Symmetry-Guided Texture Synthesis and Manipulation

VLADIMIR G. KIM, YARON LIPMAN, and THOMAS FUNKHOUSER
Princeton University

This article presents a framework for symmetry-guided texture synthesis and processing. It is motivated by the long-standing problem of how to optimize, transfer, and control the spatial patterns in textures. The key idea is that symmetry representations that measure autocorrelations with respect to all transformations of a group are a natural way to describe spatial patterns in many real-world textures. To leverage this idea, we provide methods to transfer symmetry representations from one texture to another, process the symmetries of a texture, and optimize textures with respect to properties of their symmetry representations. These methods are automatic and robust, as they don't require explicit detection of discrete symmetries. Applications are investigated for optimizing, processing, and transferring symmetries and textures.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms

Additional Key Words and Phrases: Symmetry analysis, texture synthesis

ACM Reference Format:

Kim, V. G., Lipmanm Y., and Funkhouser, T. 2012. Symmetry-guided texture synthesis and manipulation. ACM Trans. Graph. 31, 3, Article 22 (May 2012), 14 pages.
DOI = 10.1145/2167076.2167080
http://doi.acm.org/10.1145/2167076.2167080

1. INTRODUCTION

Many materials have textures (fine-scale, high-frequency variations) organized in distinctly recognizable spatial patterns (large-scale, low-frequency variations). For example, tiger pelts have fine-scale fur textures organized in large-scale striped patterns; floor carpets have fine-scale weave textures organized in large-scale ornamental patterns; and, brick walls have fine-scale mud textures organized in a large-scale block patterns.

The authors thank the NSERC, NSF (CNS-0831374 and CCF-0937139), Intel, Adobe, and Google for partial support of this project.

Authors' addresses: V. G. Kim (corresponding author), Y. Lipman, and T. Funkhouser, Princeton University, Princeton, NJ; email: vk@cs.princeton.edu.

DOI 10.1145/2167076.2167080
http://doi.acm.org/10.1145/2167076.2167080

In an attempt to model these textured patterns algorithmically, there has been a significant amount of research on example-based texture synthesis in computer graphics over the last decade [Efros and Leung 1999; Wei and Levoy 2000; Efros and Freeman 2001]. The key challenge in these methods is to provide a way for the user to guide the synthesis process, that is, specify what spatial patterns should appear in the output image. Previous methods have considered texture transfer [Efros and Freeman 2001], texture mixing [Heeger and Bergen 1995], texture-by-numbers [Hertzmann et al. 2001], and so on. In particular, Liu et al. [2004] suggested a tool for manipulating Near-Regular Textures (NRTs), providing methods for controlling geometric and color regularity and transfering deformations and lighting between NRTs. Their method provides excellent results when a deformation field induced from a deformed 2D lattice can be detected in the input image(s), but does not provide a solution for images with other types of patterns.

Concurrently, there has been a large amount of recent work in computer vision and graphics on analyzing and representing symmetries in objects. Methods have been proposed for measuring approximate symmetries [Zabrodsky et al. 1995; Kazhdan et al. 2004], for detecting partial symmetries [Mitra et al. 2006], for extracting repeating patterns [Park et al. 2009; Pauly et al. 2008], and for representing shapes in a symmetry space [Reisfeld et al. 1995; Podolak et al. 2006]. These representations encode not only perfect discrete symmetries, but also a continuous measure of how symmetric a pattern is with respect to every transformation within a group (e.g., translations by all vectors, reflections across all planes, etc.). As such, we conjecture that they are well-suited for characterizing the defining spatial patterns of most real-life textures.

Our goal is to utilize these automatically computed representations of partial and approximate symmetries to guide texture synthesis and manipulation. The key idea is to represent spatial patterns in a *symmetry space* that explicitly encodes how symmetric a pattern is with respect to a group of transformations and to use an objective function defined in that symmetry space to guide texture processing. For example, consider the symmetry transfer examples shown in Figure 1. In this case, the user wishes to create an image with the high-frequency characteristics of a "source texture" (shown in the top row) and with the low-frequency spatial pattern of the "target pattern" (left column), that is, transfer the spatial pattern of the target onto the source. Our approach is to represent the target pattern in a symmetry space (shown to the right of the target pattern) and to guide a texture synthesis process that copies patches of texture from the source to produce a result with a matching symmetry representation. This approach takes advantage of the fact that the target pattern is more prominent in the symmetry space than the original space, and therefore is better transferred in that domain.

The advantages of this approach are fourfold. First, it leverages the idea that symmetry representations are natural for characterizing spatial patterns in many images. Second, it does not require explicit detection of symmetries (e.g., extraction of a lattice), but only measurement of approximate symmetries, which is both robust and fully automatic. Third, it allows direct control over the symmetry properties of an image, for example, making an image more symmetric or asymmetric. Finally, it is general: It works with a

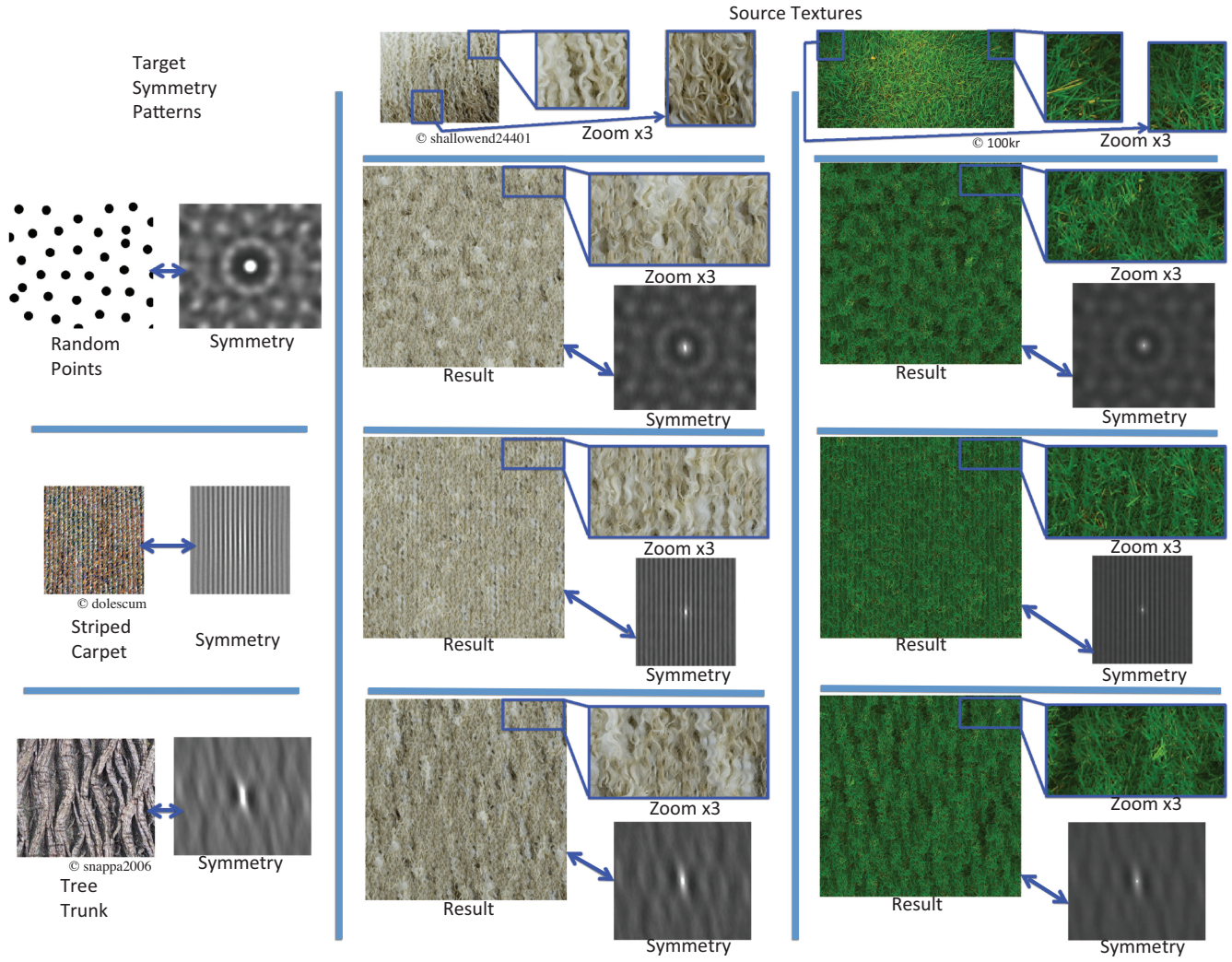

Fig. 1. Texture synthesis with symmetry transfer. Translational symmetries of targets (left column) are transfered to source textures (top row). Note how synthesized images have fine-scale details of sources and coarse-scale patterns of targets (symmetry representations are inset).

variety of input data types, texture synthesis methods, and symmetry representations, each of which can capture qualitatively different types of patterns.

The main contribution of our article is the idea that representations in symmetry space are a natural way to describe spatial patterns in many real-world textures. We also provide a framework to investigate this idea which includes a variety of methods for symmetry representation, objective function specification, and image optimization. Different combinations of the methods are shown useful for symmetry transfer (Section 4), symmetry processing (Section 5), and symmetry optimization (Section 6).

2. PREVIOUS WORK

Texture synthesis and symmetry analysis have both received a lot of attention recently in computer vision and computer graphics.

Texture synthesis. Algorithmic generation of textures dates back several decades [Ebert et al. 2002]. Most recent work has focused on texture synthesis by example, in which an input texture image drawn by an artist or captured in a photograph is resampled to produce a new texture image with similar local appearance and arbitrary size [Wei et al. 2009]. For example, Heeger and Bergen [1995] and Portilla and Simoncelli [2000] proposed parametric models that represent the pattern of an example texture with color histograms and multiscale-oriented filter responses and then synthesize new textures with matching statistics. This approach allows mixing of textures (taking pattern and color from two different sources) and interpolation of textures [Bar-Joseph et al. 2001; Matusik et al. 2005].However, there is still limited control over the result. Our method is synergistic with this approach: It provides a new way to guide texture synthesis toward a target by matching statistical representations of its symmetries.

