EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer (Guest Editors)

Volume 34 (*2015*), *Number 2*

Self Tuning Texture Optimization

Alexandre Kaspar[1], Boris Neubert[1], Dani Lischinski[2], Mark Pauly[1], Johannes Kopf[3]

[1]École Polytechnique Fédérale de Lausanne
[2]The Hebrew University of Jerusalem, [3]Microsoft Research

Abstract

The goal of example-based texture synthesis methods is to generate arbitrarily large textures from limited exemplars in order to fit the exact dimensions and resolution required for a specific modeling task. The challenge is to faithfully capture all of the visual characteristics of the exemplar texture, without introducing obvious repetitions or unnatural looking visual elements. While existing non-parametric synthesis methods have made remarkable progress towards this goal, most such methods have been demonstrated only on relatively low-resolution exemplars. Real-world high resolution textures often contain texture details at multiple scales, which these methods have difficulty reproducing faithfully. In this work, we present a new general-purpose and fully automatic self-tuning *non-parametric texture synthesis method that extends Texture Optimization by introducing several key improvements that result in superior synthesis ability. Our method is able to self-tune its various parameters and weights and focuses on addressing three challenging aspects of texture synthesis: (i) irregular large scale structures are faithfully reproduced through the use of automatically generated and weighted guidance channels; (ii) repetition and smoothing of texture patches is avoided by new* spatial uniformity constraints*; (iii) a* smart initialization *strategy is used in order to improve the synthesis of regular and near-regular textures, without affecting textures that do not exhibit regularities. We demonstrate the versatility and robustness of our completely automatic approach on a variety of challenging high-resolution texture exemplars.*

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

1 Introduction

Texture mapping is a fundamental technique in computer graphics for augmenting coarse geometric models with fine surface details, thus enabling the generation of photorealistic imagery far more efficiently than can be done with geometry alone. It can be difficult, however, to acquire a seamless texture that fits the exact dimensions and resolution required for a specific modeling task. Thus, *example-based texture synthesis* methods, which allow generating arbitrarily large textures from limited exemplars have been an area of intensive research in the past two decades.

The fundamental goal of example-based texture synthesis is to generate a texture that faithfully captures all the *visual* characteristics of the exemplar, yet is neither identical to it, nor exhibits obvious repetitions or other unnatural looking artifacts.

Previous methods have made remarkable progress towards this goal. The most successful methods to date are based on non-parametric Markov Random Field formulations [WLKT09], most importantly stitching-based approaches [KSE*03] and texture optimization [KEBK05, WSI07, DSB*12].

Most previous methods have been demonstrated only on low resolution exemplars (e.g., up to 128^2 pixels). However, hi-fidelity computer graphics imagery typically requires significantly finer resolutions, as evidenced by the textures in many commercial repositories (e.g., CGTextures[†] or TextureKing[‡]). These high resolution exemplars are challenging for existing algorithms because they often contain multi-scale texture details, e.g., a rock texture might exhibit macro scale structures such as large cracks on top of micro scale surface details (Figure 1, left). Because state-of-the-art approaches generate a single scale at a time while using small causal windows, they often have difficulties in preserving large scale and near-regular features, and also suffer from

[†] http://www.cgtextures.com/
[‡] http://www.textureking.com/

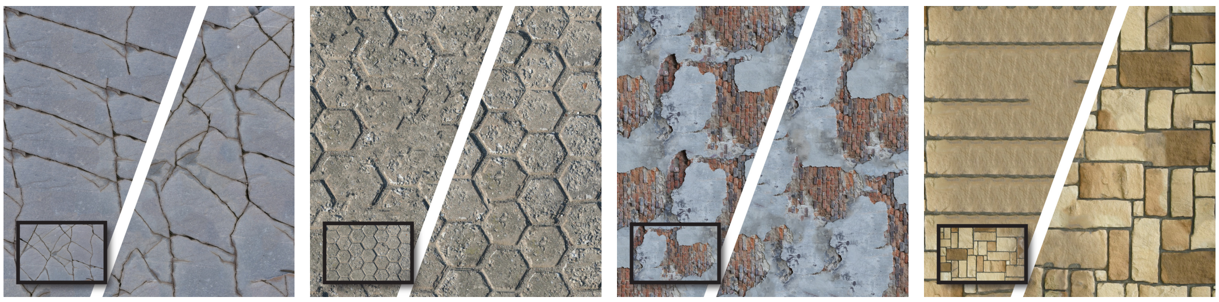


Image Quilting / Our Result
Large-scale Structures
Resynthesizer / Our Result
Near-regular Structures
Graphcut Textures / Our Result
Repetitions
Image Melding / Our Result
Smoothing

Figure 1: Existing texture synthesis algorithms have problems dealing with certain large-scale and near-regular structures, and sometimes suffer from repetition and smoothing artifacts. We present a new general and fully automatic algorithm that alleviates all these problems.

repetitions and smoothing. For example, texture optimization methods often converge to solutions where a smooth patch is repeated over and over (e.g., rightmost example in Figure 1).

In this paper we present a new general-purpose and fully automatic *self-tuning* texture synthesis method that extends Texture Optimization [KSE*03, DSB*12], by introducing several key improvements that, for the most part, overcome these problems. Our improvements result from a careful analysis of the shortcomings of previous methods, in particular with respect to the required manual adjustment of parameters on a case-by-case basis, or the need for other user input. In contrast, we pay particular attention to making the synthesis process fully automatic and suitable for a wide variety of high-resolution texture exemplars. This requires our method to be able to self-tune its various parameters and weights. Specifically, our method addresses the following three challenging aspects of texture synthesis:

Large-scale structures: One way to alleviate this problem is through the use of *guidance channels* (also known as *feature channels*) that encode non-local information explicitly derived from large-scale structures [ZZV*03, WY04, LH06].

However, in most methods the guidance channels are user-provided. For example, Zhang et al. [ZZV*03] require the user to provide a texton map, while Lefebvre and Hoppe [LH06] rely on user-drawn binary feature maps, which can be tedious and stands in the way of fully automatic texture synthesis. Although a few methods use color segmentation or edge detection to automatically generate a feature map, we are not aware of any generally applicable recipe for generating a guidance channel for a given exemplar. In addition, the common way of using such channels only helps to preserve the *connectedness* of features, but does not improve their *spatial distribution*; the results might still contain repetitions and overly smooth regions.

