Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

Texture Splicing

Yiming Liu[1,2†] Jiaping Wang[2] Su Xue[2,3†] Xin Tong[2] Sing Bing Kang[4] Baining Guo[2]

[1]Nanyang Technological University
[2]Microsoft Research Asia
[3]Yale University
[4]Microsoft Research Redmond

Abstract

We propose a new texture editing operation called texture splicing. For this operation, we regard a texture as having repetitive elements (textons) seamlessly distributed in a particular pattern. Taking two textures as input, texture splicing generates a new texture by selecting the texton appearance from one texture and distribution from the other. Texture splicing involves self-similarity search to extract the distribution, distribution warping, context-dependent warping, and finally, texture refinement to preserve overall appearance. We show a variety of results to illustrate this operation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Texture I.4.7 [Image Processing and Computer Vision]: Feature Measurement—Texture

1. Introduction

Textures are fundamental in computer graphics, and the topic of texture synthesis has been well-explored. The typical goal of texture synthesis is to reproduce *existing* textures in different sizes, forms, and contexts (e.g., [HB95, Deb97, EL99, EF01, KSE*03, ZZV*03, KEBK05]). Interestingly, relatively little has been done on texture editing, where new textures are created by modifying existing ones.

Textures are recognizable not only by the appearance of its basic elements (textons), but also their placement distribution. The placement distribution may include translation, rotation, and scaling of textons. Example-based texture synthesis generates textures by mimicking the texton and its placement distribution simultaneously (without separating the two properties). As a result, it is difficult to generate new textures with different attributes using current example-based texture synthesis algorithms.

In this paper, we propose a new texture editing operation which we call *texture splicing*. Unlike texture synthesis, texture splicing decomposes textures into two parts: texton appearance and placement distribution. Given two textures, texture splicing generates a new texture by using the texton appearance from one texture and distribution from the other. This operation is especially useful when a database of textures is available, in which case the user only need to specify which texton appearance and distribution are desired—these attributes can be inherited from different textures. An interesting special case is *self-splicing*. In this case, the two input textures are the same and the user has to modify the distribution manually to generate new textures. Figure 1 shows a new texture created using texture splicing.

2. Related Work

Most existing approaches for texture synthesis are pixel-based [HB95, Deb97, EL99] or patch-based [EF01, KSE*03, ZZV*03, KEBK05]. While they are capable of producing compelling-looking results, they do not explicitly separate the texton and its distribution. As a result, it is not apparent how they can generate new textures with separate attributes.

Techniques for texture editing have also been proposed, for example, Texture Transfer [EF01], Image Analogies [HJO*01], and Texture Flow [KEBK05, LH06, OiAIS09]. Fundamentally, these approaches generate a new output by transferring a given texture to an image or video;

† Yiming Liu and Su Xue were visiting students at Microsoft Research Asia.

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

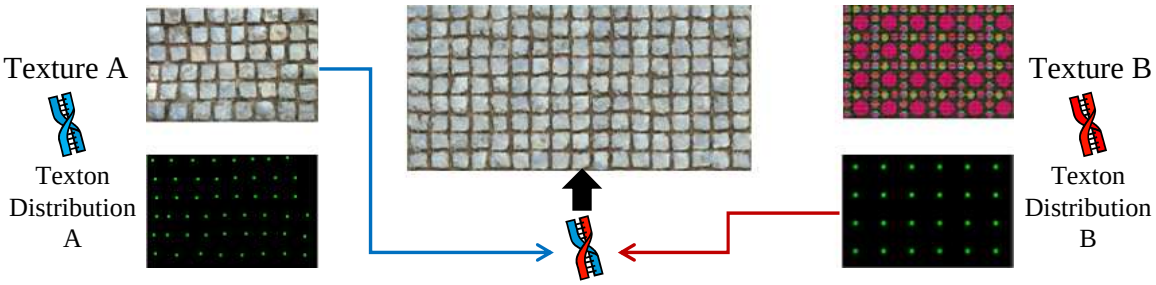

Figure 1: *A new texture is generated by combining textons from texture A with the placement distribution from texture B.*

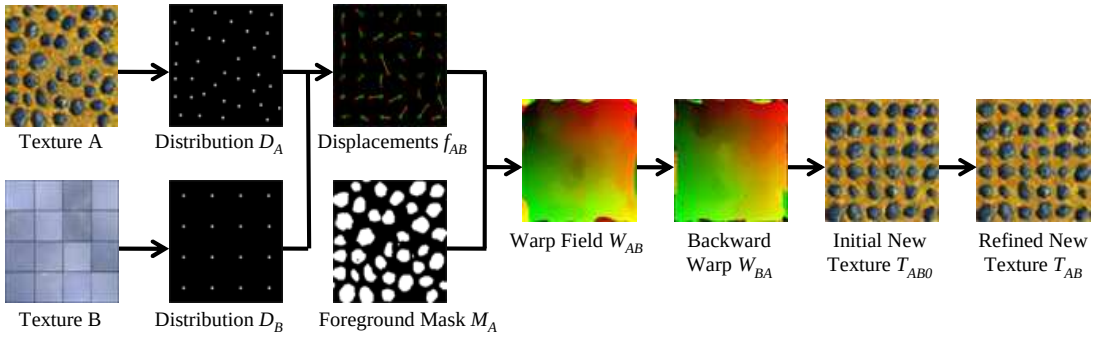

Figure 2: *Overview of texture splicing.*

however, the new output basically retains the properties of the original texture.