The most common approach to texture synthesis is based on Markov Random Fields (MRF). This method models a texture as the realization of a local and stationary random process based on the assumption that pixels with similar neighborhoods should have similar colors. Efros and Leung [1999] provided an early MRF method that synthesizes textures one pixel at a time to produce pixel neighborhoods consistent with an example. Several approaches have been

proposed to improve the speed and quality of their method, including Wei and Levoy [2000], who used tree-structured vector quantization to speed up the neighborhood search, and Efros and Freeman [2001], who suggested copying bigger patches, introducing a way to blend them seamlessly. The patch-based approach can also provide more control over the texture synthesis process. For example, by selecting patches or pixels of certain intensity one can do texture transfer as in Efros and Freeman [2001]. A texture by number application proposed within image analogies framework [Hertzmann et al. 2001] allows synthesizing scenes guided by user's sketch. Better synthesis models use graph cuts [Kwatra et al. 2003] and/or global optimization [Kwatra et al. 2005] to synthesize textures from multiple irregularly sized patches. These methods also have enough flexibility to incorporate user control, for example Xu et al. [2009] suggested aligning texture features along shape feature lines, highlighting underlying shape. Our work extends these methods to include control over symmetries.

Symmetry analysis. Characterizing symmetries in an image is a long-standing problem in computer vision. Over the last two decades, several methods have been proposed for detecting, representing, and manipulating approximate and partial symmetries in shapes. For example, Zabrodsky et al. [1995] proposed an early method for measuring the *symmetry distance* of a 2D boundary with respect to a given transformation based on the amount of work it would require to make the boundary symmetric with respect to that transformation. Kazhdan et al. [2003] extended this definition to the domain of spherical grids and provided a Fourier space algorithm for efficient computation of symmetry distance for all plane reflections and rotations about a center point [Kazhdan et al. 2004]. Mitra et al. and others have proposed voting and clustering algorithms to identify approximate and partial symmetries robustly [Mitra et al. 2006, 2007; Podolak et al. 2006]. These methods have mainly been used for detecting a small, discrete set of symmetries, which later are used to guide shape processing applications [Golovinskiy et al. 2009].

Our approach is based on prior work that represents a shape in a *symmetry space*, a space parameterized by a group of transformations storing measurements of the symmetry distance of a image/shape with respect to every transformation in the group. For example, Reisfeld et al. [1995] defined a generalized symmetry transform, which measures the symmetry distance of an image with respect to point reflections across every position in an image. Podolak et al. [2006] considered a similar approach to define the planar reflective symmetry transform, which provides a measure of symmetry distance with respect to every plane (line in 2D) through the space of an object. Kazhdan et al. [2004] defined reflectional and rotational symmetry descriptors that measure correlations of an image with respect to transformations that fix the center of mass. These methods have been used for a variety of applications, including finding features in noisy images (e.g., eyes on a face [Reisfeld and Yeshurun 1992]), discriminating textures [Bonneh et al. 1994; Chetverikov 1995], and segmenting images based upon local symmetries [Kelly and Levine 1995]. However, they have not been used for texture synthesis.

Symmetry-aware texture processing. Symmetry detection has also been used in the analysis and recognition of textures. For example, Leung and Malik [1996] used a greedy optimization technique to group repeating texels detected in salient image regions. This technique only found correspondences and transformations for local patches, but did not consider global symmetries. Turina et al. [2001] improved the grouping by using a strong geometric regularity assumption.

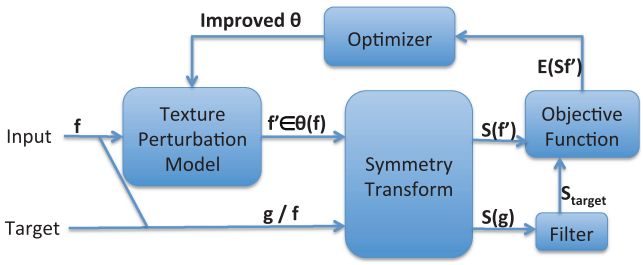

Fig. 2. Symmetry-guided texture processing framework.

Recently Park et al. [2009], Hays et al. [2006], and others have proposed algorithms for detecting a deformed lattice corresponding to regular elements in images, which can be used to guide texture processing. In the work most closely related to ours, Liu et al. [2004] explicitly model a near-regular texture as a warped lattice of repeating texels. They achieve irregularity that is common for real-life textures by storing color and geometry variance as offsets from otherwise perfectly symmetric structure. This generative model provides control over regularity of a texture and allows transferring large-scale properties like deformation or lighting between textures. However, it depends on a deformation model based on an underlying 2D lattice, and thus it can only be used when such a lattice can be detected automatically or when the underying lattice is specified with user input.

3. OVERVIEW

In this article, we investigate symmetry-guided texture synthesis and manipulation. The main idea is to specify the core spatial patterns of an image in a symmetry representation and to use an objective function defined on that symmetry representation to guide texture synthesis and manipulation.

This idea is very general, and it suggests a multitude of possible implementations for different applications. For example, one application might optimize an input image until it has a symmetry representation matching that of an example target image (symmetry transfer), while another application might deform an input image until the magnitude of its symmetry representation is maximal (symmetrization). These applications have many similarities, but the implementation details may be very different. So, for our investigation, we decompose the space of possible approaches into a generic set of computational steps, provide different implementations for each one, and investigate how they can be integrated to support different applications.

Figure 2 provides an overview of the framework. First, we select a *texture perturbation model*, θ, which transforms an input image, f, into an output image, f', while maintaining the high-frequency texture characteristics of f (examples include example-based texture synthesis, as-rigid-as-possible deformation, etc.). Next, we introduce a *symmetry transform*, S, which computes a *symmetry representation*, $S(f)$, of any image f representing its low-frequency pattern characteristics (examples include the reflective symmetry transform, rotational symmetry descriptor, etc.). Then, we introduce a target symmetry representation S_{target} that describes the large-scale pattern desired in the output image (examples ways of specifying S_{target} include filtering $S(f)$ or computing $S(g)$ for another input image g). Then, we define an objective function, $E(S(f'))$, which measures how well the symmetry representation of f' matches a desired target symmetry representation S_{target} (examples include the L^2 difference between S(f') and S_{target}, the variance of $S(f')$, etc.). Finally, we perform an optimization to search for the output image

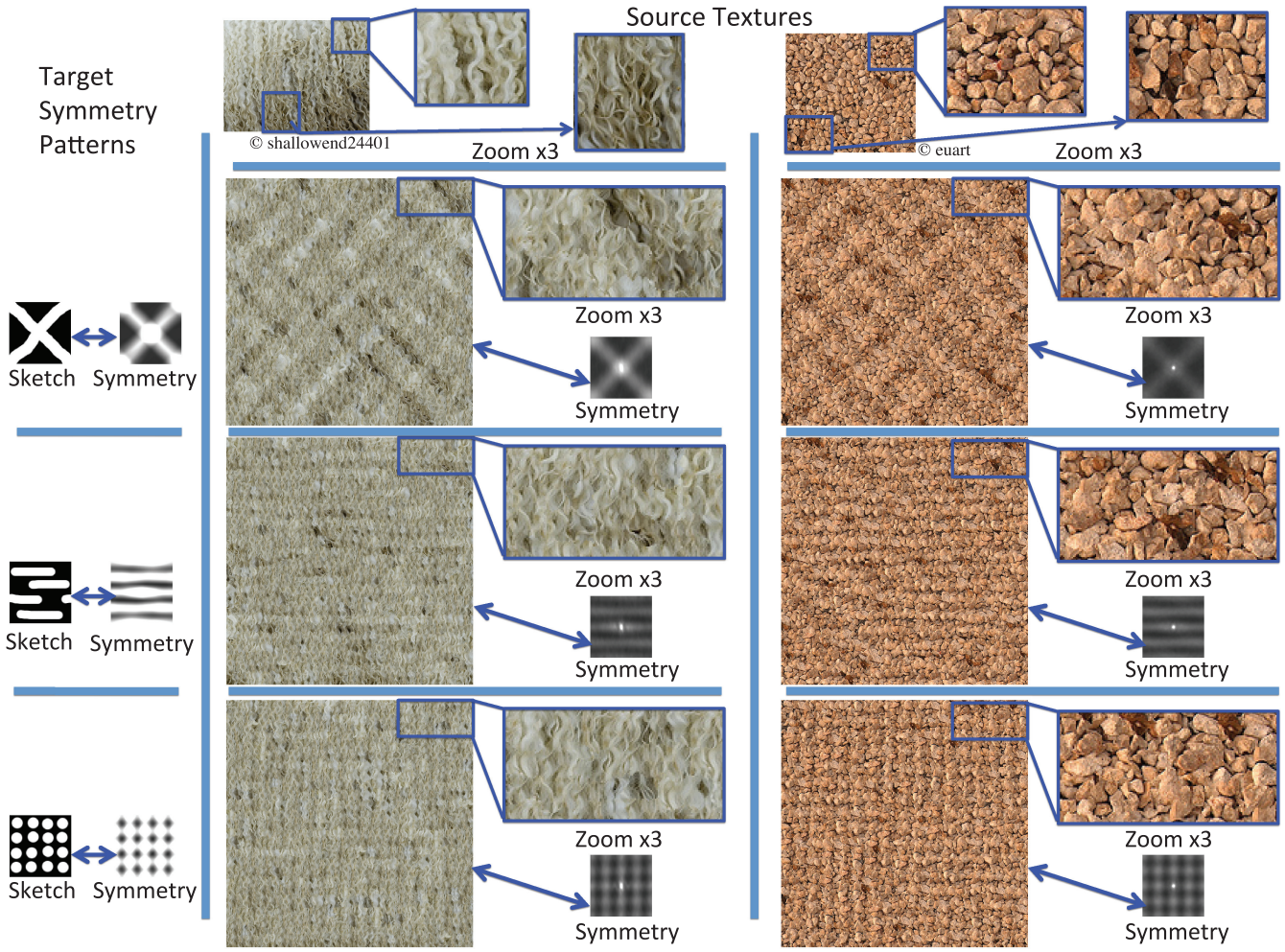

Fig. 3. Texture synthesis with symmetry transfer from sketches. New images are synthesized with local details of source texture images (top row) and global patterns of sketched target images (left column). For each synthesized image, the larger left image shows the result, while the smaller images to its right show a zoomed view (top) and symmetry representation (bottom).

f' whose symmetry representation $S(f')$ minimizes the objective function $E(S(f'))$ among all possible images in the output space of $\theta(f)$.

