Volume 0 (1981), Number 0 pp. 1–10

Material Space Texturing

Nicolas Ray[1] and Bruno Lévy[1] and Huamin Wang[2] and Greg Turk[2] and Bruno Vallet[1]

[1]ALICE, Inria, France [2]Georgia Institute of Technology, USA

Abstract

Many objects have patterns that vary in appearance at different surface locations. We say that these are differences in materials*, and we present a* material-space *approach for interactively designing such textures. At the heart of our approach is a new method to pre-calculate and use a 3D texture tile that is periodic in the spatial dimensions (s,t) and that also has a material axis along which the materials change smoothly. Given two textures and their feature masks, our algorithm produces such a tile in two steps. The first step resolves the features morphing by a level set advection approach, improved to ensure convergence. The second step performs the texture synthesis at each slice in material-space, constrained by the morphed feature masks. With such tiles, our system lets a user interactively place and edit textures on a surface, and in particular, allows the user to specify which material appears at given positions on the object. Additional operations include changing the scale and orientation of the texture. We support these operations by using a global surface parameterization that is closely related to quad re-meshing. Re-parameterization is performed on-the-fly whenever the user's constraints are modified.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.6 [Computer Graphics]: Methodology and Techniques

Introduction

Many natural and human-made objects have slowly changing variations in texture over their surface. For instance, many animals have fur that changes from one pattern to another across their body, and different parts of buildings may show variations in their degree of weathering based on their exposition to wind, rain and sun. We use the term *material* to describe the changes in the nature of a texture as it varies across an object. We present a system that allows a user to easily create such spatially-varying textures.

Central to our work is the new idea of a *material-space tile* that is represented by a 3*D* texture where (s,t) are the usual spatial coordinates, and an extra dimension m represents the variation of the material (see figure 1, next page). These tiles are periodic in the (s,t) dimensions, that is, they wrap seamlessly left-to-right and top-to-bottom. The pattern of the texture varies smoothly along the material dimension m of a tile. Such tiles are generated by a new method based on level set advection that creates high quality morphs with smooth transitions between different textures.

Using these material-space tiles, our system allows the user to specify constraints on texture orientation, scale and material that are interpolated over a surface mesh. As the tiles are periodic, the texture coordinates (s,t) of a surface point can also be referred by any $(s,t)+(p_s,p_t)$ with $(p_s,p_t)\in\mathbb{Z}^2$. This allows us to use a Periodic Global Parameterization (PGP) [RLL*06] steered by the interpolated orientation and scale instead of a texture atlas [MYV93]. Based on this approach, only a single material-space tile needs to be stored with modest memory costs, and the texture distortion remains low even for surfaces of arbitrary genus.

The main contributions of our work are:

- ◇ the introduction of a new representation of spatially-varying material that is encoded in a tileable 3D texture. Our representation is simple, compact and can be interactively mapped onto a surface using simple "brushing" or "combing" interaction metaphor;
- ◇ the automatic generation of these 3D tiles by a new method of morphing between *texture features* that is based on level set advection, and enhanced to ensure convergence;
- ◇ the constrained texture synthesis method that generates texture with a given feature mask, and still preserves the color histograms.

In the rest of the paper presents a review of previous works, introduces our material-space 3*D* tiles synthesis algorithm

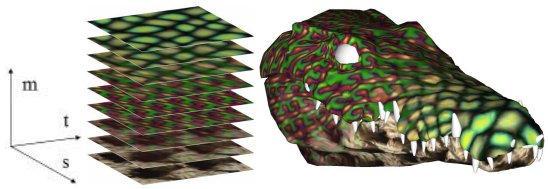

Figure 1: *The third axis (m) of the 3D texture (left) makes it possible to easily encode material variations over the surface (right).*

(Section 1), then explains how to interactively map it on the surface (Section 2) and render the result (Section 3).

Previous work

In this section we review some of the literature in texture synthesis and parameterization that is most closely related to our work.