There are techniques that do model the texton [BA89, TTG01,LF06] and its spatial distribution [LLH04,HLEL06]. Explicit modeling of such texture properties provides more flexibility for texture editing; it allows one property to be modified while preserving the other. The technique of [LDR09] uses manifold diffusion distance to separate the different texture contents in an image. It then generates new textures by manipulating the decomposed parts and recomposing. However, as we shall see, there are issues with current approaches that limit their use.

In [BD02], a self-similarity based method is proposed to propagate the editing operation on a texton to the neighborhood regions that are self-similar. Examples of such editing operations include expanding, color editing, and warping. Matusik *et al.* [MZD05], on the other hand, use a more global model. They represent the space of textures from a database by a simplicial complex, where each vertex represents a texture. Oriented edge features in each texture are used to compute warping fields and establish similarity; similar textures are interconnected in the complex. New textures are generated by nonlinear interpolation. Basically, the new texture is a morphed version of two closest textures in the complex. Other than specifying the path in the complex, the user has little other control over the texture to be generated.

The technique of [LH05] allows the user to use circular regions to manually specify position of texture textons. However, the synthesized textons are possible to adhere partial regions of their neighboring ones when they lie close in the source texture. Furthermore, since the foreground/background area is not distinguished explicitly, the user needs to manually delete textons on the synthesized texture.

There are many approaches for texton extraction [VP88, LM96, SZ99, TTG01, LF06, AT07], but it is still an open problem. These approaches do not address the issue of texture synthesis. Liu *et al.* [LLH04] propose a texture editing and analysis scheme of near-regular textures by having the user specify the distribution. In a followup work, Hays *et al.* [HLEL06] propose a technique to automatically extract the texton distribution of near-regular textures through regularized thin-plate spline warping. However, the methods of [LLH04, HLEL06] are limited to topology-regular distributions. Dichler *et al.* [DMLC02] propose a method to automatically extract the distribution by simple RGB quantization. However, this method cannot work well for textures with connected elements or textures with closely adjacent elements (e.g., brick wall textures).

Gal *et al.* [GSCO06] propose a feature-sensitive inhomogeneous texture mapping method. This method maps textures according to a warping function W while preserving

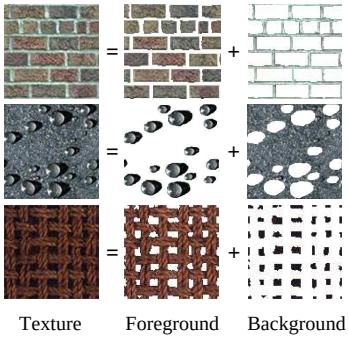

Figure 3: *Examples of textures and their foregrounds and backgrounds.*

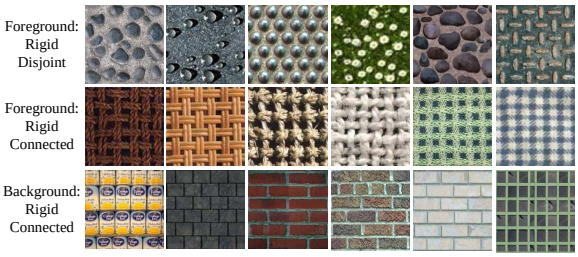

Figure 4: *Tags based on property of foreground or background, with representative examples.*

features. The user is required to provide a feature mask and manually specify the input warping function W.

Our texture splicing technique relies on a database of textures from which the user chooses to produce a new texture. Prior to its use, however, the texture database is preprocessed first.

3. Texture Database

We preprocess each texture in our database to generate the following information: foreground/background types (either connected or disjoint) T_F, T_B, texton foreground (binary) mask $M(x,y)$, and the spatial distribution $D = \{\mathbf{p}_i, i = 1,...,N_D\}$. D lists the 2D coordinates $\mathbf{p}_i$ associated with the placement of textons; N_D is the number of textons in the sample texture. Note that the database preprocessing phase is not completely automatic.

3.1. Foreground and Background Type

For a given texture, the extent and appearance of a texton may be spatially varying. Given a texton, we define its foreground and background areas (their segmentation is described in Section 3.3). The definitions are somewhat arbitrary, but generally the foreground is used to designate a structure within the texton that either is visually prominent or occupies a large central area. The background is typically used to designate areas that are almost uniform. Figure 3 shows three different textures with their predefined foregrounds and backgrounds.

Once the decision is made as to which is the foreground and which is background, they are tagged as either *rigid connected* (c) or *rigid disjoint* (d); the tagging process is done manually. As the tag names imply, the foreground is connected ($T_F = c$) if it is connected *across* neighboring textons, and disjoint otherwise ($T_F = d$). For the background, it can only be tagged as connected ($T_B = c$). The term "rigid" is used to indicate that during texture synthesis, the foreground or background is not to be distorted. Figure 4 shows examples of textures tagged in this manner. Note that it is sufficient to tag a texture as $T_F = c$, $T_F = d$, or $T_B = c$.