Given this framework, we are able to consider a wide variety of symmetry-guided texture synthesis and manipulation applications. The following three sections provide example applications, grouped by how the target symmetry representation is specified.

—*Symmetry transfer:* the target is provided by the symmetry representation of a different image g.

—*Symmetry filtering:* the target is generated by filtering the symmetry representation of the input image f.

—*Symmetry optimization:* the target is defined by the optimum of an objective function computed directly from the symmetry representation.

For each application, we discuss possible design trade-offs in the choice of input image, symmetry representation, objective function, and texture perturbation model, and we provide possible implementations and present representative results.

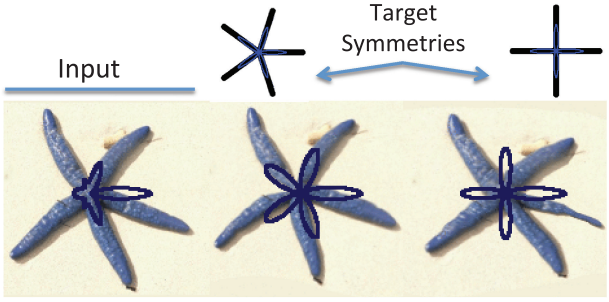

Fig. 4. Symmetry transfer by as-rigid-as-possible deformation. The input image of a star fish on the left is deformed to minimize the difference of its rotational symmetry descriptor with that of images with perfect five-fold rotational symmetry (middle) and perfect four-fold rotational symmetry (right). Note that the symmetries of the target are transferred to the input in both cases (four of the legs form approximate right angles in the rightmost image).

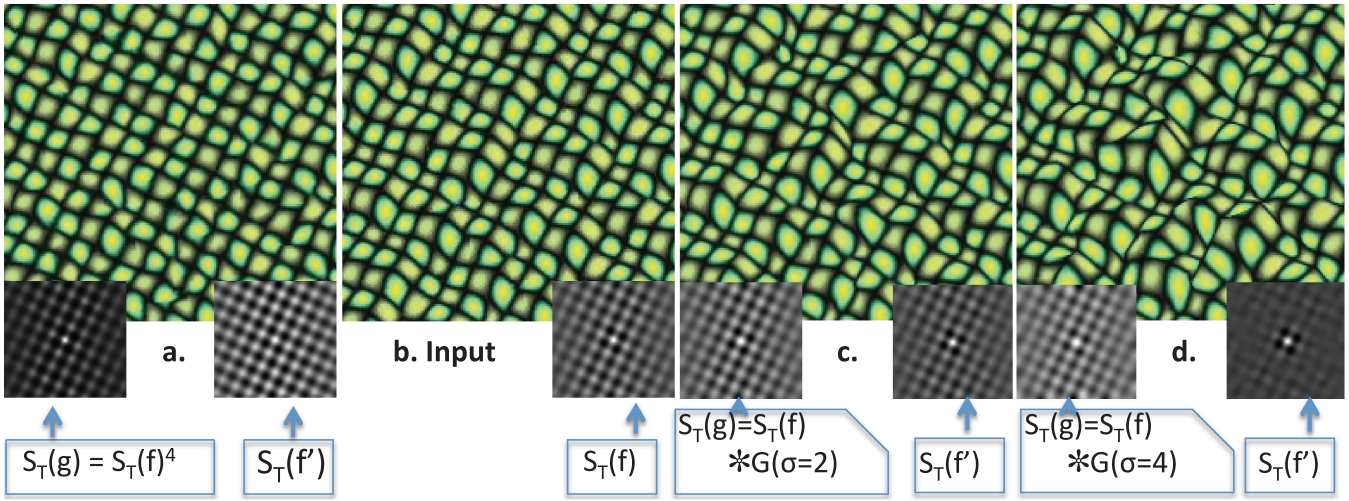

Fig. 5. Symmetry filtering by as-rigid-as-possible deformation. Starting from the input image shown in (b), its symmetry representation (inset below and right) is filtered to produce new targets. (a) shows a sharpened symmetry representation, while (c) and (d) are blurred with smaller and larger sigma, respectively. Warping the original image with as-rigid-as-possible deformation to minimize differences to the targets produces the images shown in (a,c,d). Note that this process is able to control the symmetry of the image with simple image processing filters.

4. SYMMETRY TRANSFER

There are many scenarios in which a user might want to create an image with the fine-scale textures of a source image, f, but the large-scale patterns of a target image, g (e.g., as in Figure 1).

Our framework addresses this problem by capturing the large-scale patterns of g in a symmetry representation, $S(g)$, and then perturbing the source image f to find the new image $f' \in \theta(f)$ that minimizes the difference of its symmetry representation, $S(f')$, from the target's, $S(g)$. We have investigated several applications of this type, using the L^2 distance between symmetry representations as the objective function to minimize, for example,

$$\arg\min_{\theta} E_{L^2}(S(\theta(f))) = \int \|S(\theta(f)) - S(g)\|^2. \qquad (1)$$

The following provides two examples based on different texture perturbation models: texture synthesis and as-rigid-as-possible deformation.

Example 1: *Texture Synthesis.* In our first example, we investigate how symmetry transfer can be utilized to guide a patch-based texture synthesis algorithm. In this case, a new image f' of arbitrary dimensions is created by composing patches of pixels extracted from a source image f in order to minimize the L^2 difference between the symmetry representation of f' to that of a target image g.

For this example (and several others that follow), our symmetry representation is the *translational symmetry transform*, $S_T(f)$, which measures the correlation between a toroidal function f and itself at every possible translation.

$$S_T(f)[\tau_x, \tau_y] = \frac{\int\int f(x,y) f(x+\tau_x, y+\tau_y) dx dy}{||f||^2} \qquad (2)$$

This function is efficient to compute in the frequency domain, it is not bijective, and it has high values only for translation vectors that align repeating image elements, and thus it provides a good representation for translational structure within our framework. To compute $S_T(f)$ for an image f, we calculate the discrete autocorrelation function for f, normalize it to mean $\mu(S_T(f)) = 0$ and variance $\sigma^2(S_T(f)) = 1$, and then (optionally) limit its domain to a fraction of the input image size (to capture smaller patterns). Examples of $S_T(f)$ are shown as insets for their corresponding images f in Figures 1 and 3, note how repeated patterns are clearly evident.

With this symmetry representation extracted from a target image as a guide, we synthesize new images using a variation of the quilting technique described in [Efros and Freeman 2001] as our perturbation model θ_{synth}. Our perturbation model $f' \in \theta_{synth}(f)$ allows exploring a space of plausible images by specifying a list of patches in the source texture f, such that neighboring patches in the resulting image f' can be stitched seamlessly. We find a locally optimal solution in that space by a random walk procedure. In particular, we first initialize a tiled image of a desired size using random patches extracted from the source texture. Then, we iteratively change the patch in every tile, one-by-one picking a new patch consistent with its neighbors using the methods described in the original paper. However, rather than picking a patch randomly from the ones that have small differences in overlaps with their neighbors, as is done in the original work, we choose the patch that causes the symmetry representation of the resulting image to move closest to the symmetry representation of the target. As such, we end up with an image composed of small-scale elements from the source texture, but the symmetry representation of the target pattern.

Example images synthesized with this method are presented in Figures 1 and 3. For each example, the top row shows the source texture, and the left column shows the target. Note how synthesized image shave the coarse-scale repeated patterns modeled after the targets (i.e., their translational symmetry representations match the targets') but the fine-scale texture of the source (e.g., images shown at zoom x3 are similar). Our method works equally well for transferring patterns from texture images (Figure 1) and from rough hand-drawn sketches (Figure 3). We find the latter case particularly interesting because it provides the opportunity for an interactive program to modify the translational structure of a texture with a simple sketching interface (e.g., by drawing an "X" to create

cross-hatches, as in the first row of Figure 3). Note that even a very small sketch is sufficient to define target symmetry.

Example 2: *Image Deformation.* Another potential application of symmetry transfer is to deform an image to conform to the symmetry representation of a target.

To investigate this case, we implemented an as-rigid-as-possible deformation [Igarashi et al. 2005] as texture perturbation model θ_{deform} and applied it within an iterative optimization algorithm whose objective function measures the L^2 difference between the symmetry representation of the image and that of a target. Thus, an image produced by this perturbation model $f' \in \theta_{deform}(f)$ is a warped version of the input image f and is defined by a displacement of control points. It is optimized with a gradient descent algorithm. Specifically, given an image, $N = 256$ feature points are extracted automatically by a Haar corner detector (if less than N points are extracted, additional points are distributed via dart throwing). Points are then meshed via a Delauney triangulation, and assembled into a multiscale hierarchy for coarse-to-fine warping. Starting at the coarsest level for each control point we estimate a gradient with respect to energy function $E(S(\theta(f)))$, using a prescribed initial step (e.g., 5 pixels are used in our examples). We iteratively go over all control points at the current level and move them in the direction of a gradient, until we reach local minima or maxima. Then the step is halved and cycles repeat until step is below 1 pixel. Then our optimizer goes to a finer level with halved initial step.