In our method, guidance channels are computed fully *automatically* through a combination of contour detection, thresholding, and distance transform (Section 3.1). We automatically determine a balanced weighting of the guidance channel, by analyzing the texture's color distribution. We enforce the exemplar's guidance channel histogram onto the synthesized texture to preserve the spatial distribution of features. We show that this automatic strategy is highly successful for all kinds of irregular large-scale structures, ranging from cellular (e.g., bricks) to line networks (e.g., cracks).

Repetitions and smoothing: Most existing methods ensure local similarity between the synthesized texture and the exemplar, but neither prevent repetitions, nor guarantee global appearance similarity. A notable exception is the Bidirectional Similarity algorithm [SCSI08, WHZ*08], which effectively ensures that every part of the exemplar is present in the synthesized result. However, even with this approach, excessive repetitions and smoothing can still occur. We also show that the particular way in which bidirectional similarity is implemented can sometimes stand in the way of successfully synthesizing structured textures.

We present a new *spatial uniformity constraint* that encourages uniformly sampling all locations of the exemplar (Section 3.2). It keeps track of and constrains the number of occurrences of each exemplar pixel, thereby directly preventing excessive repetitions and smoothing artifacts from densely tiling low energy patches. It is also less computationally intensive than bidirectional similarity.

Near-regular structures: The synthesis process is typically initialized randomly. This can impede successful synthesis of near-regular structures, because achieving a near-regular configuration requires global coordination, while most algorithms make only local updates to the texture. Previous works [DLTD08, RHDG10, MWT11] have shown that a more sophisticated initialization strategy can lead to improved synthesis results. We follow this route and propose a new initialization strategy that bootstraps the synthesis using random blocks from the exemplar, whose extents and offsets are determined from a self-similarity analysis of the exemplar [HS12, KLF12]. This *smart initialization* strategy (described in Section 3.3) improves the synthesis of regular and near-regular textures, without affecting textures that do not exhibit regularity.

Together these improvements result in a method that significantly outperforms previous state-of-the-art algorithms on a wide variety of high-resolution exemplars, including regular, near-regular, stochastic, and structured (small and large scale features, cellular and curvilinear features). Note that all parameters involved have either fixed settings or are self-tuning based on automatic texture analysis, so we were able to process a large number of textures from all these categories without any user interaction. The full test set is provided on our project page.

To make our algorithm fully reproducible we provide our full implementation and data at this url:
`http://w-x.ch/pub/self-tuning-texopt`.

2 Previous Work

2.1 Texture Synthesis

A complete review of example-based texture synthesis methods is outside the scope of this paper, and we refer the reader to the excellent survey by Wei et al. [WLKT09]. Specifically, our method falls into the class of non-parametric texture synthesis methods, which include pixel-based methods [EL99, WL00], stitching-based methods [EF01, KSE*03, LL12], optimization-based methods [KEBK05, HZW*06, WSI07], and appearance-space texture synthesis [LH06]. Pixel- and stitching-based methods are able to reproduce a wide variety of textures by growing the texture one pixel or a texture patch at a time. Optimization-based methods evolve the texture as a whole, by minimizing a suitably defined objective function. This typically improves the quality of the results and also makes the synthesis more controllable.

In recent work, Darabi et al. [DSB*12] present *image melding*: a synthesis method that unifies and generalizes patch-based synthesis and texture optimization, leveraging PatchMatch, a fast randomized patch search algorithm [BSFG09]. They have been able to demonstrate state-of-the-art results in a variety of image editing and synthesis applications, including example-based texture synthesis and texture interpolation. Our texture synthesis method builds on image melding, but as mentioned earlier, we introduce a number of important enhancements that enable us to significantly outperform the state-of-the-art on a wide variety of exemplars.

2.2 Guidance Channels

Non-parametric MRF-based methods, such as the ones listed above, often encounter difficulties when attempting to synthesize textures that feature large scale structures, such as the network of cracks shown in Figure 1. It is often possible to cope with such textures by guiding the synthesis process using extra channels, typically referred to as *label maps* or *feature channels*, which encode the large scale variations and the non-local features present in the exemplar (see, e.g., [HJO*01, ZZV*03, WY04, LH06, RCOL09, WWY14]).

Hertzmann et al. [HJO*01] describe *texture-by-numbers* synthesis: the input exemplar is augmented with a discrete label map, where regions with distinct texture are assigned different labels. A target label map is then also provided for the output and used to drive the synthesis. Zhang et al. [ZZV*03] synthesize progressively-variant textures by augmenting the exemplar with user-provided texton mask, orientation field, and a transition function. Wu and Yu [WY04] match and align curvilinear features across patch boundaries. They only consider easy-to-detect features, such as strong edges and ridges that remain after bilateral filtering, and alignment is achieved by deforming the texture.

Lefebvre and Hoppe [LH06] also use a guidance channel: a distance transform of a user-provided binary feature mask. They use PCA to transform the exemplar texture along with any additional channels into a new exemplar whose points reside in a low-dimensional appearance space and encode non-local information. However, using PCA is not effective for high resolution textures and/or large scale structures.

Rosenberger et al. [RCOL09] proposes an example-based 2D binary shape synthesis algorithm for synthesizing a control map (label map) for natural highly inhomogeneous textures. They assume that the exemplar control map may be obtained via color-based segmentation of the exemplar texture.

In summary, to our knowledge, all of the methods that use guidance channels rely on these channels being provided as input along with the exemplar, or available through a simple edge-detection or segmentation process. We are not aware of any method that produces such channels automatically for general exemplars, as we do in this work.