The next two steps are self-similarity search (to estimate the texton spatial distribution) and foreground-background separation. Both rely on a manual initialization step. We manually select one foreground area (from which we can compute its center $\mathbf{p}_e$ and size s_e) and scribble on the background. This lets the system know the approximate shape of the foreground part of the texton as well as the color distributions of the background and foreground.

3.2. Self-Similarity Search

The step of self-similarity search is required to estimate the spatial distribution of texton placement. We use the appearance vector map [LH06] to accomplish this. This map encodes neighborhood information of a texture. The channels in the appearance vector map reveal spatial structure that arises from factors such as color, intensity, edge orientation, and patterns at different scales. From experiments, we found it is adequate to use the first 8 channels of the appearance vector map for this search step.

Our algorithm searches for self-similar locations as follows: For the pixel location $\mathbf{p}_e$ (manually initialized) and its neighborhood $N_a(\mathbf{p}_e)$ in channel a of the appearance vector map, we define a similarity distribution map $D_{pe,a}$ with respect to pixel $\mathbf{p}_e$ and its neighborhood as $D_{pe,a}(x,y) = G(N_a(\mathbf{p}_e), N_a(x,y))$, where $G(.,.)$ is a similarity measure between two neighborhoods (we use normalized cross-correlation [Lew95]). The similarity distribution map $D_{pe,a}(x,y)$ indicates the likelihood of neighborhood $N_a(\mathbf{p}_e)$ appearing at location (x,y) in channel a. The similarity distribution map is computed for $a = 1,...,8$; the actual map D_{pe} used is the one with minimum entropy.

The local maxima in D_{pe} are potential candidates for texton locations. They are then sorted according to the similarity value, from highest to lowest. To determine the actual locations, our algorithm applies non-maximal suppression on the candidates one by one, in the order they were sorted. Non-maximal suppression is done by removing neighboring pixels of the current candidate. The size of the neighborhood

is determined by s_e, which was produced in the manual initialization step. Our algorithm stops when 95% pixels on the similarity distribution map are either picked or removed.

3.3. Foreground-background Separation of Textons

At this point, we have the following information: foreground mask of one texton and scribbles on the background (from manual initialization), and spatial distribution D_{pe} (described in the previous section). Thus, we have samples of foreground colors $C(F)$ and background colors $C(B)$. Note that F and B are the *sets* of foreground and background colors, respectively.

We use a graph-cut segmentation algorithm [GPS89] to extract the texton foreground (binary) mask. To segment the foreground and background in texture A, we minimize

$$E(L_A) = \sum_{\mathbf{p}\in A} D_{\mathbf{p}}(L(\mathbf{p})) + \sum_{\mathbf{p}\in A}\sum_{\mathbf{q}\in N_4(\mathbf{p})} S(L(\mathbf{p}),L(\mathbf{q})), \quad (1)$$

where $L_A = \{L(\mathbf{p}) \in \{B,F\} | \mathbf{p} \in A\}$ is a labeling of A, $L(\mathbf{p})$ is the label (either B, for background, or F, for foreground) at pixel $\mathbf{p}$, and $N_4(\mathbf{p})$ is the 4-neighborhood of $\mathbf{p}$. $D_{\mathbf{p}}(\cdot)$ is the data term associated with labeling costs, while $S(L(\mathbf{p}),L(\mathbf{q}))$ is the regularization term that prefers spatial smoothness of labels. They are defined as

$$D_{\mathbf{p}}(L(\mathbf{p})) = -\min_{c\in C(L(\mathbf{p}))} \|A(\mathbf{p}) - c\|^2 \text{ and} \quad (2)$$

$$S(L(\mathbf{p}),L(\mathbf{q}))) = \delta(L(\mathbf{p}) - L(\mathbf{q}))e^{-\|A(\mathbf{p})-A(\mathbf{q})\|^2/2\sigma^2}, \quad (3)$$

where $\delta(L(\mathbf{p}) - L(\mathbf{q}))$ is 1 if $L(\mathbf{p}) \neq L(\mathbf{q})$, 0 if $L(\mathbf{p}) = L(\mathbf{q})$. Note that in (2), $C(L(\mathbf{p})) = C(F)$ or $C(B)$, both of which are known from the manual initialization and self-similarity steps. In our experiments, $\sigma = 1$. The energy minimization problem is modeled as a min-cut/max-flow problem on a flow network G, and solved using a standard technique [FF62, EK72, BK04].

Given the database with the preprocessed information, we can now proceed to generate new textures. From this point on, all the steps associated with texture splicing are automatic.

4. Texture Splicing

Our technique is summarized in Figure 2. Given source textures A and B from the database, a new texture is generated by setting B's distribution D_B as the target distribution for source texture A. A set of discrete displacements f is established to warp A's distribution D_A to the target distribution D_B. A dense forward warping field W is computed using f. Depending on the foreground/background tag, W is computed under the constraint of selective rigidity [GSCO06]. For example, if the tag is set to $T_F = d$, the shapes of the foreground elements are to preserved as much as possible. Finally, the new texture is generated through barycentric interpolation, with non-rigid regions refined using conventional texture synthesis.