Figure 4 shows an example result of this process. In this case, we experimented with a symmetry representation based on the *rotational symmetry descriptor*, $S_R(f)$, which measures the correlation between a function f and itself at every rotation around a center point [Kazhdan et al. 2004].

$$S_R(f)[\gamma] = \frac{\int\int f(r,\theta) f(r,\theta+\gamma) r\,dr\,d\theta}{||f||^2} \tag{3}$$

This function is one-dimensional (it is shown in the inset images as a blue curve offset further from the center at rotations with higher values, where the horizontal line corresponds to 0 rotation), it is not bijective, it is efficient to compute in the frequency domain, and it has high values only for rotations that align repeating image elements. We use it for this example because it captures the rotational pattern of the target image effectively, and to investigate the generality of our framework with respect to different symmetry representations.

In Figure 4, the image on the left shows an input photograph of a star fish; the middle image shows how it was warped when provided a perfect five-way cross as the target image; and, the image on the right shows the result of warping when given a four-way cross as a target. Note that the fivefold rotational symmetry of the star fish is enhanced in the middle example, while it is (purposely) destroyed in the example on the right (the star fish moves towards a four-way cross with an extra leg). In both cases, the input is successfully warped to have the rotational symmetries of the synthetic target.

5. SYMMETRY PROCESSING

A second way for a user to control the symmetric patterns in an image is to apply operations to adust its symmetry representation. For example, one might want to "sharpen" or "blur" the symmetries of an image to make its repeating patterns more or less prominent. Or, one might want to adjust the spacing of repeated patterns without deforming fine-scale details of an image. Since these types of patterns are represented better in the symmetry representation than they are in the original image, it is natural to adjust them with filtering operations in symmetry space.

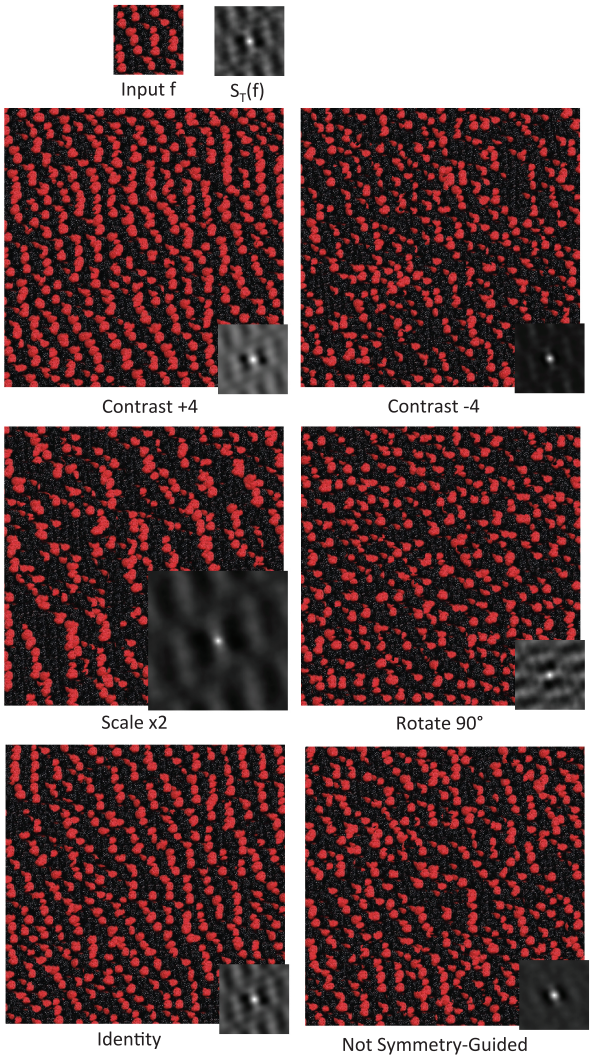

Fig. 6. Texture synthesis with pattern processing. In these examples, contrast, scale, rotation, and identity filters have been applied to the original translational symmetry representation (top right) to form a target for texture synthesis.

Implementing this idea is very simple within our framework. Starting with an input image f, we: (1) compute its symmetry representation $S(f)$, (2) apply any operation on $S(f)$ to produce a new target symmetry representation S_{target}, and then (3) optimize f with a perturbation model so that its symmetry representation $S(f')$ matches the processed one S_{target} as closely as possible (e.g., minimizes the L^2 difference).

The following paragraphs discuss some example implementations of this idea and provide results for two applications.

Example 1: *Symmetry Filtering.* There are cases where a user would like to control the regularity of repeating patterns in an image, for example, to fix unwanted distortions in a pattern that should be symmetric, or to reduce the symmetries in a pattern that should appear random.

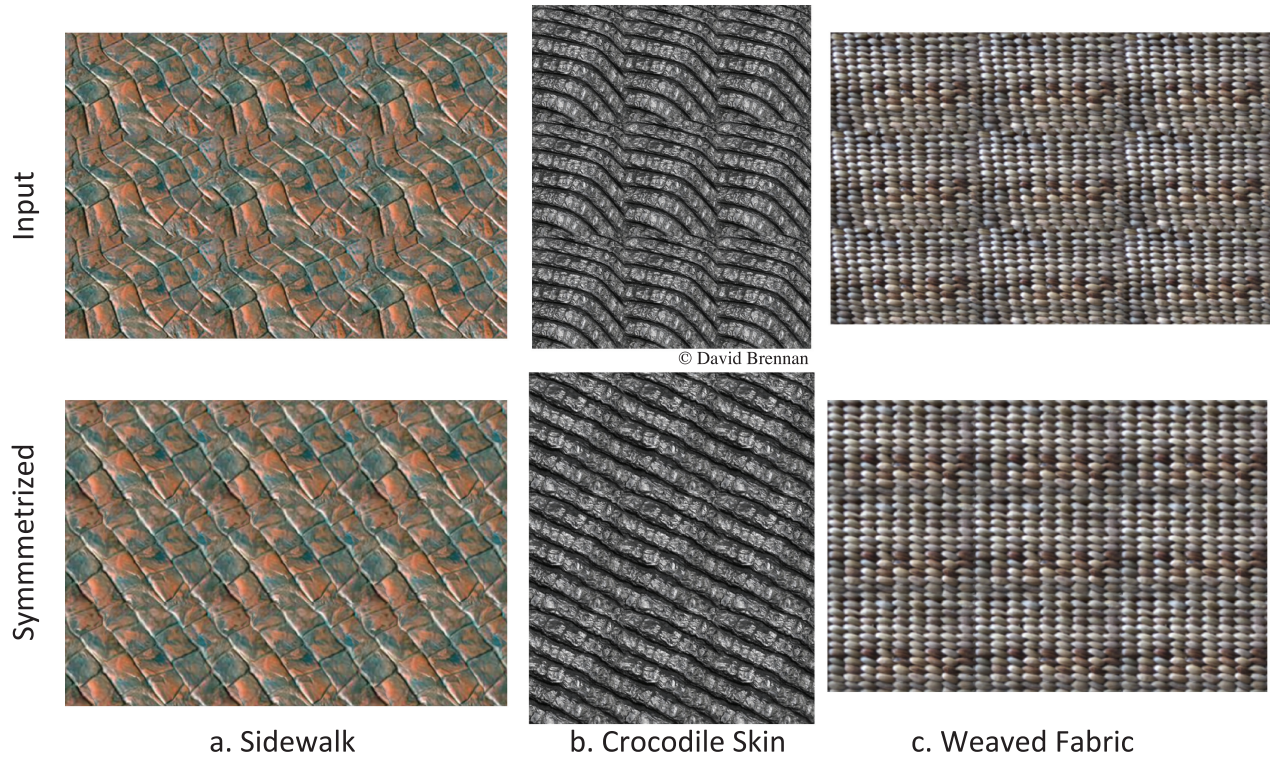

Fig. 7. Texture tiling. Each pair shows a 3 × 3 repeated tiling of the input texture with seamless blending of edges. Image at the top shows tiling without symmetry optimization and image at the bottom shows tiling with symmetry optimization. Note that our method handles various scales and types of symmetric patterns without any user intervention.

Previous approaches have addressed this problem by representing an image as a 2D lattice describing its underlying repeating pattern and a deformation field describing how the image deviates (in geometry and lighting) from the closest image with the repeating pattern. With that representation, the regularity of an image can be controlled very naturally by increasing or decreasing the deformation field. However, this approach only works when the underlying 2D lattice can be extracted from the input image, which is not possible with automatic for many images, including most of the examples in this article (as shown in Section 7).

Our approach is an alternative when the input has no detectable lattice-based model. We utilize approximate symmetry representations computed via correlations of an image with itself (e.g., $S_T(f)$ and $S_R(f)$) and apply standard signal processing filters to make adjustments to the pattern. The advantages of using these symmetry representations (rather than a generator) are: (1) they can be computed automatically and robustly, since every pixel output by an autocorrelation combines many elements of the image; (2) they encode perfect symmetries, partial symmetries, approximate symmetries, and even perfect asymmetries, and thus they can describe patterns that do not have a parameterizable generator; and, (3) they can be represented as discrete signals in symmetry space (e.g., $S_T(f)$ is an image, and $S_R(f)$ is a histogram), and thus arbitrary filtering operations can be applied on symmetry representations naturally.

Figure 5 shows examples of controlling the regularity of repeated elements in an image by applying "sharpen" and "blur" filters on its translational symmetry representation. The figure is generated with as-rigid-as-possible perturbation model. The input image f is shown in the second column (b). The image on its left (a) was

Fig. 8. Image symmetrization by unwarping lens distortion.