3 Algorithm

We start with a brief description of the Texture Optimization algorithm [KSE*03] that our contributions are based on. This algorithm optimizes an objective that measures how similar the synthesized texture T is to the exemplar S over a set of overlapping local patches. This objective takes the form

$$\min_{\{\mathbf{t}_i,\mathbf{s}_i\}} \sum_{i \in T} d\left(\mathbf{t}_i, \mathbf{s}_i\right), \tag{1}$$

where $\mathbf{t}_i$ refers to the square $N \times N$ patch in T with top-left pixel i, and $\mathbf{s}_i$ refers to an approximate nearest neighbor of $\mathbf{t}_i$ in the exemplar S. $N = 10$ in our implementation. Distances between patches are measured as the sum of squared color distances, i.e.,

$$d\left(\mathbf{t}_i, \mathbf{s}_i\right) = \left\|\mathbf{t}_i - \mathbf{s}_i\right\|_2^2. \tag{2}$$

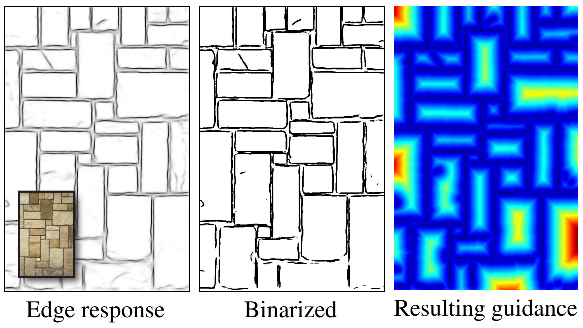


Figure 2: Computing the guidance channel from edge response maps.

This objective is optimized using an iterative strategy alternating between minimizing with respect to $\{\mathbf{t}_i\}$ or $\{\mathbf{s}_i\}$, while keeping the other set fixed. Minimizing $\{\mathbf{t}_i\}$ amounts to simply averaging the overlapping $\mathbf{s}_i$ assignments [WSI07]. Minimizing $\{\mathbf{s}_i\}$ requires finding the nearest neighbor for each synthesized patch $\mathbf{t}_i$, which we do using the PatchMatch algorithm [BSFG09]. We operate in the CIELAB color space and run the algorithm in a coarse-to-fine manner on a multi-resolution pyramid. The number of scales is computed by enforcing a dimension ratio of 1.3 between two consecutive scales and a coarsest dimension of at least 35 pixels. Please refer to the cited papers for further details.

Our implementation is based on the public source code of Image Melding [DSB*12] with several basic modifications to make the algorithm more suitable for texture synthesis. First, since texture exemplars are rectified and do not exhibit perspective distortions, we disabled searching among rotations and scale. Since they do not contain spatial lighting variation we disabled gain and bias adjustments. We also disable gradient channels and Poisson fusion, as these features do not improve the results but introduce slight blurring of details. Finally, we quantize the nearest neighbor field to integer locations, which avoids loss of sharpness from resampling the exemplar. Please note that these changes are only recommended for *texture* synthesis and not for *general scene* synthesis, e.g., image completion. All Image Melding results shown in this paper are generated with this modified version of the algorithm, while in the supplementary material we also compare against the unmodified version of the algorithm.

In the following sections we describe three major extensions to this algorithm that form our main contributions: Section 3.1 describes our *automatic computation of guidance channels*; Section 3.2 introduces our *spatial uniformity constraint*; and Section 3.3 discusses our *smart initialization* strategy.

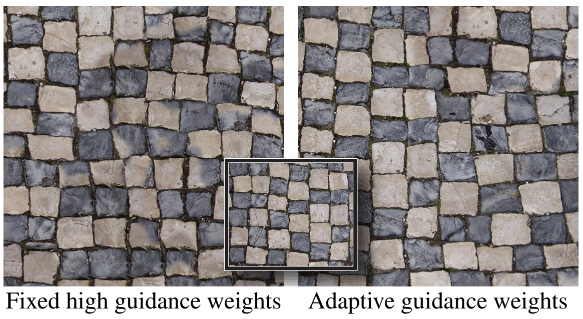


Figure 3: *Left:* assigning a fixed weight to the guidance channel can lead to poor results for textures that have different color distributions on both sides of the edge. In the example shown here tiles of different colors erroneously blend together because the relative contribution of the color channels is too low compared to the guidance channel. *Right:* our adaptive weighting alleviates this problem.

3.1 Automatic Guidance Channel

We improve our algorithm's ability to deal with large-scale structures by augmenting the color space with an additional guidance channel. It has been shown in previous work [LH06] that an effective strategy is to store the distance to the nearest "feature" (i.e., a visual contour) in the guidance channel. In most previous work guidance channels are provided by the user, however, we are interested in an automatic algorithm for computing the guidance channel using image processing methods.

We start by extracting large-scale features using a modern contour detector. Specifically, we use the Structured Edge detector [DZ13], which is currently the best performing detector on several established contour detection benchmarks. This method has been trained with hand-labeled natural images to detect *visually salient* edges, including color and texture discontinuities.

Our next goal is to compute the distance transform of the edge response map, however, a complication is that the response map contains continuous values. Therefore, we first binarize the map using Otsu's automatic thresholding method [Ots79]. Next, we compute the Euclidean distance transform using MATLAB's `bwdist` function. Since the thresholded features have some finite thickness we compute the distance transform both outside and inside the features, and use negative values for the inner distances. Finally, we normalize the resulting guidance channel so it takes the same scalar value range as the L channel. The entire process is illustrated in Figure 2.

As an alternative to thresholding we also considered the *generalized distance transform* [CSRP10], which can be directly applied to the continuous edge response map. However, that method requires hand tuning a parameter that balances between spatial and gray level distances. We could not find a reasonable default for this parameter.

The last remaining issue is how to balance the relative contribution of the guidance channel and the three color channels, when doing pixel comparisons. Assigning too strong a weight to the guidance channel can have a detrimental effect on the synthesis. This is in particular the case for textures where a different color distribution is present on each of the two sides of a feature. In this case a strongly weighted guidance channel lessens the algorithm's ability to distinguish the two different materials. Figures 3 (left) illustrates this problem.

We perform a simple test to detect this situation. First, we collect the pixel colors in the vicinity but not directly on the features, i.e. in a distance range $(0, \tau_{vic}]$, where τ_{vic} is set to 1% of the exemplar width. Next, we compute density maps for the *La*, *Lb*, *ab* slices of the color space. Finally, we check if at least one density map contains multiple peaks or at least half of its area is non-zero. If none of these conditions is true we consider it safe to use a strong guidance weight (three times as high as the color channels), otherwise we use a more conservative setting (same as the color channels). In Figure 3 we show an example of how adaptively downweighting the feature channel can improve the synthesis. The textures in Figure 7, on the other hand, all use a high weight setting. About 70% of the textures we tried use a high guidance channel weighting.