Texture Synthesis on Surfaces Texture synthesis addresses the following problem: given a reference image, generate colors on the whole object that has the same appearance as the reference image. Many approaches for performing texture synthesis on surfaces have appeared in the literature. Some of these approaches are point-at-a-time synthesis methods [Tur01, WL01, YHBZ01, ZZV*03], and thus require additional mapping or re-sampling for rendering. In contrast, there are few techniques that solely use the original texture image during rendering, and perform references into this image using texture mapping hardware. One of the earliest such approaches is the synthesis method of Soler *et al.* [SCA02], in which they created patches from clusters of triangles. Magda *et al.* [MK03] use a preprocessing step to optimize the assignment of (s,t) coordinates to each triangle. Zelnika and Garland [ZG04] also extended their Jump Maps to be used with per-vertex (s,t) coordinates that reference the original texture. Lefebvre and Hoppe [LH06] use hardware to perform an additional stage of memory indirection to do texture lookups, and their synthesis method makes use of the GPU to provide interactive synthesis rates. The high quality of their synthesis results is at least partly due to the texton masks (more on this below) and PCA analysis of pixel neighborhoods.

Texture Morphing There have been several approaches taken to the problem of creating a morph between two or more textures. Bar-Joseph *et al.* [BJEYLW01] performed texture analysis using wavelets. Although this paper only created 50/50 blends of textures, the method could probably be adapted to create full texture morphs. Ziqiang Liu *et al.* [LLSY02] let users specify landmarks between two textures to establish feature correspondence, and then use this to create a warp field. They created a texture morph by warping and cross-dissolving. Zhang *et al.* [ZZV*03] used texton masks to identify features in textures, and produced a mixed feature mask using blending and binary thresholding. This new mask was then used to guide per-pixel texture synthesis to generate a spatially-varying blend between textures. Although they did not demonstrate general texture morphing, the near-regular texture analysis method of Yanxi Liu *et al.* [LLH04] could be used for aligning texture features for morphing. Matusik *et al.* [MZD05] created user-identified feature maps to produce warps between textures. They improved on the standard cross-dissolve by performing histogram matching to keep the features more sharply resolved. A similar idea is used by Qing Wu *et al.* [WY04] for improving texture synthesis by texture charts warping when the mask contains curvilinear thin features.

Aperiodic Tiling Several texture synthesis approaches have used the idea of pre-computing one or more texture tiles. Jos Stam [Sta97] demonstrated that a group of 16 such tiles could break up the periodicity of a pattern if the tiles are made to match across their edges. Neyret and Cani [NC99] extended this idea to triangulated surfaces, and used a set of as few as four triangular tiles. Cohen *et al.* [CSHD03] used the notion of pre-computed Wang tiles to create large textured regions that have no obvious repetitions to the final patterns.

Parameterization Various researchers have presented methods for the automatic generation of (s,t) coordinates for a mesh. This is the problem of *mesh parameterization*, and we recommend Floater's survey [DFS05] on the topic. In our tile-based approach, two aspects of parameterization are important: taking *constraints* into account and defining seamless (s,t) coordinates *globally* on objects of any topology.

Constrained parameterization enables a user to specify a set of positional constraints interpolated by the mapping. Lévy [Lev01] has proposed a solution based on a gradient preserving energy and Kraevoy *et al.* [KSG03] use a coarser mesh to deform an existing unconstrained parameterization. In our context, the periodic and high-frequency natures of the signal make theses approaches inappropriate. We are more interested in constraining the anisotropy (orientation and scale) of the material on the surface.

Global parameterization : Gu *et al.* [GY03] create global conformal maps that have a minimal number of singularities, at the price of introducing high distortion. Ray *et al.* [RLL*06] introduce PGP (Periodic Global Parameterization), that has the reverse trade-off (lower distortion and more singularities). Tong *et.al* [TACSD06] propose a manual method to specify an homology basis and deduce a periodic method. PGP was extended [LRL06] to take constraints into account. Since PGP automatically generates a map invariant up to a grid-preserving transformation with reasonable distortion, it is well-suited to periodic texture mapping. In our texture mapping context, we will show how PGP can be simplified.