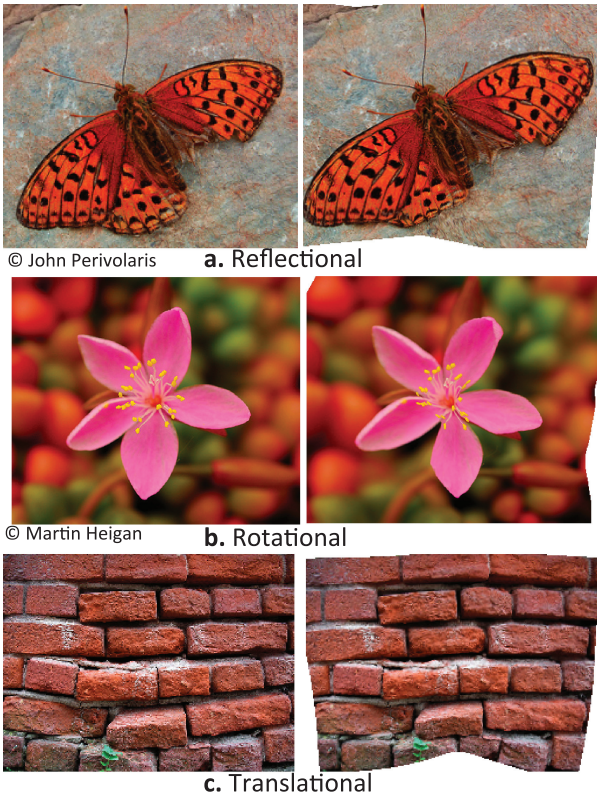

Fig. 9. Image symmetrization by as-rigid-as-possible deformation. Input images (left) are deformed to maximize either reflectional or rotational symmetries (right).

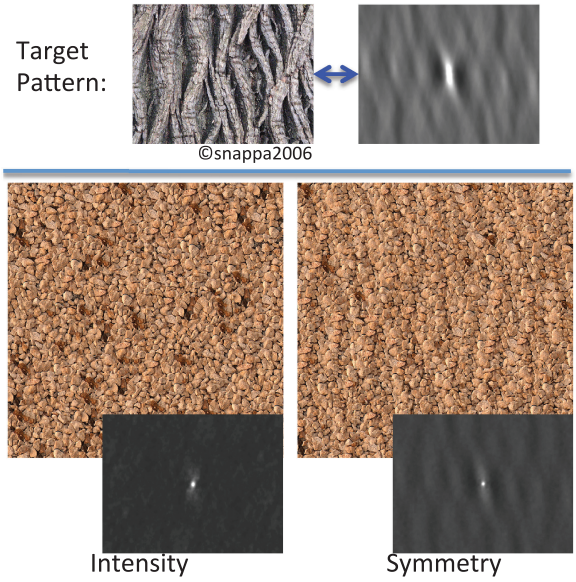

Fig. 10. Comparison of symmetry transfer to a previous method based on intensity transfer [Efros and Freeman 2001]. The left image was synthesized with intensity transfer, while the right image was synthesized with our method matching symmetry representations of synthesized texture and a target. Note that due to little lighting variation in the texture Intensity Transfer fails completely. Our method does not fail, because it matches correlation of intensities, rather than intensities. The source image of the gravel texture (c) euart.

produced by "sharpening the translational symmetry representation", that is, finding the image f' whose $S_T(f')$ is most similar to $S_T(f)^4$ after normalization ($\int |S_T(g)| = \int |S_T(f')|$). Note how the symmetry representation has sharper peaks, causing the image to become more symmetric. The two images on the right (c and d) were produced by "blurring the translational symmetry representation": applying a Gaussian filter with $\sigma = 2$ and $\sigma = 4$ pixels, respectively, and then solving for the deformation of f whose S_T best matches the result. Note how the image becomes more asymmetric. These filters are intuitive, and the processing pipeline is automatic, robust, and general (works for many types of symmetry representations), and thus we believe it provides a useful tool for controlling symmetries in images.

Example 2: *Pattern Processing.* Image manipulations that modify large-scale patterns while preserving fine-scale texture details can be implemented by combining the filtering approach of this section with the texture synthesis of the previous section. Specifically, given an input image f and its computed symmetry representation $S(f)$, a filter can be applied to $S(f)$ to form a target symmetry representation S_{target}, and the quilting texture synthesis algorithm can be used to generate an output image f' whose symmetry representation $S(f')$ matches S_{target} as best as possible.

Figure 6 shows some results produced with this approach. The input image f (top row left) contains "wavy, partial rows of berries," a complex pattern that is not easily described by a lattice-based model. Yet, the translational symmetry representation $S_T(f)$ (shown to the right in the top row) is able to capture the pattern, and applying filters to $S_T(f)$ provides a simple way to modify it.

The middle rows of Figure 6 show the results when S_{target} is constructed by a variety of filters: changing the contrast in $S_T(f)$, scaling $S_T(f)$ by a factor of two, and rotating $S_T(f)$ by ninety degrees. Please note how the amount of symmetric structure changes as contrast is added and removed from $S_T(f)$ and how the spacing between the wavy rows increases when $S_T(f)$ is scaled up, while the local texture of the berries is preserved. In the rotation result, horizontal "rows" of berries are synthesized, but the rows are not nicely shaped due to the limited availability of suitable patches containing horizontal arrangements of berries in the input image; perhaps smaller patches (we use 32×32 pixels in all examples) or allowing synthesis with rotated patches would address this problem.

Examining the bottom row of images in Figure 6, we see that the quilting texture synthesis algorithm reproduces the large-scale pattern of the input better when guided by the target symmetry representation than when not. The left image (labeled "Identity") shows the result of our symmetry-guided quilting algorithm when the S_{target} is the original $S_T(f)$. In contrast, the right image (labeled "Not Symmetry-Guided") shows the result of applying the standard quilting algorithm to synthesize the output image (without trying to match $S_T(f)$); please note that the large-scale, wavy row pattern is better preserved in the symmetry-guided result. This difference suggests that the symmetry representation indeed provides a useful description of the input pattern.

6. SYMMETRY OPTIMIZATION

A third way to adjust the symmetric patterns of an image is by optimizing an energy function defined directly on its symmetry

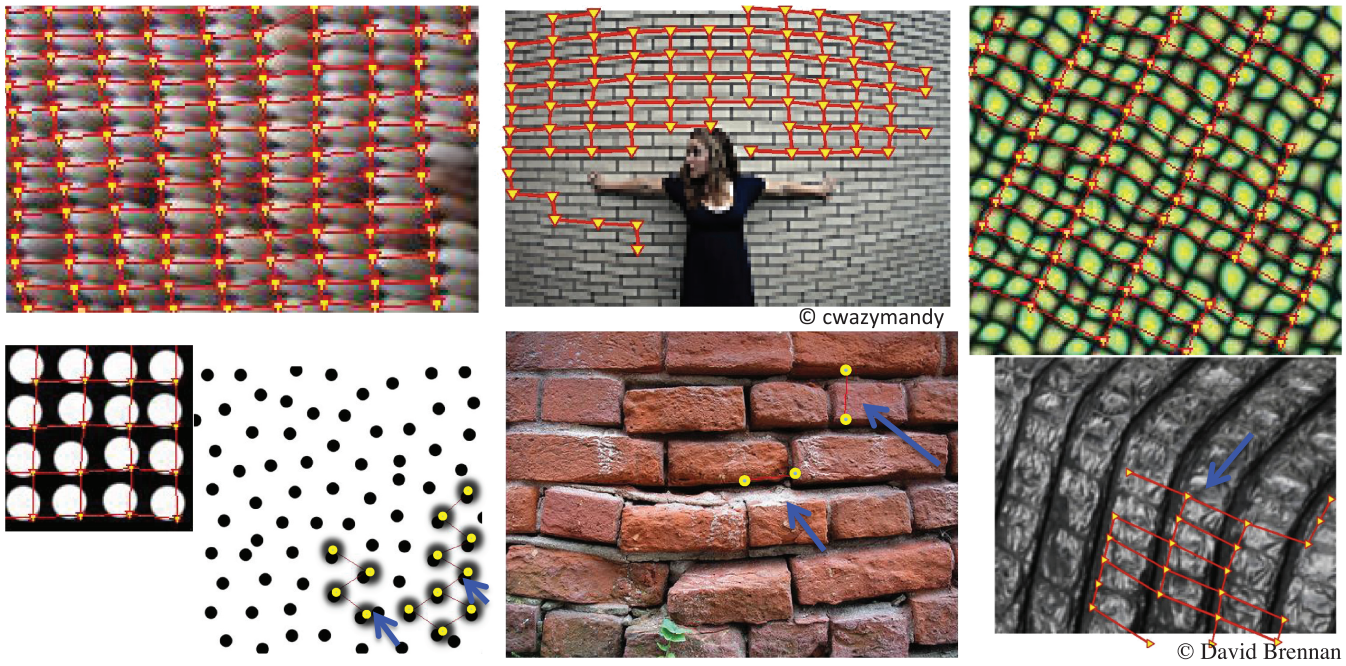

Fig. 11. We ran state-of-art lattice detection code provided by Park et al. [2009] on all images used in this article (except for Figures 5, 10(a), and 10(b), where we have used nontranslational symmetries). This figure shows all examples where a lattice was detected at all (blue arrows point to extracted regions, which are not very visible). Note that a lattice appropriate for our application was detected in only 3 out of 23 images (top-left corner).

representation. For example, it is possible to "symmetrize" an image by deforming it to maximize the concentration of energy in its symmetry representation. This type of control requires no target image or processing operation, just an objective function.

We have investigated the utlity of several types of objective functions defined on symmetry representations. The most natural one models the "strength of symmetries" in an image by measuring the variance of its symmetry representation, that is,

$$E_{symm}(S(f)) = \int \|S(f) - \mu(S(f))\|^2, \tag{4}$$

where $\mu(S(f)$ is the mean of the symmetry representation. This objective function will be highest when all the energy of the symmetry transformation is concentrated on a few transformations (image has perfectly repeated patterns) and lowest when the image is completely uniform in color (no patterns at all). As such, optimizing with respect to this objective function provides a way to symmetrize and desymmetrize images, operations that can be useful in the following computer graphics applications.

Example 1: *Texture Tiling.* Due to memory and complexity constraints it is not always possible to synthesize a texture on a large surface, and thus it is common to use toroidal textures that tile the plane without noticeable seams (e.g., in OpenGL applications). However, creating tiling textures from an input photograph is nontrivial. Seams can be avoided by blurring across the boundaries, but artifacts in large-scale patterns will be noticeable unless the pattern is toroidally symmetric (as shown in the top rows of Figure 7).