3.2 Global Appearance

Most previous methods optimize the local similarity between the result and exemplar, i.e. each local window in the result is made to look similar to some local window in the exemplar. This is not sufficient, however, to ensure a globally similar appearance, as evidenced by the results in Figure 4 (left), which do not preserve the richness of the exemplar.

The bidirectional similarity constraint [SCSI08] has been proposed to address this issue (Wei et al. [WHZ*08] describe a similar algorithm in a concurrent paper). The main idea of bidirectional similarity is to also require that each exemplar window is represented in the result. Using this constraint (Figure 4, middle) improves the synthesis, however, it is not enough to completely alleviate the problems. There are still excessive repetitions in the top row, and while in the second row the constraint enforced some synthesized structures, there are still many structural defects.

To improve on this situation, we propose a new strategy for more faithfully capturing global appearance. First, we present a new *spatial uniformity* constraint that encourages sampling all areas of the exemplar in equal amounts and directly prevents excessive repetitions by penalizing single pixels from the exemplar from occurring too often in the result (Figure 4, right-top). Second, we enforce the histogram constraint of Kopf et al. [KFCO*07] on all channels (color and guidance). Preserving the histogram of feature distances,

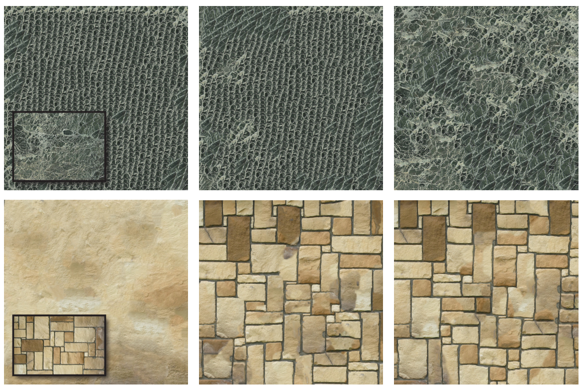

No completeness constraints
Bidirectional Similarity
Spatial uniformity + histogram constraints

Figure 4: Completeness enforcement. *Left:* only enforcing *unidirectional* similarity between result and exemplar is not sufficient to achieve good results. *Middle:* using a bidirectional similarity constraint improves, but there are still excessive repetitions and structural defects in the top and bottom rows, respectively. *Right:* Encouraging spatially uniform exemplar sampling and preserving the color histograms corrects both artifacts.

which are stored in the guidance channel, ensures a high *global* structural similarity between the exemplar and the result and fixes, for the most part, the problems with the bricks texture.

Spatial uniformity: Our goal is to prevent excessive repetitions of single elements from the exemplar in the result. To facilitate this task we keep track of an *occurrence map*

$$\Omega(x,y) = \left|\left\{\mathbf{s}_i \mid (x,y) \in \mathcal{N}(\mathbf{s}_i)\right\}\right|, \tag{3}$$

where $\mathcal{N}(\mathbf{s}_i)$ denotes the set of pixels in the patch $\mathbf{s}_i$. In other words, the occurrence map stores how many times each exemplar pixel is used in the patches that make up the result. In a perfectly balanced result each exemplar pixel would be used the same number of times,

$$\omega_{best} = \frac{|T|}{|S|} N^2, \tag{4}$$

i.e., proportionally to the relative areas of the result T and the exemplar S, and the number of pixels N^2 in each patch.

To encourage a uniform occurance map we modify the patch distance in Equation 2 to become

$$d(\mathbf{t}_i, \mathbf{s}_i) = \|\mathbf{t}_i - \mathbf{s}_i\|_2^2 + \lambda_{occ} \frac{\Omega(\mathbf{s}_i)}{\omega_{best}}, \tag{5}$$

where $\Omega(\mathbf{s}_i)$ refers to the average occurrence value of all pixels in a patch,

$$\Omega(\mathbf{s}_i) = \frac{\sum_{(x,y)\in \mathbf{s}_i} \Omega(x,y)}{N^2}. \tag{6}$$

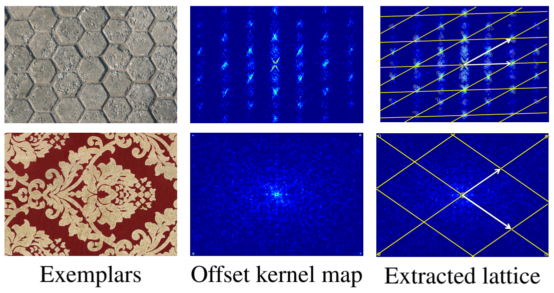


Figure 5: Lattice extraction examples. In the top row, the lattice corresponds to the hexagon bounds. The bottom row is more suble as the exemplar contains a tileable pattern. It exhibits a strong translational symmetry for the offset corresponding to the tile size (visible in the corners of the density map). Because the peaks offsets are above half the image size, we use a block size that is half the maximum offset.

This new term penalizes pixels that are used often. $\lambda_{occ} = 10$ controls the relative contribution of uniformity enforcement.

In addition to changing the patch distance we also modify the random search phase of the PatchMatch algorithm, to reject every new patch candidate that has an occurence above a threshold $\Omega(\mathbf{s}_i^{new}) > 2\omega_{best}$, unless the patch that is currently in its position has an even worse score (i.e., we have $\Omega(\mathbf{s}_i^{new}) < \Omega(\mathbf{s}_i^{cur})$ so that the patch would improve spatial uniformity).

Histogram enforcement: In addition to spatial uniformity, we also use the histogram matching constraint proposed by Kopf et al. [KFCO*07]. This constraint modifies weights in the "voting" stage of the PatchMatch algorithm. Instead of using uniform weights it gives higher weights to colors that bring the color channel histograms of the result texture closer to the exemplar texture.

This is particularly effective in conjunction with our guidance channels that include the distances to the nearest features, since preserving the histogram of the guidance channel has the effect of preserving the distribution and arrangement of features on a global scale. This is crucial for textures that have large-scale features, such as the bottom row in Figure 4 or the bottom row in Figure 7.