1. Material-Space Synthesis

The objective of this section is to produce $3D$ textures of material-space (s,t,m) such that all iso-m, denoted as T_m, are tiles. We first define the extremes iso-m (for $m=0$ and $m=M$) by taking two input textures I_0 and I_M, and generating

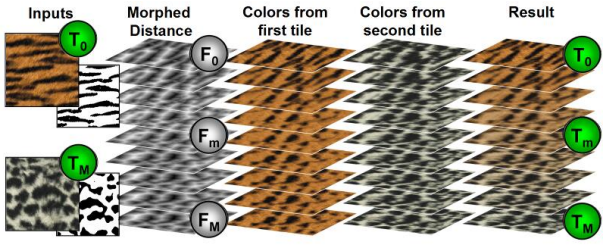

Figure 2: ***Overview of the parametric tiles synthesis :*** *The signed distance to feature is interpolated from the tile T_0 to the T_N, to give constraint masks. Then, for each layer, a tile is synthesized from the corresponding interpolated mask and from T_0. The same processing is applied starting from T_N. Finally, the two resulting 3D tiles are blended to smoothly morph from T_0 to T_N.*

tileable versions of them T_0 and T_M. This can be done by using the graph-cut algorithm [KdE*03]. Now, our goal is to generate high-quality texture tiles for our real-time system by synthesizing new intermediate textures T_m, $m \in [1, M-1]$. In other words, T is a $3D$ texture parameterized by $2D$ texture coordinates $\mathbf{s} = (s,t)$ and by a material coordinate m $(0 \leq m \leq M)$.

In order to take the structure of the texture patterns into account during the material transition between key texture samples, our synthesis algorithm is based on texture features (texton masks) as suggested in [ZZV*03]. Texture features, stored as binary masks F_m , can either be painted by the user or constructed automatically using image gradients. As illustrated in figure 3, this texture morphing based on texture synthesis allows to handle more significant structure variations than with warping approaches [LLSY02, MZD05]. Starting from the two input tiles, T_0 and T_M, and their binary feature masks, F_0 and F_M, we use the following pipeline (see Figure 2) to create material-space tiles:

1. Create signed distance fields $\phi_0(\mathbf{s})$ and $\phi_M(\mathbf{s})$ for the feature masks F_0 and F_M using Yamashita *et al.*'s algorithm [YI86].

2. Create intermediate feature masks $F_m(\mathbf{s})$ (along the m axis) by morphing between the feature mask distance fields (see 1.1).

3. For each layer m: synthesize a new tile from the texture T_0 where the feature mask is constrained to be F_m (see 1.2), do the same operation based on the texture T_M, then create the final colors T_m by blending the synthesis results from both input tiles using m/M as the blend coefficient (see Figure 5).

We will first describe our advection approach to morphing between feature masks (Section 1.1), and then turn to the issue of generating the colors of the material-space accordingly (Section 1.2).

Figure 4: *Result of our feature mask morphing algorithm. Gray areas corresponds to the sweeping process.*

1.1. Texture Feature Morphing

A key aspect when creating 3D tiles is to morph the patterns of the first texture into the patterns of the second texture. In our case, as patterns are represented by feature masks, this problem turns into finding intermediate feature masks $F_m(\mathbf{s})$, $m \in [1, M-1]$. This is usually done by a level set approach where the masks F_0 and F_M are converted into signed distance fields ϕ_0 and ϕ_M, such that the morphing problem becomes to construct a smooth transition from ϕ_0 to ϕ_M. A straightforward way to formulate the transition is to linearly interpolate the level sets $(\phi_m = (1 - m/M)\phi_0 + m/M\phi_M)$. This idea has been used for 2D and 3D morphing of shapes [PT92, COSL98] and enhanced in [TO99] where $\Phi(\mathbf{s}, m) = \phi_m(\mathbf{s})$ is found as the solution of a variational problem in 3D. Unfortunately, these methods do not consider the feature integrity and work well only when most features are nearly aligned. For this reason, we will propose a morphing method based on feature advection [Set99, OF02] where the level set ϕ is advected by a velocity field and enhanced by interior dilation and sweeping that ensure the convergence.

Advection: The idea here is to advect F_0 by $\nabla\phi_M$. This process pushes the features of the mask F_0 (called *source* features, where $\phi_0 > 0$) to the features of the mask F_M (called *target* features, where $\phi_M > 0$). Because the final number of feature masks M is usually small, we get better precision by *oversampling* the material space for the advection of the distance function, i.e. choosing a number of advection steps $N >> M$. Thus after advecting, material space will be *subsampled* by choosing a proper correspondence between the distance functions ϕ_n, $n \in [0, N]$ and the feature masks F_m, $m \in [0, M]$. In our experiments, standard advection suffers from four limitations:

1. Once a source feature starts overlapping a target feature, it does not fill it/align with it properly. Defining a proper advection inside the F_M-features is tricky and not robust in general.