To address this problem, we optimize textures to maximize the variance of their translational symmetry representations. In our experiments (Figure 7), symmetry energy (Eq. (4)) is measured for two input functions: f_{lum} and f_{1-lum}, and texture is optimized with respect to sum of these energies, with the exception of the image in Figure 7(d), where f_{edge} is used due to high variation in color. The optimization produces an output image using a texture perturbation model that allows as-rigid-as-possible deformation followed by mean value coordinate interpolation across toroidal boundaries. Note that in our perturbation model we start by zooming on the center of an input tile, leaving 20% (relative to width or height) boundary pixels which might be pulled in during the deformation. This process diminishes the asymmetries that would become noticeable in a tiling and avoids visible artifacts near tile boundaries.

Figure 7 shows the results of this process for several examples. The top row shows the result of tiling the input images without any processing; note how the repeating tiles are clearly visible due to breaks in large-scale patterns and discontinuities at tile boundaries. The bottom row shows the results of tiling the textures output by our optimization process. Note how large-scale patterns are not dominated by the tiling structure, and that strong edges continue seamlessly across tile boundaries. It is important to note that our processes do not explicitly align edges, but increasing symmetry automatically favors alignment of lines across boundaries, as well as in the middle of the texture, which helps to diminish noticeable artifacts.

Example 2: *Image Symmetrization.* There are many cases in which it is desirable to fix the asymmetries in an image to improve its appearance and/or to simplify further processing. For example, a user may want to remove distortions in a photographic image of a symmetric object to eliminate perspective foreshortening and/or radial distortions due to lens effects. Or, a scientist may want to "symmetrize" a photograph of a nearly regular structure before further image processing to simplify search of correspondences between elements in a repeating pattern.

As in the previous example, this application can be addressed directly by optimization of the input's symmetry representation. For

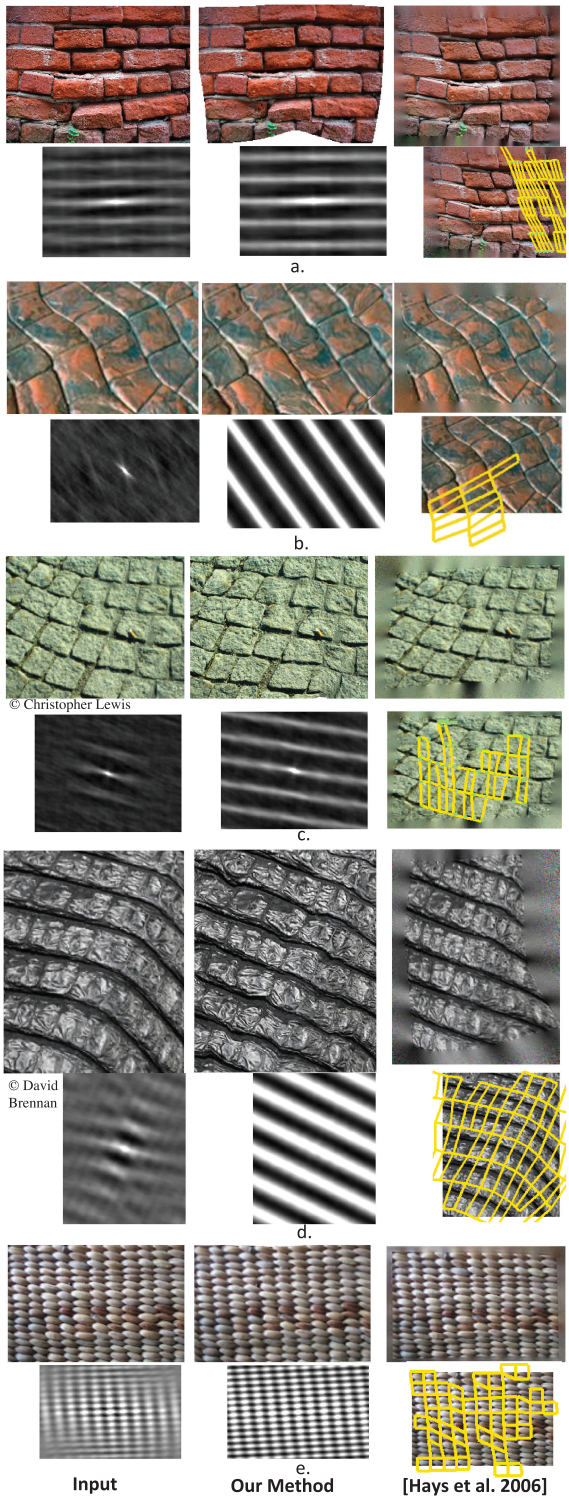

Fig. 12. This figure compares our symmetrization results (middle column) to those of Hays et al. [2006] (right column). Within each row, the symmetry representation is shown just below every image, that is, S_T for images in the left and middle columns, and the extracted lattice for images in the right column.

example, Figure 8 shows how radial distortions can be removed by optimization of translational symmetries. In this case, the center of radial distortion was specified manually, and then the space of images producable by varying the radial lens distortion parameter in Tsai's camera model [Tsai 1986] (θ_{lens}) was searched for the one with maximal variance in the translational symmetry representation. This process automatically straightens the lines between rows of bricks, which removes the distortion in the image. While this example may seem contrived, since the input has a regular pattern in it, we believe that it could be useful to calibrate a camera on-the-fly, that is, take a picture of highly regular object (like brick wall) and then use our method to learn the warp applied by the camera. This process would work without prior knowledge of the repeating structure in the photographed image, when the repeating structure is only approximate, and when it would be difficult to fit a parameterized representation of the pattern.

Figure 9 shows the results of maximizing the variance of other symmetry representations. The image on the left (a) shows the as-rigid-as-possible warp that maximizes variance in the reflective symmetry descriptor ($S_{Refl}(f)$) described in Kazhdan et al. [2002], and the image in the middle (b) shows the same for the rotational symmetry descriptor, $S_R(f)$. Note how the repeated structures of the images are enhanced, even though the structure is not detected explicitly.

7. COMPARISON

In this section we compare to alternative methods and discuss advantages and disadvantages of our approach.

Pattern as intensity field. One can model a low-frequency pattern as an intensity map overlaid over a texture. If luminance of high-frequency elements matches the intensity map the result would resemble a pattern of an underlying intensity map. However, if the source texture does not have enough illumination variety, the results obtained by symmetry transfer are significantly better than the results obtained by intensity transfer. For example, Figure 10 compares our results to a texture transfer method that matches pixel intensities to a target [Efros and Freeman 2001]. First we quilted the target to an appropriate size to create the target intensity map, and then used code provided by authors to perform texture transfer. In their result (left column), the target pattern was not reproduced because source texture does not have enough dark regions to create black points. The texture synthesized with our method (on the right), however, has similar pattern as the target, because of correlation of similar colors at the appropriate offsets.

Lattice-based texture synthesis. Some texture patterns are nearly regular and can be modeled as a warped lattice of texture elements. In these cases, Park et al. [2009] provide a method to construct a lattice-based model automatically, and Liu et al. [2004] provide a method for texture processing and transfer operations using such a model; these methods could be combined to produce an automatic system for processing of near-regular textures. While this approach is preferable when a 2D lattice pattern can be extracted automatically from the input image(s), it is not applicable to the wide range of examples considered in this article. For example, consider the target pattern in top-left of Figure 10: The tree trunk has a clear pattern, but it is definitely not a 2D regular lattice. To test how many of the examples in this article could have been reproduced with lattice-based texture processing methods, we ran state-of-art lattice detection code provided by Park et al. [2009] on all our input images, restarting the program for each image at multiple scales and then manually selecting the best result. We found that it was

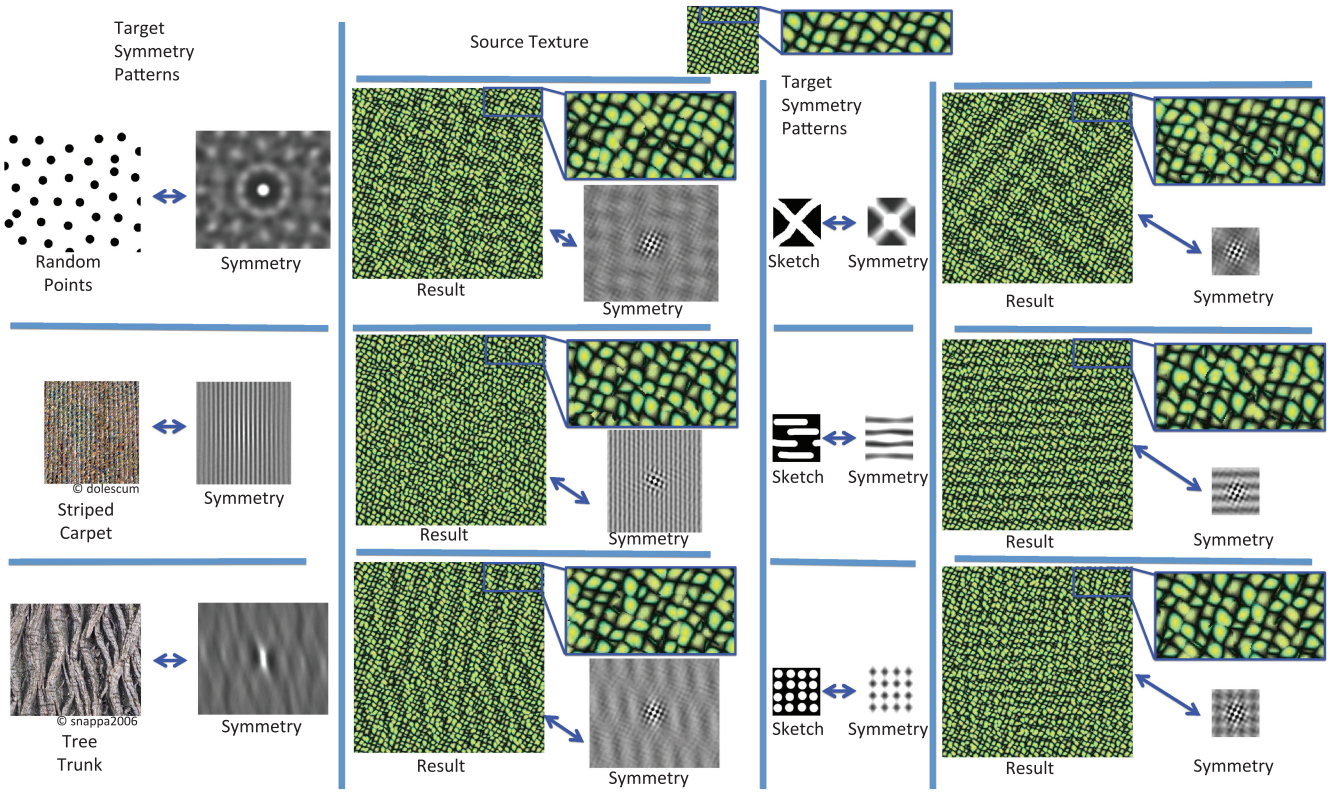

Fig. 13. Texture synthesis example demonstrating limitations of the proposed approach.