Stability: In practice, bidirectional similarity works well. However it renders the synthesis unstable by forcing new patches to appear in regions with non-matching colors, further leading to broken features (bottom row of Figure 4). While histogram enforcement suffers from a similar issue, our spatial uniformity term is more stable. Therefore we use the spatial uniformity at each scale, and only use the histogram enforcement for the first half of the scale pyramid (i.e., we allow feature changes only in the early stages of the synthesis).

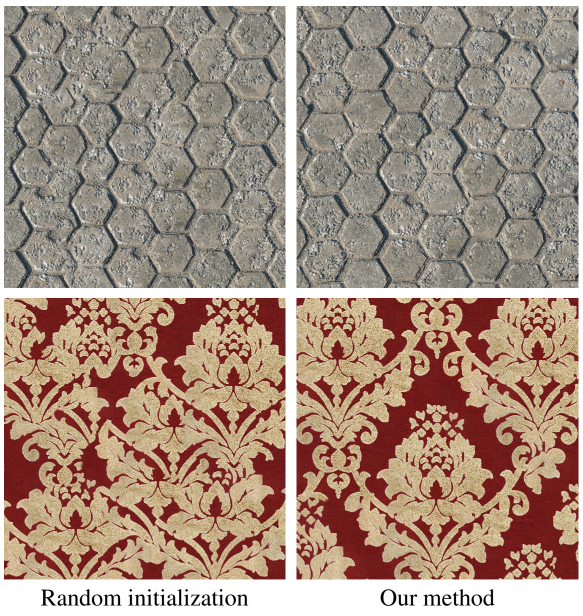


Figure 6: Semi-regular textures as well as textures including translational symmetries both benefit from our smart initialization method.

3.3 Initialization

Most previous algorithms initialize the target texture with random scrambling of exemplar pixels. While this ensures that the synthesis process starts with the exemplar's color distribution, it reflects none of the exemplar's global structures. Due to the local nature of the synthesis process, this actually reduces the chances that any regular or near-regular patterns will be correctly reproduced, because such patterns are of a highly global nature. Some more sophisticated techniques have been proposed in previous work.

Liu et al. [LLH04] rely on user assisted lattice extraction to assist in near-regular texture synthesis.

Dong et al. [DLTD08] propose a strategy for initializing *solid texture* optimization by essentially tiling a small exemplar neighborhood (see their paper for details). However, this might lead to repetition artifacts with high resolution textures.

Risser et al. [RHDG10] synthesize variations of a set of exemplars that are structurally compatible and have the same size. Their synthesis is initialized with at a coarse level with a jittered combination from several exemplars. Since the jitter is only slight they preserve global features, however, they cannot synthesize entirely new structures or go beyond the size of the exemplars.

Ma et al. [MWT11] describe a patch-based initialization strategy that works well for *discrete element* texture synthesis (i.e., the unit of synthesis are fully connected small particles). They cut the exemplar into patches of a user-provided size and randomly copy these patches into the output domain. In the context of standard 2D image synthesis, however, features in the exemplars are not annotated and might be of a global nature. Therefore, a simple random-patch initialization might lead to poor results with disconnected features.

We build on the aforementioned works and propose a new fully automatic initialization strategy that preserves a near-regular configuration if it is present in the exemplar. We first find the dominant *translational symmetry* in the exemplar, represented by a pair of linearly independent 2D translation vectors, also known as lattice generating vectors. Next, we initialize the target texture using larger blocks whose offset is a random *integer* multiple of the detected lattice generators. This yields a randomized initialization, which nevertheless preserves translational regularity in the exemplar. If the exemplar does not exhibit such regularity, the strategy has no adverse effect on the synthesis.

To find the best lattice generators we follow recent work on image completion [HS12, HKAK14] and consider patch offset statistics. We begin with detecting standard Difference of Gaussian feature points and compute the SIFT descriptors for each feature point [Low04]. Next, we compute the nearest neighbor for each feature using a kd-tree and retain only those matches whose L_2 feature distances are below half the mean distance. To account for noise and deal with slight variation in *near*-regular cases we treat the retained 2D offsets as probability distributions and compute a kernel density map $\phi(\vec{x})$ by splatting a Gaussian kernel ($\sigma = 3$) for every offset (Figure 5).

The two lattice generating vectors $\vec{g}_1, \vec{g}_2$ can now be obtained by maximizing the F_β-measure [VR79], a tool from statistical analysis that we use to find the optimal lattice considering both the precision P (do the lattice points coinside with density peaks?) and recall R (are we covering all peaks?). We maximize

$$F_\beta(\vec{g}_1,\vec{g}_2) = (1+\beta^2)\frac{P(\vec{g}_1,\vec{g}_2)\cdot R(\vec{g}_1,\vec{g}_2)}{\beta^2 P(\vec{g}_1,\vec{g}_2)+R(\vec{g}_1,\vec{g}_2)}, \tag{7}$$

where $\beta = 0.1$ balances between precision and recall. Let

$$X(\vec{g}_1,\vec{g}_2) = \{k_1\vec{g}_1 + k_2\vec{g}_2 \in B \mid k_1,k_2 \in \mathbb{Z}\} \tag{8}$$

denote the set of lattice points given two generators (within the bounding box of all offsets B). We define the precision P as the average density of the lattice points,

$$P(\vec{g}_1,\vec{g}_2) = \sum_{\vec{x}\in X(\vec{g}_1,\vec{g}_2)} \frac{\phi(\vec{x})}{|X(\vec{g}_1,\vec{g}_2)|}, \tag{9}$$

and the recall R as the fraction of total density we are covering with the lattice,

$$R(\vec{g}_1,\vec{g}_2) = \frac{\sum_{\vec{x}\in X(\vec{g}_1,\vec{g}_2)}\phi(\vec{x})}{\sum_{\vec{y}\in B}\phi(\vec{y})}. \tag{10}$$

Figure 5 visualizes some lattices detected in this manner.

Having found the lattice we first cut the exemplar into non-overlapping rectangular blocks whose size is equal to the bounding box of one lattice cell. We use these blocks to initialize our target texture by placing them successively in scanline order, and allowing every block to shift slightly to provide for better alignment (we maximize the cross correlation in a small overlap area). Note that we perform all these computations at full resolution but then downscale the resulting initialization to the coarse scale that the synthesis begins with.