2. Some target features might not even be reached by an advected source feature.

3. Around sources of the advection field (at maximum distance to target features), the value of the advected distance function ϕ tends to zero, so it whether if these areas should be inside or outside the feature mask is undefined.

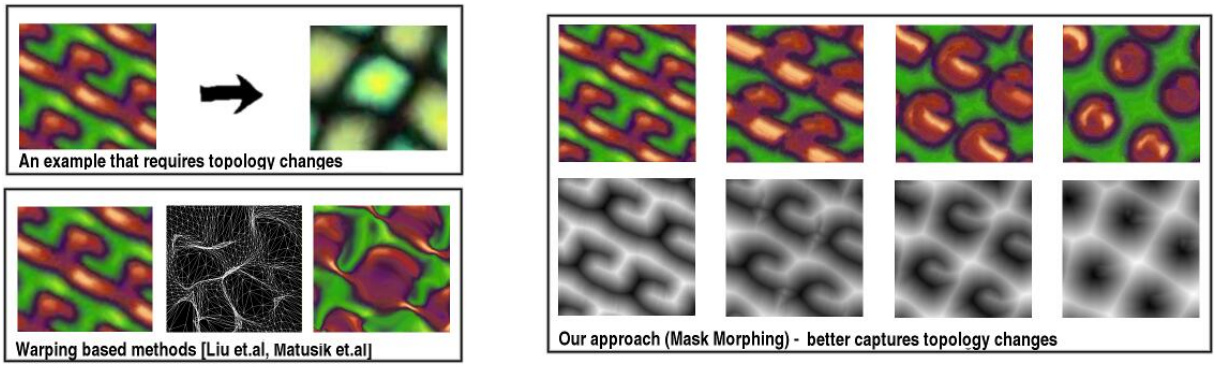

Figure 3: *Our mask-morphing method (top) allows to change the features' topology without introducing stretch, while classic warping (bottom) cannot change the topology and may introduce a lot of stretch.*

4. The source features are transformed into the target features at a very irregular speed.

We will overcome limitation 1. by *interior dilation*, limitation 2. and 3. by *sweeping*, and limitation 4. by *speed adaptive subsampling* as detailed in the following Texture Feature Morphing algorithm description (see Figure 4):

- ◇ *Material space oversampling*: we start by choosing a number of advection steps $N >> M$ in order to accurately simulate the advection. We then define a step length $dt = \min_{\mathbf{s}} \phi_N(\mathbf{s})/N$ to ensure that after N advection steps, even the source features which were the furthest from target features will have been advected far enough.
- ◇ *Enhanced advection:*
  - ◇ Advection: the source features are pushed toward target features by an advection field. This is done by advecting ϕ outside F_M ($\phi_N < 0$) along the field $\nabla\phi_N(\mathbf{s})$. As ϕ consists of one real value associated with each texel, we displace each texel of coordinate $\mathbf{s}$ to $adv(\mathbf{s}) = \mathbf{s} + \nabla\phi_N(\mathbf{s})dt$ and rasterize it on the texel grid: $\phi_{i+1}(\mathbf{s}') = \sum_{\mathbf{s}} overlap(\mathbf{s}', adv(\mathbf{s})).\phi_i(\mathbf{s})$ where $overlap(\mathbf{s}', \mathbf{s}) = sup(1 - |s - s'|, 0) + sup(1 - |t - t'|, 0)$ denotes the overlap ratio between a texel centered at $\mathbf{s}$ and a texel centered at $\mathbf{s}'$.
  - ◇ Interior dilation: we dilate the advected F_0-features inside the F_M-features. For all the neighbor texels $\mathbf{s}'$ of each texel $\mathbf{s}$: if $\mathbf{s}'$ is inside F_M-features (i.e $\phi_N(\mathbf{s}') > 0$) and $\mathbf{s}$ is inside an advected F_0-feature (i.e. $\phi_n(\mathbf{s}) > 0$), then $\phi_{n+1}(\mathbf{s}') \leftarrow \phi_n(\mathbf{s})$.
  - ◇ Sweeping: we have to ensure that, at the end of the process, the advected feature mask will corresponds to the target features. We do that by replacing $\phi_n(\mathbf{s})$ by $\phi_N(\mathbf{s})$ on the texels where $|\phi_0(\mathbf{s})| > (1 - n/N) sup_{\mathbf{s}'} |\phi_N(\mathbf{s}')|$. This process removes the features that will not be able to be advected to a target feature, and it progressively fills the target feature that may not by reached by any advected feature.
- ◇ *Speed-adaptative subsampling*: we extract a subsequence $\phi_{H(m)}(\mathbf{s})$ of level sets through a monotonous function $H : [0, M] \rightarrow [0, N]$ such that $H(0) = 0, H(M) = N$. We choose $H(m)$ for $m \in [1, M-1]$ such that the number of pixels for which $\phi_{H(m)}$ and $\phi_{H(m+1)}$ have opposite signs is the closest to constant. The feature masks $F_m(\mathbf{s})$ are then the positive sets of $\phi_{H(m)}(\mathbf{s})$.