able to extract any lattice at all for only 8 out of 23 images (shown in Figure 11). In only 3 of those 8 cases (first two images of top row and first image of second row), the detected lattice was appropriate for the application demonstrated in our article. For the others, the detected lattice captured the pattern for only part of the image (top-right corner) or provided an incorrect model of repeating elements (bottom row) that would lead to poor results for the intended application. This result suggests that most of our examples are not easy to reproduce with alternative methods based on lattice extraction.

Warping a lattice for symmetrization. We also tested whether the lattice-based method of Hays et al. [2006] could be used as an alternative to our method for symmetrizing the images tested in our article (Figures 7, 16, and 15). Their approach iteratively warps an image to regularize a detected lattice, and thus it should provide good results when the correct lattice is detected automatically and poor results otherwise. Indeed, using the original implementation provided by the author, we found that Hays et al. [2006] is quite effective at symmetrizing images with extractable lattice structures. However, most of the examples in this article are not of that type. For each of our examples, we show the symmetrized image and extracted lattice with the best a-score (amongst all iterations for 20 random restarts) in the rightmost column of Figure 12. In these results, a complete lattice was extracted for only one of the five examples (Figure 12(d)). For the other examples, Hays et al. [2006] extracts a partial lattice, and thus would be difficult to use for symmetrization without noticeable artifacts. For comparison, our results for these same examples are shown in the middle column of the figure.

8. DISCUSSION

In this section we discuss limitations of our approach and factors affecting design decisions and parameter selections in our example applications.

Limitations of the approach. Our framework is limited to texture synthesis and processing applications where: 1) the target pattern has spatial structures with frequencies lower than those in the source texture, 2) the symmetry representation captures those spatial structures (not necessarily uniquely), 3) the texture perturbation model is flexible enough to reproduce large-scale properties of the target pattern when applied to the source texture, but constrained enough to retain fine-scale properties of the source texture, and 4) the optimization is constrained to find a (possibly local optimum) solution that is not trivally equal to the target.

Figure 13 investigates some of these limitations empirically with a texture synthesis example, where a source texture (green dots) is used to synthesize images with the six target patterns used in Figures 1 and 3. Looking at these results, one could conclude that the target pattern is reproduced in four of the six outputs (all except the top two in the left column). The system fails to reproduce the "Random Dots" pattern (top left) because the texture perturbation model is not flexible enough to generate an output image with large dots by quilting 32×32 pixel patches of small green dots. It fails to reproduce the "Striped Carpet" pattern (second row on left) because the dominant frequencies in the source texture are similar to those of the target pattern. The system also fails to reproduce local properties of the source texture in some of these examples (there are seams and oddly shaped dots), but that is mainly a limitation of

Fig	Input	Symm.	Obj. Fn.	Trg	Perturb.
1	f_{lum}	S_T	E_{L^2}	S(g)	θ_{synth}
3	f_{lum}	S_T	E_{L^2}	S(g)	θ_{synth}
4	f_{1-lum}	S_R	E_{L^2}	S(g)	θ_{deform}
5	f_{lum}	S_T	E_{L^2}	FLT(S(f))	θ_{deform}
6	f_{lum}	S_T	E_{L^2}	FLT(S(f))	θ_{synth}
8	f_{lum}	S_T	E_{symm}	N/A	θ_{lens}
9,a.	f_{edge}	S_{Refl}	E_{symm}	N/A	θ_{deform}
9,b.	f_{edge}	S_R	E_{symm}	N/A	θ_{deform}
9,c.	f_{edge}	S_T	E_{symm}	N/A	θ_{deform}
10	f_{lum}	S_T	E_{L^2}	S(g)	θ_{synth}
15	f_{lum}	S_T	E_{symm}	N/A	θ_{deform}
7	f_{lum}, f_{1-lum}	S_T	E_{symm}	N/A	θ_{deform}

Fig. 14. Per figure details. This table lists combination of modules used to produce each example in the article. Input signals for symmetry transform can be f_{lum}: luminance of each pixel, f_{1-lum}: inverted luminance, f_{edge}: an output of the edge detector. Description for symmetry representations S_T, S_R, S_{Refl} can be found in Eqs. (2), (3) and Kazhdan et al. [2002] accordingly. Objective functions E_L^2, E_{symm} are defined in Eqs. (1) and (4). Target in a symmetry space can be obtained either from another example image ($S(g)$) as explained in Section 4, or by filtering symmetry representation of an input ($FLT(S(f))$) as explained in Section 5. Finally, perturbation models and corresponding optimization methods θ_{synth}, θ_{deform} are described in Section 4: Example 1 and Example 2, and the deformation model for compensating for radial lens distortion θ_{lens} is described in Section 6, Example 2.

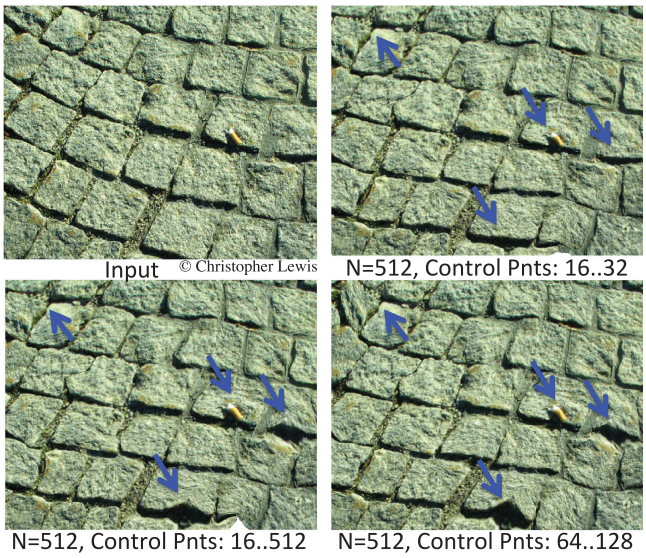

Fig. 15. Influence of flexibility of deformation on a resulting texture. Note as the deformation model becomes more flexible, it can potentially create undesirable textures with too much warping. Similarly, artifacts can appear if the image is deformed only at higher resolution.

our texture synthesis algorithm and reluctance to tune parameters for each individual example rather than a limitation of the overall approach.

In spite of these limitations, the framework is quite general. In our investigation, we have found several combinations of input textures, target patterns, symmetry representations, objective functions, and texture perturbation models that produce interesting results within our framework (see Figure 14 for a complete list of combinations used for every example in this article). However, our investigation is far from exhaustive, and we expect others to find other interesting combinations in future work.

Parameter selection. Selecting an appropriate texture perturbation model is the most difficult aspect of our system. The model must be flexible enough to reproduce the target pattern, but rigid enough to preserve local properties of the source texture. In this article, we experiment with quilting texture synthesis and as-rigid-as-possible deformation, both of which require selection of parameters to balance this trade-off.

The quilting texture synthesis algorithm of Efros and Freeman [2001] requires patch size as its input. In our work, for all textures synthesized (except for Figure 16), patch sizes were set to 32 pixels, with overlap of 12 pixels. Choosing a smaller patch size gives a more flexible perturbation model, but might synthesize textures with undesirable seam artifacts and might not preserve desired high-frequency properties of the input texture. Choosing a larger patch size is more likely to preserve the local structure of the source texture, but might be too restrictive to reproduce the target symmetry pattern. This trade-off is illustrated in Figure 16. The leftmost image clearly has the linear pattern, such as the target (i.e., the low-frequency pattern), but also destroys the leaf structure (i.e., the high-frequency pattern). The rightmost image, on the contrary, creates a texture that is very similar to the source, but does not resemble the target linear pattern. Thus, a good selection for a patch size is to make it proportional to the size of a high-frequency pattern that one wants to preserve in the input source texture, while keeping it as small as possible. Note that the patch size does not affect the low-frequency symmetry pattern of the resulting texture as long as it is small enough so that pattern is achievable.

The as-rigid-as-possible deformation algorithm relies upon a set of evenly spaced points to control the deformation, and selecting the number of control points affects the flexibility of the texture perturbation model, which in turn affects the results of our algorithm: More control points allows greater local distortion, which is helpful for matching the target, but may introduce unwanted distortion artifacts in the result. For an example of this effect, please consider Figure 15, which shows the results of using the as-rigid-as-possible deformation model with different numbers of control points for symmetrization of the input image shown on the top left. Note that the flexibility provided by more control points allows better symmetrization at the cost of small local distortions (blue arrows). For the image warping examples in Figures 5, 9 and 7, we allowed a user to select between 8 and 128 regularly spaced control points, depending on the size of features to be preserved in the source image. Selecting the optimal number and position of control points automatically is an interesting problem, which is beyond the scope of this article.