Figure 6 shows the results of synthesizing a near-regular texture (top) and a texture with translational symmetry (bottom) without and with our novel strategy. Note that performing this initialization does not have a significant effect on the synthesis of textures that do not exhibit any regularity. In fact, for such textures, the recall term dominates Equation 7 with small $\vec{g}_1$ and $\vec{g}_2$, and the initialization becomes similar to random patch scrambling. All textures we show, regular or not, are produced with this strategy.

4 Results

We implemented our algorithm in MATLAB with C++ Mex files for time critical sections. The source code for our implementation is available online[§]. The performance of our algorithm depends on the exemplar and on the size of the synthesized texture. The following table summarizes performance statistics gathered from the 33 exemplars in our supplementary material for a target texture of size 1024^2. They were computed on a 2.93GHz Intel Core i7 870. Our implementation runs single-threaded.

	Median	Average	Min	Max
Smart initialization	**233s**	312s	119s	898s
Guidance channel	**6s**	6s	2s	10s
Texture optimization	**386s**	438s	204s	1019s
Total	**625s**	756s	340s	1927s

All parameters of our algorithm are self-tuning or have fixed default values that are provided in the paper. All textures shown anywhere in the paper or supplementary material are computed with these same fixed parameters (except where noted). The pseudo-random number generator used in the algorithm can be initialized with different seed values to generate several variations of a texture. In the supplementary material we provide a page with results generated for three different seed values to demonstrate the consistent quality of our results. All other textures shown in the paper and supplementary material are generated with the default seed value (i.e., we didn't cherry pick).

We applied our algorithm to a wide variety of textures including regular, near-regular, and irregular ones; textures that exhibit structures at various spatial scales, cellular textures, textures with linear features, and stochastic textures.

§ http://w-x.ch/pub/self-tuning-texopt

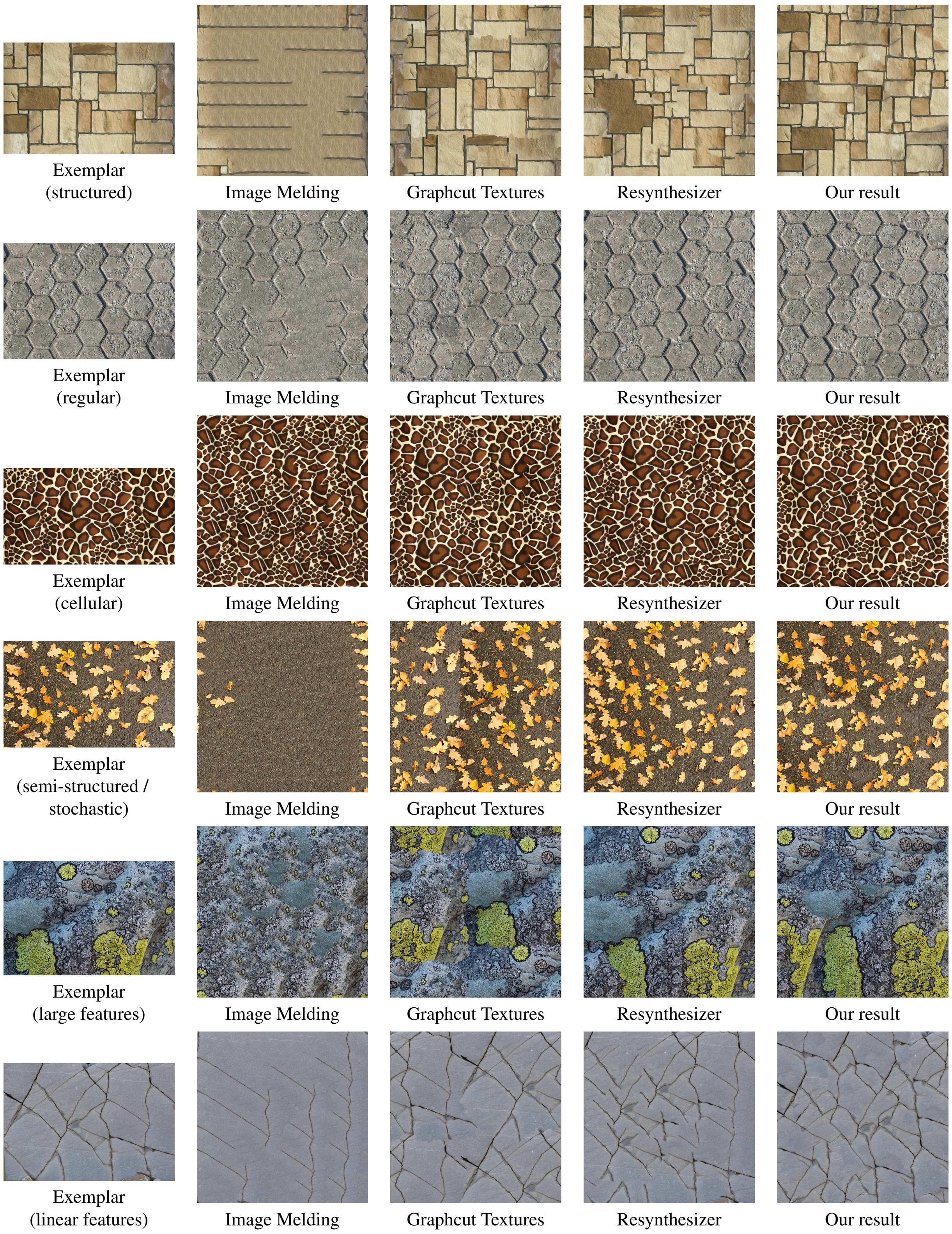


Figure 7: Representative synthesis results for various texture categories. For Image Melding we disabled searching in rotation and scale space. Please zoom into the PDF to see details. In the supplementary material we compare against more methods on a wider set of exemplars.

