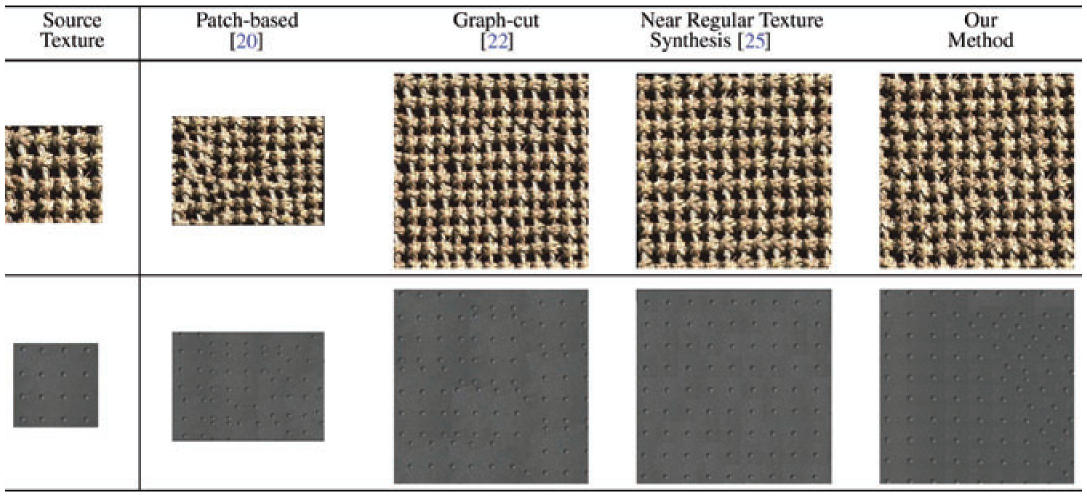

Figure 12: *A comparison of our pixel-based technique with the reported results of some patch-based techniques.*

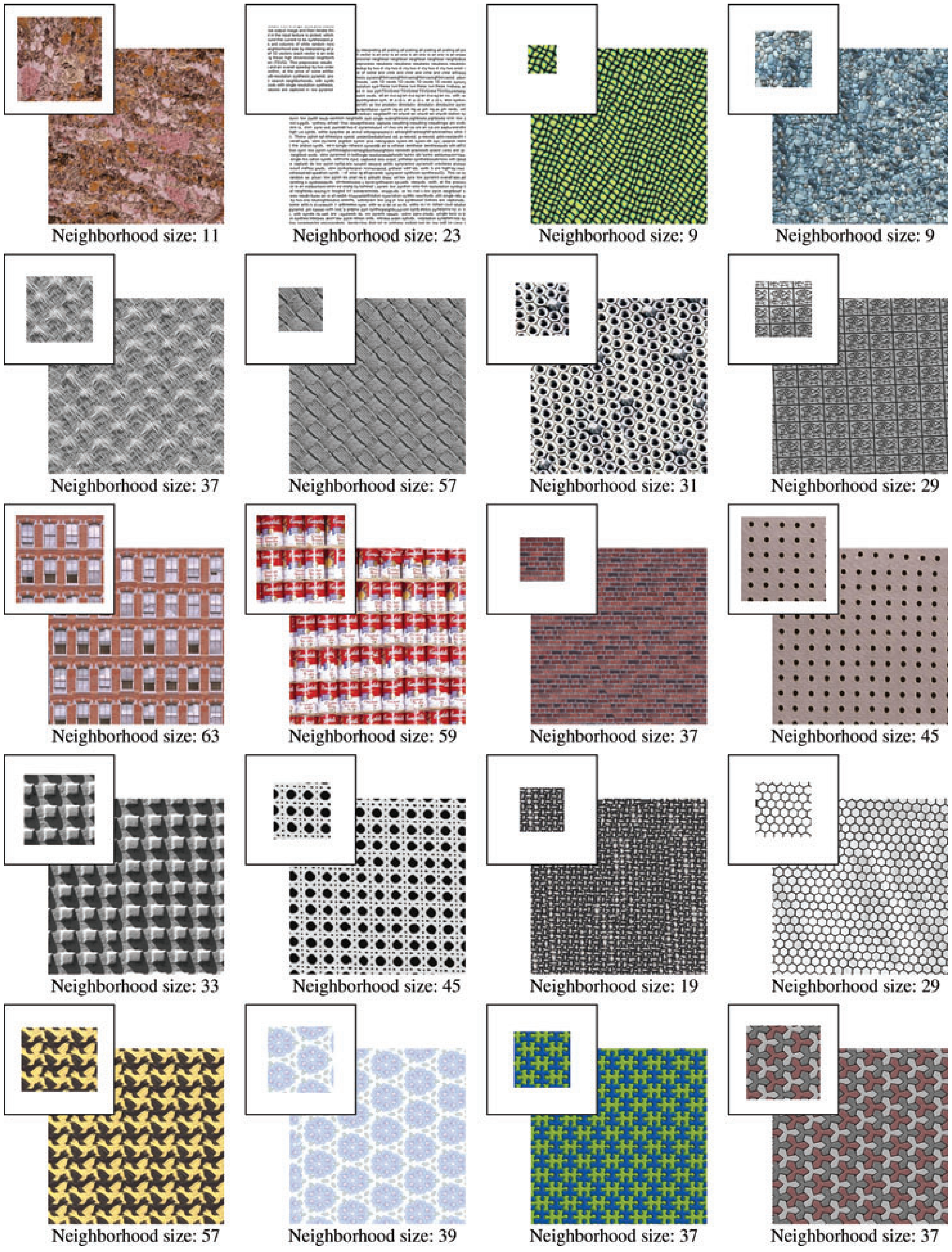

Figure 13: *Some more results generated with the presented method.*