Conclusion: Using this method, the feature masks may undergo topological changes such as splitting or merging, which produces visually pleasing results, close to what one would expect. Note however that our scheme is asymmetric with respect to the two textures (we are pushing the source features to the target features). Our advection is similar to the 3D shape metamorphosis method of Breen and Whitaker [BW01], with the important addition of interior dilation and sweeping. This is important for our particular problem, where one source feature may be pulled into more than one target feature. Finally our method is robust and automatic, requiring no user input for feature correspondences.

1.2. Constrained Tile Synthesis

For each layer, a morphed feature mask F_m has been produced (previous section), we now turn to the problem of generating two tiles having F_m as feature mask and the colors defined by the input textures T_0 (respectively T_N). Instead of working directly on colors, we simplify the upscaling in the

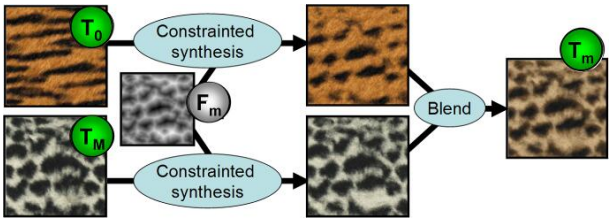

Figure 5: *Each layer T_m of a parametric pattern is found by blending two tiles. Each tile is obtained by texture synthesis based on colors from the corresponding input tile, but with the common constraint feature mask F_m.*

hierarchical synthesis process by using an indirection map I_m^r for each material m and resolution r.

That is, each pixel in the synthesized texture has $2D$ coordinates that refers to the input tile. Our synthesis algorithm is inspired by [LH06], but we have many differences that come form the constrained mask : it is impossible to paste large charts of the original image, it does not allow using K Nearest Neighbors and it requires to pay special attention to the conservation of the color histograms.

The first step of the synthesis pipeline repeatedly downscales the inputs (input color, input mask and interpolated feature mask) by a factor of 2 in order to create image pyramids for each of the inputs. Then, from coarsest ($r = 0$, typically 32×32) to finest ($r = R$, typically 256×256) resolution, the following process is applied:

Initialization: In this process, the initialization plays an important role since it ensures coherence between adjacent layers in m and also between resolutions r. The indirection map of the tiles is initialized as follows (recursion is initiated by the input image for which $r = m = 0$):

- $m = 0$ (the current layer is the first one): $I_0^r(i,j) \leftarrow (i\delta_r, j\delta_r)$ where δ_r is the pixel size at resolution r.
- $m > 0$, $r = 0$ (the resolution is the coarsest): the indirection map is initialized by a simple copy of the optimized texture coordinates of the previous layer: $I_m^0(i,j) \leftarrow I_{m-1}^0(i,j)$.
- $m > 0$, $r > 0$: the result of the coarser level is upsampled : $\forall (b_i,b_j) \in \{0,1\} \times \{0,1\}$, $I_m^r(2i+b_i, 2j+b_j) \leftarrow I_m^{r-1}(i,j) + (b_i\delta_r, b_j\delta_r)$

Optimization: The optimization repeats the following process several times (20 iterations for the coarsest resolution, and 3 iterations for the other resolutions): for each pixel (i,j) taken in random order, replace $I_m^r(i,j)$ by the coordinates of the pixel of the input tile that have the most similar neighborhood (a 3×3 neighborhood in our implementation). The comparisons of the neighborhoods are accelerated by a compression scheme similar to [LH06], and the kd-tree implementation of the ANN (Approximate Nearest Neighbor) library to find the best matching regions. To enhance the results, we penalize the input pixels according to the number of times they have already been used. This ensures that almost all pixels from the input tile are used, and in particular that the color histograms of the input and output tiles are similar.