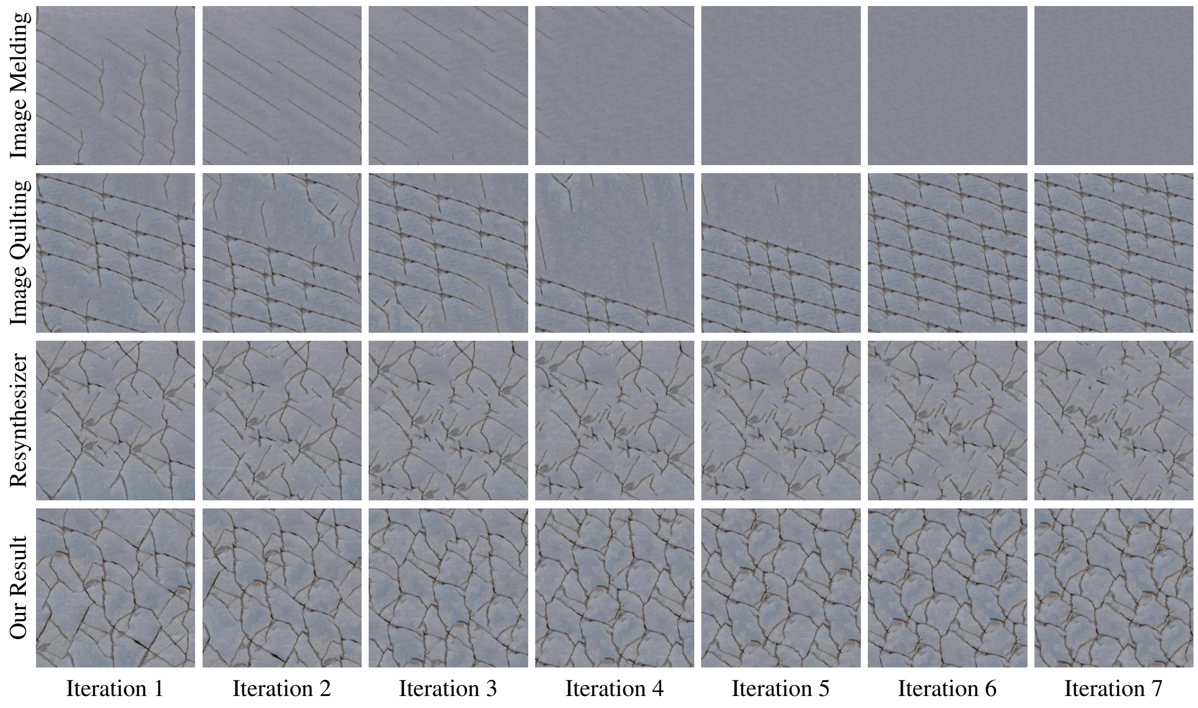


Figure 8: Stress testing texture synthesis algorithms with "texture sequences". Each iteration uses the synthesized output of the previous iteration as its input exemplar. In the supplementary material we show sequences for 33 different textures.

All these textures are "real-world" high-resolution textures downloaded from CGTextures[†]. Since there are no good quantitative metrics for evaluating the results of texture synthesis, we provide extensive comparative results in the supplementary material. Figure 7 shows some representative examples.

We compare our method against several state-of-the-art algorithms. In Figure 7 we show comparisons against Image Melding (our modified version, see Section 3), and Graphcut Textures [KSE*03] (using the version implemented in the GIMP Texturizer plugin[¶]). In the supplementary material we also show comparisons against the original Image Melding implementation, Image Quilting [EF01], and GIMP Resynthesizer[‖].

We also performed a new kind of analysis we call "texture sequences", where we run a synthesis algorithm iteratively, using the output of one iteration as the input exemplar for the next one. This provides an interesting "stress test" for the ability of a texture synthesis algorithm to preserve texture statistics and structures, generate variations, as well as demonstrate how quickly errors accumulate. We show one example of a texture sequence in Figure 8. In the supplementary material we provide sequence results for all 33 other exemplars we have tried. These texture sequences reveal that our method succeeds in preserving the original exemplar's appearance for more iterations. For example, features stay connected, and their overall spatial distribution is better preserved. Of course, eventually, deviations in appearance creep into our results as well, but this happens after a significantly larger number of iterations.

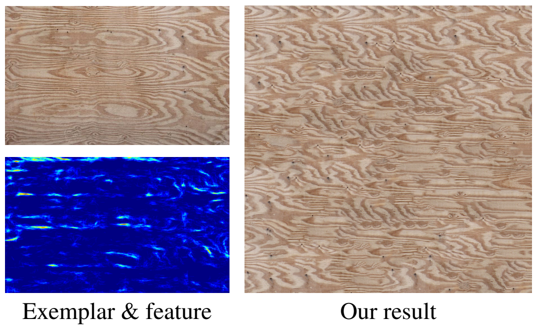


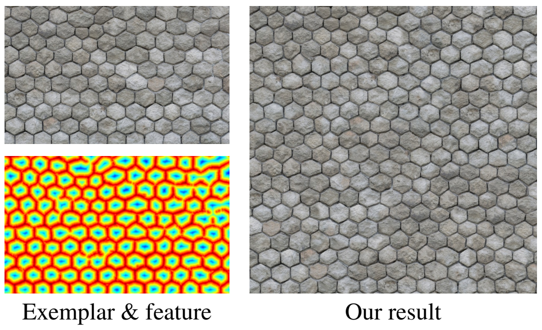
Figure 9: Limitation: contour detection can fail with textures containing large but unpronounced features

5 Limitations and Conclusions

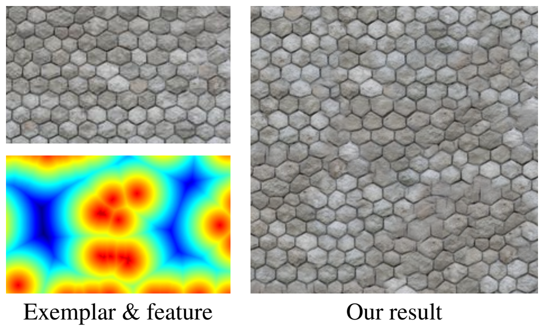
We have presented a new automatic optimization-based texture synthesis method that makes key improvements to sev-

[¶] http://gimp-texturize.sourceforge.net

[‖] http://www.logarithmic.net/pfh/resynthesizer



(a) High resolution: 907 × 589 exemplar, 1024 × 1024 target



(b) Low resolution: 256 × 170 exemplar, 1024 × 1024 target. The result is cropped to compare with the high resolution version.