4.1. Correspondence of Placement Distributions

Given source distribution D_A and target distribution D_B, we seek the mapping $f_{AB} : D_A \to D_B$. First, we normalize the scales of D_A and D_B such that the average distances to k-nearest positions of both distributions are unity (k is between 4 and 6, depending on texture density). We then align them through their centroids. We also align the orientations using the Radon transform [JK05].

A cost matrix C_{AB} is defined as 2D distances between points in D_A and D_B. f_{AB} is computed by minimizing the total cost of corresponding (unique) pairs:

$$f_{AB} = \arg\min_{f:D_A\to D_B} \sum_{i\in D_A} C_{AB}(i, f(i)). \quad (4)$$

We use the classical Hungarian method [Mun57] to find the global minimum.

4.2. Context Dependent Deformation

Given input texture A, foreground binary mask M_A, and displacements f_{AB}, we would now like to compute the dense warping field W_{AB}. W_{AB} maps a point $\mathbf{p} = (p_x, p_y)$ in A to another point $\mathbf{q} = (q_x, q_y)$ in B, i.e., $\mathbf{q} = W_{AB}(\mathbf{p})$.

Let us define the shift $\Delta(\mathbf{p}) = (\Delta_x(\mathbf{p}), \Delta_y(\mathbf{p})) = W_{AB}(\mathbf{p}) - \mathbf{p}$; we formulate the problem as a minimization of deformation energy. Since deformation on a 2D plane is separable, without loss of generality, we describe how we extract the x-component of the deformation U_x (U_y is computed in the similar way). We minimize

$$E_x(\Delta_x) = \lambda_G \sum_{\mathbf{p}\in C_A} (\Delta_x(\mathbf{p}) - (f_x(\mathbf{p}) - p_x))^2 + \lambda_R \sum_{\mathbf{p}|M_A(\mathbf{p})=1} \left(\Delta_x(\mathbf{p}) - \sum_{\mathbf{q}\in N_4(\mathbf{p})} \tilde{u}_{pq}\Delta_x(\mathbf{q})\right)^2, \quad (5)$$

with $N_4(\mathbf{p})$ being the 4-neighborhood of $\mathbf{p}$ and $\tilde{u}_{pq}$ is a normalized quantity described shortly. Each term with x as its subscript refers to its x-component. The first term tends to preserve displacements f_{AB} while the second term encourages spatial smoothness of displacements. Since we prefer to preserve f_{AB}, we set $\lambda_G \gg \lambda_R$ (more specifically, to 10^4 and 10^2, respectively).

Note that the second term is computed over pixels that ought to be rigid only (i.e., where $M_A(\mathbf{p}) = 1$). The definition of which pixel should be rigid depends on the type, as indicated in Table 1. This enables the shape of specific parts of the texture to be preserved while the other parts can be distorted. $\tilde{u}_{pq}$ is the normalized weight of each neighboring pixel: $\tilde{u}_{pq} = u_{pq} / \sum_{q\in N_4(p)} u_{pq}$, with u_{pq} computed as follows:

Type	$\mathbf{p} \in F$	$\mathbf{p} \in B$	λ_P	ε
$T_F = c$	$M_A(\mathbf{p}) = 1$	$M_A(\mathbf{p}) = 0$	1	10^{-3}
$T_F = d$	$M_A(\mathbf{p}) = 1$	$M_A(\mathbf{p}) = 0$	1	10^{-3}
$T_B = c$	$M_A(\mathbf{p}) = 0$	$M_A(\mathbf{p}) = 1$	0	1

Table 1: *Mask values based on tag. F and B are the sets of foreground and background pixels, respectively. Recall that T_F and T_B are the type of foreground and background, respectively; c refers to rigid connected while d refers to rigid disjoint. Note that the case $T_B = d$ is not used.*

Figure 5: *Four special conditions on the foreground mask M_A, any of which yields $u_{\mathbf{pq}} = M_A(\mathbf{q}) + \varepsilon$. Each block represents a pixel. Black: $M_A(.) = 0$, white: $M_A(.) = 1$, grey: $M_A(.) = 1$ or 0 ("don't care").*

- $u_{pq} = M_A(\mathbf{q}) + \varepsilon$ if any of these conditions are met: $p_x \neq q_x$, or $M_A(\mathbf{p}) = M_A(\mathbf{q}) = 0$, or any one of the four conditions illustrated in Figure 5.
- Otherwise, $u_{pq} = \lambda_P(M_A(\mathbf{q}) + \varepsilon)$.

The parameter ε controls the consistency of foreground and background deformations, while λ_P controls the constraint between horizontal and vertical borders of rigid areas. Both depend on the texture tag (see Table 1). For textures with a rigidly connected background ($T_B = c$), we set $\lambda_P = 0$ to decouple constraints on the horizontal and vertical borders. The seams for the red brick texture in Figure 7 are preserved because of this constraint decoupling. Although the bricks are moved and scaled, their basic shapes are preserved. For other types of textures, $\lambda_P = 1$.

Δ can be obtained using a sparse equation solver. In practice, similar as [GSCO06], we use double back substitution to solve it. The warping field can then be computed: $W_{AB} = I + \Delta$, where I is the identity function. W_{AB} typically maps to fractional pixel locations; we use barycentric interpolation to extract color or mask value.