In our experiments, the PCA reduces the neighborhood vectors to 8 coefficients, and the feature mask is scaled by two to ensure that the texture follows the mask. Compared with [LH06], the optimization process is slower because the constrained mask makes it impossible to paste large charts from the input image and to use kNN instead of ANN.

2. Interactive Design of the Mapping

Each material-space tile that is generated by our method is represented by a 3D texture, indexed by (s,t,m) coordinates, where (s,t) denote the tileable spatial coordinates, and where m denotes the material axis.

Our goal is now to provide the user with interactive tools to map the material-space onto an object, i.e. to define the (s,t,m) coordinates at each vertex of the surface that we want to texture. We provide to the user an interactive tool to constrain the characteristics of the texture at some points of the surface mesh:

- Orientation: two orthogonal unit vectors $\mathbf{d}_s^T$ and $\mathbf{d}_t^T$ attached to each triangle T,
- Scale: scalars S_i attached to each vertex i,
- Material: m_i coordinates attached to each vertex i,

As shown on Figure 7, each time the user constrains such a characteristic, the mapping is updated in 3 steps:

1. Compute an *effect area* around the constraint by a greedy algorithm (see subsection 2.1). All the subsequent steps will be limited to this effect area by locking all other vertices.
2. Interpolate the constraints (see subsection 2.2):
   - Constraints on *scale* S and *material* m are interpolated by a Laplacian smoothing algorithm.
   - Constraints on *orientation* are interpolated by a vector field smoothing algorithm that generates the two orthogonal unit vector fields $\mathbf{d}_s$ and $\mathbf{d}_t$, attached to the triangles of the surface.
3. Compute the (s,t) coordinates based on a simplified version of Periodic Global Parameterization (PGP) steered by both the scale S and the guidance vector fields $\mathbf{d}_s$ and $\mathbf{d}_t$ (see subsection 2.3).

2.1. Computing the effect area

When the user specifies or edits a constraint, we compute an effect area around the constraint (as in Kobbelt *et* al.'s work [KCVS98]). This effect area is determined by a classical greedy algorithm that visits the triangles in order of increasing distance to the nearest constraint. When the user defines an orientation constraint, triangles are iteratively added to the effect area when the user moves the mouse (see Figure 8).

The effect area is used for the interpolations as well as for the

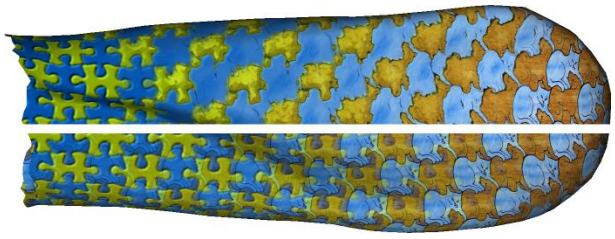

Figure 6: *The upper part of this bunny ear is rendered with intermediate layers generated by our system whereas the lower part is rendered using only a linear interpolation of the input tiles.*

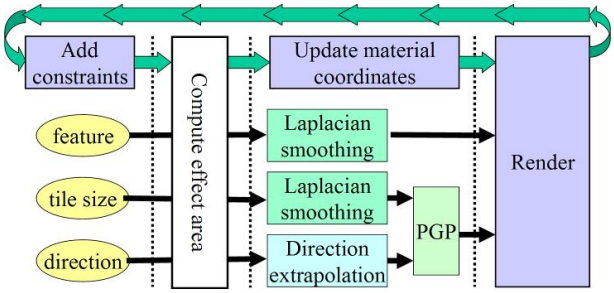

Figure 7: *Scheme of our interactive mapping method.*

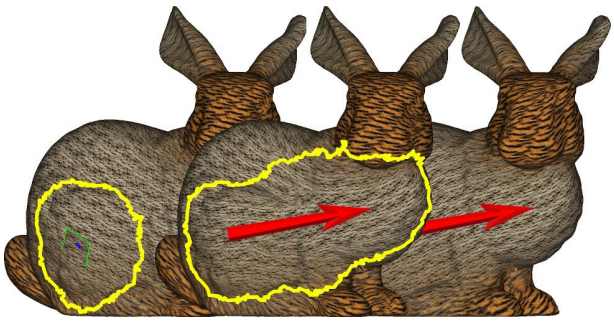

Figure 8: *Marking the effect area is like using a brush. Left: fix the size of the brush (border indicated by yellow); Middle: a stroke serves at both marking the area and specifying the orientation constraint (red arrow); Right: orientation constraint applied with an infinite effect area.*