Figure 10: Limitation: contour detection can fail with low resolution exemplars that exhibit dense structures.

eral problems that are common to many previous algorithms. We increased the ability to successfully synthesize textures with large-scale structures with *automatically* computed guidance channel. We proposed a new strategy for enforcing the global appearance of the exemplar onto the synthesized texture through a new formulation that directly penalizes excessive repetitions and by enforcing the global distribution of large-scale features using histogram matching. Finally, we improved the ability to handle regular and near-regular textures using a new smart initialization strategy that is randomized, but preserves the regularity of the exemplar, if it is present. Each of these techniques is carefully designed to be suitable for general-purpose texture synthesis. All parameters are self-tuning or have fixed default values, so we were able to process a wide variety of different textures with the same parameters.

Even though these new contributions together significantly improve the state-of-the art in textures synthesis there is still considerable room for improvement. Many of our results still contain small artifacts such as broken structures etc. that may be examined by carefully inspecting our results. In particular, our algorithm does not always perform fully satisfactory with multi-class cellular textures (e.g. the white and gray tiles in Figure 3). Then, while the chosen contour detector [DZ13] performs very well with our original exemplars, it fails with textures that contain large but unpronounced features as in Figure 9. We also found that it does badly with low-resolution textures (especially downsampled versions of our exemplars as shown in Figure 10). This might be alleviated with a better edge model using training data at various resolutions instead of the default trained model. As for our smart initialization, it is currently limited to a single layer of translational symmetry, although it could principally be extended to detect and handle a wider range of configurations. Finally, our algorithm is potentially more difficult to parallelize than other synthesis algorithms, as the bookkeeping for the spatial uniformity constraint can lead to race conditions.

We believe the before mentioned limitations provide interesting avenues for future work, and our texture sequences could be a basis for investigating synthesis quality which is of importance given the lack of meaningful metrics in texture synthesis.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 257453: COSYM.

References

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D.: PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009) 28*, 3 (2009), Article no. 24. 3, 4

[CSRP10] Criminisi A., Sharp T., Rother C., P'erez P.: Geodesic image and video editing. *ACM Transactions on Graphics 29*, 5 (2010), article no. 134. 4

[DLTD08] Dong Y., Lefebvre S., Tong X., Drettakis G.: Lazy solid texture synthesis. In *Proceedings of the Nineteenth Eurographics Conference on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2008), EGSR '08, Eurographics Association, pp. 1165–1174. 2, 6

[DSB*12] Darabi S., Shechtman E., Barnes C., Goldman D. B., Sen P.: Image Melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012) 31*, 4 (2012). 1, 2, 3, 4

[DZ13] Dollár P., Zitnick C. L.: Structured forests for fast edge detection. In *ICCV* (2013). 4, 10

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH '01* (2001), 341–346. 3, 9

[EL99] Efros A. A., Leung T. K.: Texture synthesis by non-parametric sampling. *Proceedings of ICCV '99 2* (1999), 1033–1038. 3

[HJO*01] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. *Proceedings of SIGGRAPH 2001* (2001), 327–340. 3

[HKAK14] HUANG J.-B., KANG S. B., AHUJA N., KOPF J.: Image completion using planar structure guidance. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014) 33*, 4 (2014), to appear. 7

[HS12] HE K., SUN J.: Statistics of patch offsets for image completion. *European Conference on Computer Vision (ECCV) 2012* (2012). 3, 7

[HZW*06] HAN J., ZHOU K., WEI L.-Y., GONG M., BAO H., ZHANG X., GUO B.: Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer 22*, 9-11 (2006), 918–925. 3

[KEBK05] KWATRA V., ESSA I., BOBICK A., KWATRA N.: Texture optimization for example-based synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005) 24*, 3 (2005), 795–802. 1, 3

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26*, 3 (2007), Article no. 2. 5, 6

[KLF12] KIM V. G., LIPMAN Y., FUNKHOUSER T.: Symmetry-guided texture synthesis and manipulation. *ACM Trans. Graph. 31*, 3 (June 2012), 22:1–22:14. 3

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM SIGGRAPH 2003 Papers 22*, 3 (2003), 277–286. 1, 2, 3, 9

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006) 25*, 3 (2006), 541–548. 2, 3, 4

[LL12] LASRAM A., LEFEBVRE S.: Parallel patch-based texture synthesis. *ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics (HPG) 2012* (2012), 115–124. 3

[LLH04] LIU Y., LIN W.-C., HAYS J. H.: Near-regular texture analysis and manipulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004) 23*, 3 (2004). 6

[Low04] LOWE D. G.: Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60*, 2 (Nov. 2004), 91–110. 7

[MWT11] MA C., WEI L.-Y., TONG X.: Discrete element textures. *ACM Trans. Graph. 30*, 4 (July 2011), 62:1–62:10. 2, 6

[Ots79] OTSU N.: A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on 9*, 1 (Jan 1979), 62–66. 4

[RCOL09] ROSENBERGER A., COHEN-OR D., LISCHINSKI D.: Layered shape synthesis: automatic generation of control maps for non-stationary textures. *ACM Trans. Graph. 28*, 5 (2009), Article 107. 3

[RHDG10] RISSER E., HAN C., DAHYOT R., GRINSPUN E.: Synthesizing structured image hybrids. In *ACM SIGGRAPH 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 85:1–85:6. 2, 6

[SCSI08] SIMAKOV D., CASPI Y., SHECHTMAN E., IRANI M.: Summarizing visual data using bidirectional similarity. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2013* (2008). 2, 5

[VR79] VAN RIJSBERGEN C.: *Information retrieval*. Butterworths, 1979. 7

[WHZ*08] WEI L.-Y., HAN J., ZHOU K., BAO H., GUO B., SHUM H.-Y.: Inverse texture synthesis. In *ACM SIGGRAPH 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 52:1–52:9. 2, 5

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000* (2000), 479–488. 3

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. *Eurographics 2009, State of the Art Report* (2009). 1, 3

[WSI07] WEXLER Y., SHECHTMAN E., IRANI M.: Space-time completion of video. *Transactions on Pattern Analysis and Machine Intelligence (PAMI) 29*, 3 (2007), 463–476. 1, 3, 4

[WWY14] WU R., WANG W., YU Y.: Optimized synthesis of art patterns and layered textures. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (Mar. 2014), 436–446. 3

[WY04] WU Q., YU Y.: Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004) 23*, 3 (2004), 364–367. 2, 3

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.-Y.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003) 22*, 3 (2003), 295–302. 2, 3