More specifically, we first triangulate the source texture, using the center of pixels as vertices. We warp each triangle (see Figure 6) and rasterize it on the output texture. For each node $\mathbf{q}$ inside a warped triangle, we use barycentric coordinates to determine the inverse mapping $W_{BA}(\mathbf{q})$. It is possible for a point in the output to be inside multiple warped source triangles. In this case, we select the triangle whose vertices have a maximum sum of texton mask values. Once the backward mapping W_{BA} has been established, we use it to produce the initial output texture T_{AB0}.

4.3. Final Refinement

There are two basic problems with T_{AB0}: First, W_{BA} may not be defined for all pixels in the output texture—this results in holes. Second, rigid parts of the source texture may still be slightly deformed, with attendant loss of visual fidelity.

We adopt Lefebvre and Hoppe's texture synthesis method [LH06] to handle these two problems. Suppose $W_{BA}(\mathbf{q})$ does not exist, which creates a hole at $\mathbf{q}$. We remove the hole by randomly selecting a non-rigid pixel $\mathbf{p}$ in the source texture, i.e., $W_{BA}(\mathbf{q}) = \mathbf{p}$. This produces a hole-free texture T'_{AB0}. We then apply the two-level multi-resolution synthesis on T'_{AB0}, and jitter the reference positions of non-rigid areas. This produces the final new texture T_{AB}.

5. Results

Figures 7 and 8 show a variety of results using our texture splicing technique. Figures 7 shows results with different texture types providing the texton appearance. (It is irrelevant what the texture type is for the texture providing the distribution information.) Notice that the resulting textures preserve either the foreground or background (depending on the texture tag) of one texture while assuming the distribution of the other texture.

Figures 8, on the other hand, shows all possible textures that can be obtained from textures in the leftmost column providing the texton appearance and textures in the top row providing the distribution. This matrix is very telling—while most results look reasonable, others show that not all texture pairs are compatible. For example, the apple texture (8th row inside the matrix) and water droplet texture (10th column inside the matrix) produced a bizarre and unattractive texture. While this can be regarded as a "failure" case, it is not clear what the right answer is given these two textures as input.

Self-splicing results are shown in Figures 9. Here, only one source texture is used to create a result; the new texture is generated by directly manipulating its distribution. The manipulation can be in form of scaling (1), randomization (2), arrangement into a regular pattern (3), or specific rearrangement (4).

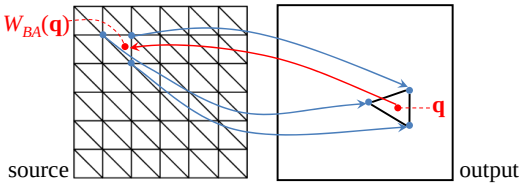

Figure 6: *Illustration for backward interpolation. The blue points in the output texture are the warped vertices of a triangle. The backward warping of $\mathbf{q}$ is done using barycentric coordinates.*