mapping that are only updated on this region. The benefits are twofolds: it provides interactive design, and ensures that local editing will not affects other areas. To ensure continuity at the region boundary, all variables m, S, s, t associated to the boundary are fixed to their initial value. Since all these scalar fields are computed by solving linear system of equations, locking the boundary variables is simple [Lev05].

2.2. Constraints interpolation

We need to interpolate the user-defined constraints in material, scale and orientation. As scalars, the material m and the scale S are simply interpolated by Laplacian smoothing (see [NGH04] for instance) with locking user constraints. In some cases (see Figure 9) Laplacian smoothing might generate material values outside of the valid interval $[0,1]$, so the material m has to be clamped. The scale S should also be enforced to be greater than a constant $\varepsilon > 0$ to avoid invalid mappings.

The orientation interpolation requires some more care. Note that *orientations* rather than *vectors* need to be smoothly interpolated to define the field $\mathbf{d}_s^T$ on triangle T (the scale is interpolated separately). The method proposed in [RVLL08] allows a fine control of both the singularities and orientations. However, we preferred to allow for more intuitive interaction by using an algorithm that lets the singularities emerge from the geometric constraints [HZ00, RLL*06, LRL06] (we chose the latter for simplicity and efficiency). Finally, $\mathbf{d}_t^T$ is simply obtained from $\mathbf{d}_s^T$ by a $\pi/2$ rotation in the triangle T.

Figure 9: *A new material constraint is added to the dragon's tail (right) and is interpolated across the surface. The clamping ensures that the material m remains in a valid range on the dragon's head.*

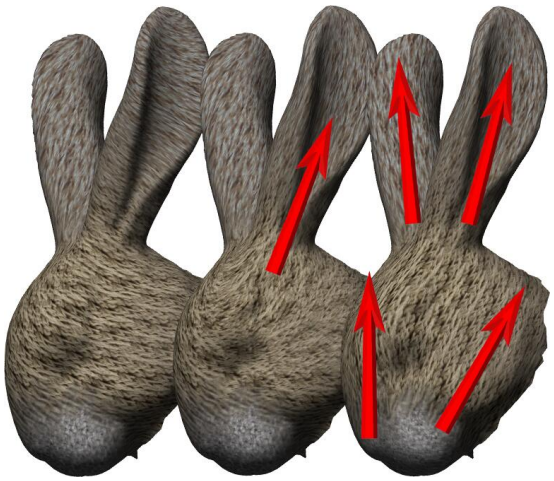

Figure 10: *Dragging the mouse on the bunny head (left) adds orientation constraints on the mouse path (red arrows). A small number of strokes suffices to orient the pattern on the whole model.*

2.3. Computing the (s,t) Coordinates

Our goal is now to generate texture coordinates s_i^T (the equations for t_i^T are identical) attached to each triangle corner (T,i) satisfying two constraints:

1. On triangle T, the tiles should be oriented by $\mathbf{d}_s^T$ and have size $S^T = (S_i + S_j + S_k)/3$. This can be written as: $\nabla s^T = \mathbf{d}_s / S^T = \mathbf{g}_s^T$ such that if we call $\vec{e}_{ij}$ the vector corresponding to the edge (i,j), we have:
$$s_j^T = s_i^T + \vec{e}_{ij}.\nabla s^T = s_i^T + \vec{e}_{ij}.\mathbf{g}_s^T \qquad (1)$$

2. All the corners incident to the same vertex should be

mapped to the same texel:

$$\exists p_i^T \in \mathbb{Z} \text{ such that } s_i^T = s_i^{T'} + p_i^T \quad \forall T, T' \supset i \qquad (2)$$