Optimization. We use a simple steepest descent search optimization procedure for all applications. This procedure is likely to find a local minimum, and thus a perturbation model could generate a texture that better matches the desirable pattern. Exploration of other optimization procedures with more sophisticated strategies can be a topic for future work.

Timing. Computing the translational symmetry transform for a 250 by 250 image takes 43 milliseconds, and a low-dimensional perturbation model such as in Figure 8 converges in about 50 iterations, or 5 seconds. Texture synthesis results such as in Figures 1, 3, and 6 with 512 by 512 pixels were produced with 14000 iterations, or 3000 seconds. Figures 5, and 9, and 7, with as-rigid-as-possible deformation took up to 20000 iterations, or 1000 seconds.

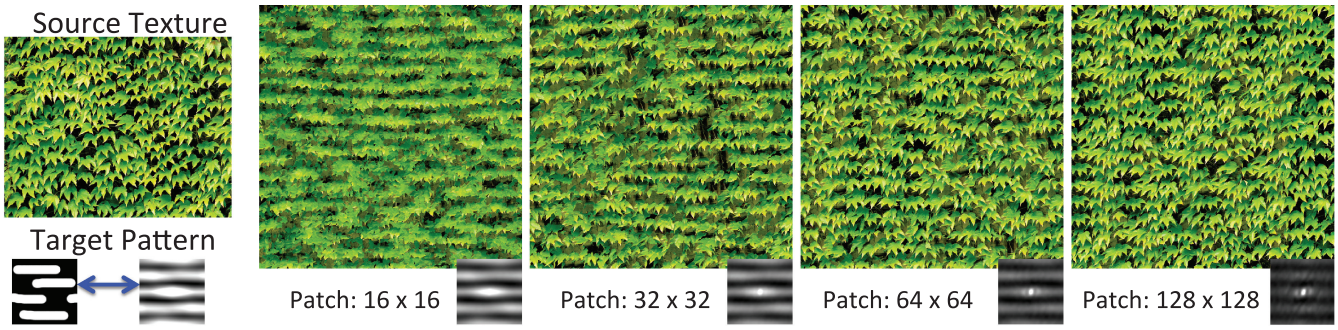

Fig. 16. Influence of patch size on a resulting texture. As patch size gets smaller the space of valid textures increases, thus resulting a solution closer to a desired symmetry; as patch size increases the solution has less of desired target pattern, but texture quality increases since textures better resemble the source.

These compute times are practical for offline applications like the ones shown in the article, but they would have to be accelerated significantly for interactive scenarios. In certain cases, faster methods are clearly available. For example, for some symmetry representations, it is possible to map image-space parameters directly to the objective function defined in symmetry space and then use gradients to accelerate the optimization. Also, in some cases, symmetry representation can be updated incrementally during the optimization, which would speed the computation by an order of magnitude for most of the examples in this article. These optimizations were not included in our implementations, which were aimed at generality rather than efficiency.

9. CONCLUSION AND FUTURE WORK

In this article, we have proposed a framework for symmetry-guided synthesis and processing of textures. Our investigation explored three different kinds of symmetry-based guidance (transfer, processing, and optimization), three symmetry representations (translational, rotational, and planar reflection), and two texture perturbation models (texture quilting and image warping). Example results demonstrate that these methods can be applied to a variety of problems including texture synthesis with symmetry transfer, symmetry transfer for images, symmetry-space filtering of textures, and image and texture symmetrization.

In our work we only studied a small space of possible applications within the proposed framework. In the future research it would be interesting to apply our framework to perform other image processing tasks like retargeting or in-painting while preserving original symmetries. Another possible direction is to augment our texture perturbation model that mainly deals with geometric deformations with a method to change the intensity and color pallete of pixels. This would allow modeling illumination and color irregularity (e.g., as in Liu et al. [2004]). Another possible application can arise from extending our work to geometry processing applications using 3D symmetry representations and perturbation models (e.g., mesh deformation). Investigating these and other variations of the framework are good topics for future work.

ACKNOWLEDGMENTS

We thank creators of CMU NRT database [2012], and the following Flickr users for allowing us to use their images as input to our system: dolescum [2012], 100kr [2012], snappa2006 [2012], shallowend24401 [2012], euart [2012], David Brennan (davidbrennan [2012]), cwazymandy [2012], John Perivolaris (dr_john2005 [2012]), Martin Heigan (martin_heigan [2012]), and Christopher Lewis (cloois [2012]). We acknowledge James Hays, Minwoo Park, and Yanxi Liu for distributing their code and providing suggestions for comparison to lattice-based methods.

REFERENCES

100KR. 2012. http://www.flickr.com/photos/100kr/209708058/.

Bar-Joseph, Z., El-Yaniv, R., Lischinski, D., and Werman, M. 2001. Texture mixing and texture movie synthesis using statistical learning. *IEEE Trans. Vis. Comput. Graph. 7,* 2, 120–135.

Bonneh, Y., Reisfeld, D., and Yeshurun, Y. 1994. Quantification of local symmetry: Application to texture discrimination. *Spatial Vis. 8,* 4, 515–530.

Brennan, D. 2012. http://www.flickr.com/photos/davidbrennan/251080600/.

Chetverikov, D. 1995. Pattern orientation and texture symmetry. *Comput. Anal. Images Patterns 970.*

Cwazymandy. 2012. http://www.flickr.com/photos/cwazymandy/3938576605/.

Database, C. N. 2012. http://vivid.cse.psu.edu/texturedb/gallery/.

Dolescum. 2012. http://www.flickr.com/photos/dolescum/4399058804/.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann.

Efros, A. A. and Freeman, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the SIGGRAPH'01 Conference.*

Efros, A. A. and Leung, T. K. 1999. Texture synthesis by nonparametric sampling. In *Proceedings of the International IEEE Conference on Computer Vision (ICCV'99).*

Euart. 2012. http://www.flickr.com/photos/euart/282152062/.

Golovinskiy, A., Podolak, J., and Funkhouser, T. 2009. Symmetry-aware mesh processing. In *Proceedings of the Mathematics of Surfaces Conference*.

Hays, J. H., Leordeanu, M., Efros, A. A., and Liu, Y. 2006. Discovering texture regularity as a higher-order correspondence problem. In *Proceedings of the European Conference on Computer Vision.*

Heeger, D. J. and Bergen, J. R. 1995. Pyramid-Based texture analysis/synthesis. In *Proceedings of the SIGGRAPH'95 Conference.*

Heigan, M. 2012. http://www.flickr.com/photos/martin_heigan/2352361336/.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. 2001. Image analogies. In *Proceedings of the SIGGRAPH'01 Conference*.

Igarashi, T., Moscovich, T., and Hughes, J. F. 2005. As-Rigid-As-Possible shape manipulation. *ACM Trans. Graph. 24*, 3.

KAZHDAN, M., CHAZELLE, B., DOBKIN, D., FINKELSTEIN, A., AND FUNKHOUSER, T. 2002. A reflective symmetry descriptor. In *Proceedings of the European Conference on Computer Vision (ECCV'02).*

KAZHDAN, M., CHAZELLE, B., DOBKIN, D., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2003. A reflective symmetry descriptor for 3D models. *Algorithmica.*

KAZHDAN, M., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2004. Symmetry descriptors and 3D shape matching. In *Proceedings of the Symposium on Geometry Processing (SGP'04).*

KELLY, M. F. AND LEVINE, M. D. 1995. Annular symmetry operators: A method for locating and describing objects. In *Proceedings of the International Conference on Computer Vision (ICCV'95).*

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. In *Proceedings of the SIGGRAPH'05 Conference.*

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of the SIGGRAPH'03 Conference.*

LEUNG, T. AND MALIK, J. 1996. Detecting localizing and grouping repeated scene elements from an image. In *Proceedings of the European Conference on Computer Vision (ECCV'96).*

LEWIS, C. 2012. http://www.flickr.com/photos/cloois/17435429/.

LIU, Y., LIN, W.-C., AND HAYS, J. H. 2004. Near regular texture analysis and manipulation. *ACM Trans. Graph. 23*, 1.

MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. In *Proceedings of the SIGGRAPH'05 Conference.*

MITRA, N. J., GUIBAS, L., AND PAULY, M. 2007. Symmetrization. In *Proceedings of the SIGGRAPH'07 Conference.*

PARK, M., BROCKLEHURST, K., COLLINS, R. T., AND LIU, Y. 2009. Deformed lattice detection in real-world images using mean-shift belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Trans. Graph. 27.*

PERIVOLARIS, J. 2012. http://www.flickr.com/photos/dr_john2005/211195030/.

PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S., AND FUNKHOUSER, T. 2006. A planar-reflective symmetry transform for 3D shapes. In *Proceedings of the SIGGRAPH'06 Conference.*

PORTILLA, J. AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vis.*

REISFELD, D., WOLFSON, H., AND YESHURUN, Y. 1995. Context-Free attentional operators: The generalized symmetry transform. *Int. J. Comput. Vis.*

SHALLOWEND24401. 2012. http://www.flickr.com/photos/shallowend24401/295133809/.

SNAPPA2006. 2012. http://www.flickr.com/photos/snappa2006/2106318872/.

TSAI, R. Y. 1986. An efficient and accurate camera calibration technique for 3D machine vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'86).*

TURINA, A., TUYTELAARS, T., AND GOOL, L. V. 2001. Efficient grouping under perspective skew. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01).*

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. Eurographics State of the Art report.

WEI, L.-Y. AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the SIGGRAPH'00 Conference.*

XU, K., COHNE-OR, D., JU, T., LIU, L., ZHANG, H., ZHOU, S., AND XIONG, Y. 2009. Feature-Aligned shape texturing. In *Proceedings of the SIGGRAPH'09 Asia Conference.*

ZABRODSKY, H., PELEG, S., AND AVNIR, D. 1995. Symmetry as a continuous feature. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 12.

Received June 2010; revised November 2011; accepted January 2012