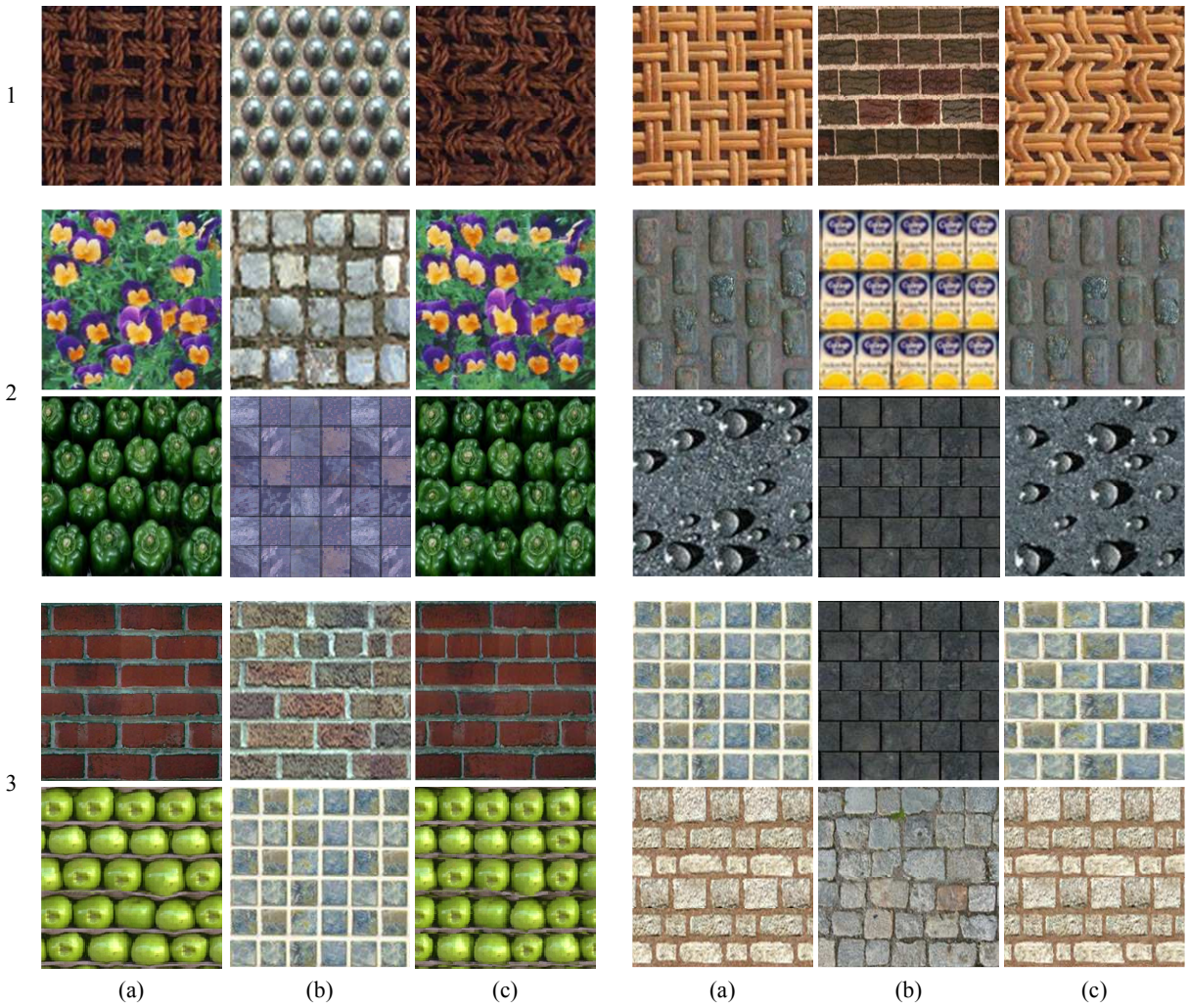

Figure 7: *Synthesized textures. The groups 1 (rigid connected foreground, or $T_F = c$), 2 (rigid disjoint foreground, or $T_F = d$), and 3 (rigid connected background, or $T_B = c$), are in reference to the source textures providing the texton appearance (a). Combined with source textures providing the distribution information (b), we get target textures (c).*

The pre-computation step of *k*-nearest-neighbor search to build candidate sets for *k*-coherence search of texture synthesis took on average 21.1 seconds for textures of size 128×128 and 33.6 seconds for 192×192. This timing is not critical to our texture splicing operation, since this pre-computation step constitute a once-only cost and can be done offline.

Texture splicing and self-splicing took on average 1.4 seconds for 128×128 textures and 2.9 seconds for 192×192 textures on a workstation with an Intel Core 2 Duo 2.66GHz CPU and 2GB RAM. The most expensive operation (taking $> 95\%$ of the time) is the Cholesky factorization of the left-hand matrix for deformation and texture synthesis. Since the factorization does not depend on the texton displacement, it can be pre-computed once and added to the texture database (currently not done). At present, our texture synthesis operation runs on the CPU. It is amenable to GPU implementation, which is likely to allow our tool to run at interactive speeds.

6. Concluding Remarks

We have proposed *texture splicing* as a novel means for texture editing. The idea is simple: decompose each texture into texton appearance and its spatial distribution, and combine the texton appearance from one texture with the distribution from another. This makes texture editing extremely simple for the user, as only two textures need to be specified. The user has the option of directly editing the texton placements,

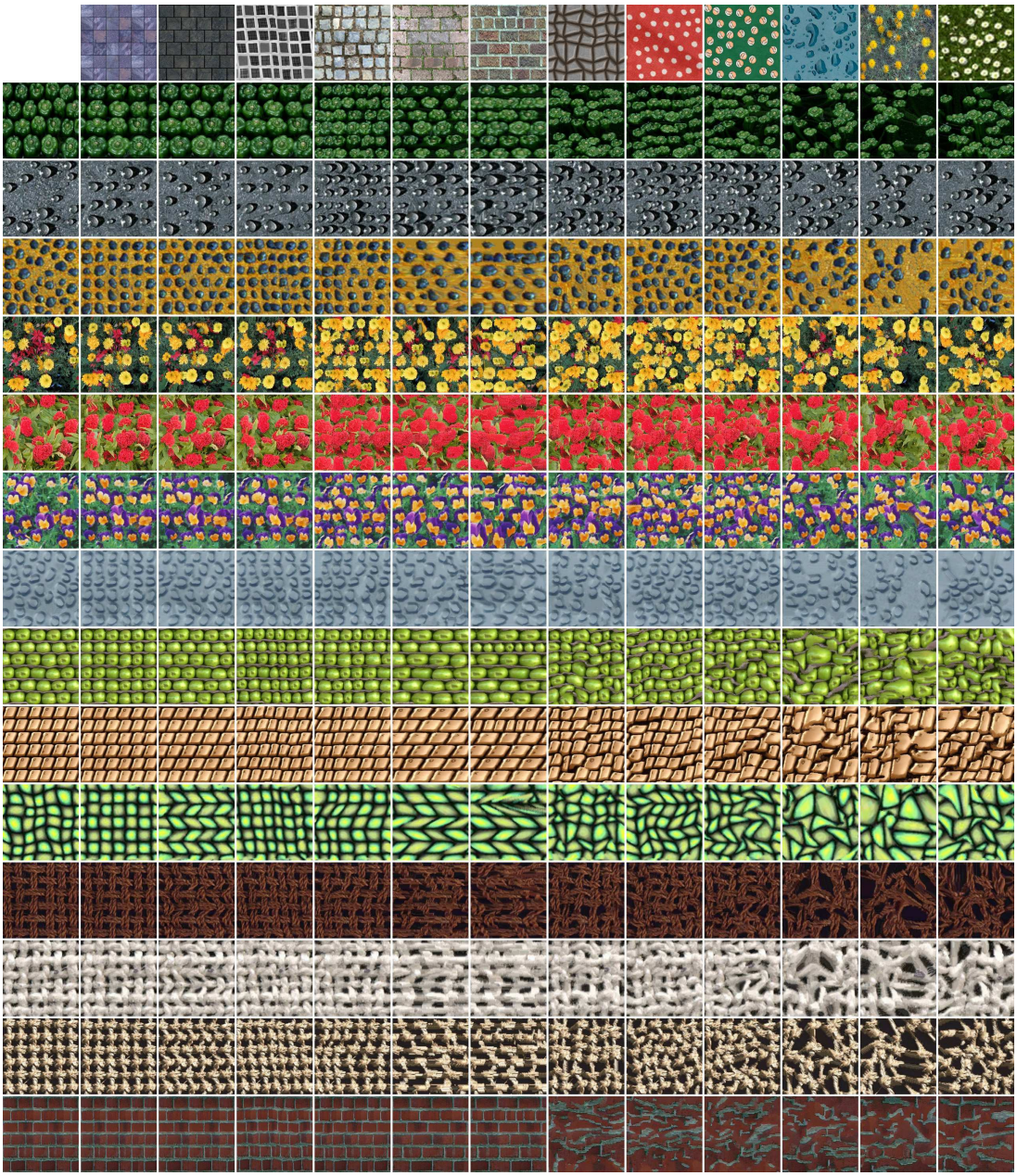

Figure 8: *Edited textures over combinations of source textures. First column: source textures providing texton appearance. First row: source textures providing distribution information. The texture within the matrix is the synthesized result using a source texture from the first column and another from the first row.*

which require more interaction, but provides more control on texture design. We showed a variety of results to demonstrate the effectiveness of texture splicing.

There are limitations in our current work. The self-similarity search does not take account texton rotation and scale. In addition, our method relies on the user-specified $\mathbf{p}_e$ to initialize the self-similarity search. Future extensions include fully automatic texton distribution extraction algorithm and handling of textures with a variety of texton scale and rotation.

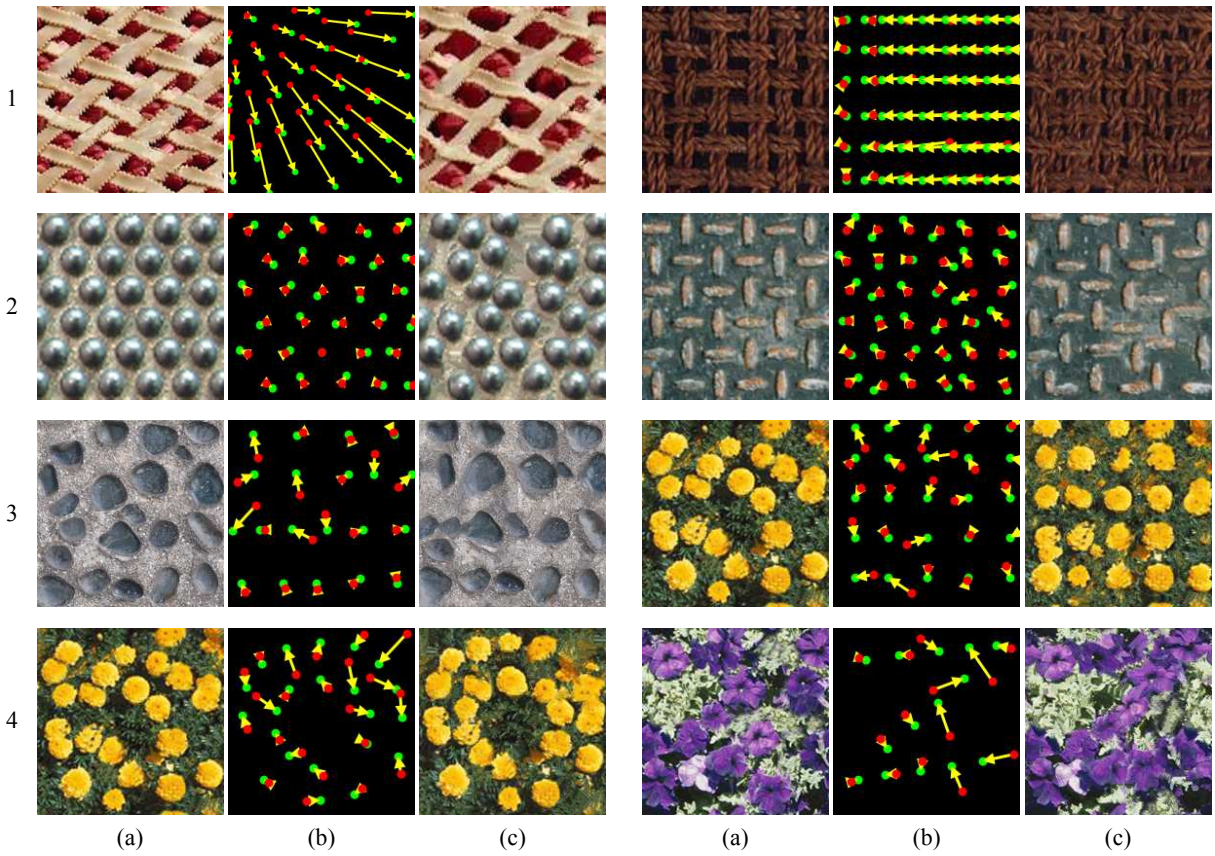

Figure 9: *Texture self-splicing results: (a) source texture, (b) manipulation of distribution, and (c) edited texture.*

References

[AT07] AHUJA N., TODOROVIC S.: Extracting texels in 2.1D natural textures. In *International Conference on Computer Vision (ICCV)* (2007). 2

[BA89] BLOSTEIN D., AHUJA N.: Shape from texture: Integrating texture-element extraction and surface estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 12 (1989), 1233–1251. 2

[BD02] BROOKS S., DODGSON N.: Self-similarity based texture editing. *Computer Graphics (SIGGRAPH '02 Proceedings)* (2002), 653–656. 2

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137. 4

[Deb97] DEBONET J. S.: Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of ACM SIGGRAPH 97* (1997), 361–368. 1

[DMLC02] DICHLER J.-M., MARITAUD K., LEVY B., CHAZANFARPOUR D.: Texture particles. In *Eurographics conference proceedings* (Sep 2002). 2

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001* (August 2001), 341–346. 1

[EK72] EDMONDS J., KARP R. M.: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM 19*, 2 (1972), 248–264. 4

[EL99] EFROS A. A., LEUNG T.: Texture synthesis by non-parametric sampling. *International Conference on Computer Vision* (1999), 1033–1038. 1

[FF62] FORD L., FULKERSON D.: *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962. 4

[GPS89] GREIG D., PORTEOUS B., SEHEULT A.: Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological) 51*, 2 (1989), 271–279. 4

[GSCO06] GAL R., SORKINE O., COHEN-OR D.: Feature-aware texturing. In *Proceedings of Eurographics Symposium on Rendering* (2006), pp. 297–303. 2, 4, 5

[HB95] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis synthesis. In *Proceedings of ACM SIGGRAPH 95* (August 1995), pp. 229–238. 1

[HJO*01] HERTZMANN A., JACOBS C., OLIVER N., CURLESS B., SALESIN. D.: Image analogies. *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), 327–340. 1

[HLEL06] HAYS J. H., LEORDEANU M., EFROS A. A., LIU Y.: Discovering texture regularity as a higher-order correspondence problem. *ECCV 2* (2006), 522–535. 2

[JK05] JAFARI-KHOUZANI K.: Radon transform orientation estimation for rotation invariant texture analysis. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 6 (2005), 1004–1008. Sr. Member-Soltanian-Zadeh„ Hamid. 4

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

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003* (2003), 277–286. 1

[LDR09] LU J., DORSEY J., RUSHMEIER H. E.: Dominant texture and diffusion distance manifolds. *Comput. Graph. Forum 28*, 2 (2009), 667–676. 2

[Lew95] LEWIS J.: Fast normalized cross-correlation. *Vision Interface* (1995), 120–123. 3

[LF06] LOBAY A., FORSYTH D.: Shape from texture without boundaries. *International Journal of Computer Vision 67*, 1 (2006), 71–91. 2

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Transactions on Graphics, SIGGRAPH 2005 24*, 3 (2005), 777–786. 2

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Transactions on Graphics, SIGGRAPH 2006* (2006), 541–548. 1, 3, 5

[LLH04] LIU Y., LIN W.-C., HAYS J.: Near-regular texture analysis and manipulation. *ACM Transactions on Graphics, SIGGRAPH 2004 23*, 3 (2004), 368–376. 2

[LM96] LEUNG T. K., MALIK J.: Detecting, localizing and grouping repeated scene elements from an image. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I* (London, UK, 1996), Springer-Verlag, pp. 546–555. 2

[Mun57] MUNKRES J.: Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics 5*, 1 (1957), 32–38. 4

[MZD05] MATUSIK W., ZWICKER M., DURAND F.: Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics, SIGGRAPH 2005* (2005), 787–794. 2

[OiAIS09] OKABE M., ICHI ANJYO K., IGARASHI T., SEIDEL H.-P.: Animating pictures of fluid using video examples. *Comput. Graph. Forum 28*, 2 (2009), 677–686. 1

[SZ99] SCHAFFALITZKY F., ZISSERMAN A.: Geometric grouping of repeated elements within images. *In Shape, Contour and Grouping in Computer Vision LNCS 1681*, 1-2 (1999), 165–181. 2

[TTG01] TURINA A., TUYTELAARS T., GOOL L. V.: Efficient grouping under perspective skew. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2001), 247–254. 2

[VP88] VOORHEES H., POGGIO T.: Computing texture boundaries from images. *Nature 333* (1988), 364–367. 2

[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, SIGGRAPH 2003* (2003), 295–302. 1