We can solve this global parameterization problem following [RLL*06], with some simplifications as in our tile-mapping context, singularities can be handled at rendering time and there is no swapping of texture coordinates. In short, (1) is averaged on the two triangles T and T' sharing (i,j), and by taking its cos and sin we get:

$$\mathbf{s}_j = \begin{bmatrix} \cos(\alpha_{i,j}) & \sin(\alpha_{i,j}) \\ -\sin(\alpha_{i,j}) & \cos(\alpha_{i,j}) \end{bmatrix} \mathbf{s}_i = R(\alpha_{i,j})\mathbf{s}_i \qquad (3)$$

where $\alpha_{i,j} = 2\pi\vec{e}_{ij}.(\mathbf{g}_s^T + \mathbf{g}_s^{T'})/2$ and $\mathbf{s}_i = [\cos(2\pi s_i^T), \sin(2\pi s_i^T)]$ implicitly satisfies the second constraint as we take the same variable for all T. Equation (3) can be satisfied in the least squares sense by minimizing:

$$F_{PGP}(\mathbf{s}) = \sum_{(i,j)} (A_T + A_{T'}) ||\mathbf{s}_j - R(\alpha_{i,j})\mathbf{s}_i||^2$$

where A_T is the area of triangle T. In the special case of initialization, we lock a single vertex $s_0^T = 0 \Leftrightarrow \mathbf{s}_0 = [1,0]$, else $\mathbf{s}$ is locked outside the effect area.

Finally, in each triangle $T = (i,j,k)$, the texture coordinates s_i^T, s_j^T, s_k^T are retrieved from $\mathbf{s}_i, \mathbf{s}_j, \mathbf{s}_k$:

$$\begin{array}{rcl} s_i^T & = & \mathrm{Arg}(\mathbf{s}_i)/2\pi \\ s_j^T & = & \mathrm{Arg}(\mathbf{s}_j)/2\pi + p_j^T \\ s_k^T & = & \mathrm{Arg}(\mathbf{s}_k)/2\pi + p_k^T \end{array} \qquad (4)$$

where $\mathrm{Arg}(\mathbf{s}_i)$ denotes the oriented angle between the vector $[1,0]$ and $\mathbf{s}_i$, and where the integer p_j^T minimizes $|(s_j^T - s_i^T) - \alpha_{i,j} + n_j^T|$.

3. Rendering

PGP generates a visually pleasing parameterization by relaxing deformations, but at the expense of introducing singular zones, where the mapping is invalid. This generates noticeable visual artifacts, concentrated in these zones (Figure 11). They correspond to triangles (i,j,k) for which the optimal mappings from the point of view of vertices i, j and k disagree. Fixing these artifacts is easy, using an approach that is similar to the method used to display jump maps [ZG04].

To detect the singularities, the algorithm analyzes the mapping for each vertex i and its neighbors $j_1, .. j_n$. The singular neighborhoods are characterized by two criteria, one geometric and one topologic. The neighborhood is singular:

- ◇ if it contains a triangle T that is highly distorted ($\|\nabla s - \mathbf{g}_s^T\|^2 > 0.5$ or $\|\nabla t - \mathbf{g}_t^T\|^2 > 0.5$)
- ◇ or if it contains an edge (i,j) for which the (s,t) coordinates mismatch: $s_i^T - s_j^T \neq s_i^{T'} - s_j^{T'}$ or $t_i^T - t_j^T \neq t_i^{T'} - t_j^{T'}$ where T, T' denote the two triangles shared by the edge (i,j).

For each invalid 1-ring neighborhood $i, j_1, j_2, ... j_n$, we recompute a set of texture coordinates $(s'_{i,i}, t'_{i,i}), (s'_{i,j_1}, t'_{i,j_1}), \ldots (s'_{i,j_n}, t'_{i,j_n})$, optimized from the point of view of vertex i, and defined by:

$$\begin{array}{rcl} (s'_{i,i}\ ,\ t'_{i,i}) & = & (s_i^T\ ,\ t_i^T) \\ (s'_{i,j}\ ,\ t'_{i,j}) & = & (s'_{i,i} + \alpha_{i,j}^s\ ,\ t'_{i,i} + \alpha_{i,j}^t) \end{array}$$

where $\alpha_{i,j}^s$ and $\alpha_{i,j}^t$ are given in Equation 3 and where T denotes one of the triangles incident to vertex i.

Each triangle $T = (i,j,k)$ now has three sets of texture coordinates (obtained from the neighborhoods of each vertex i,j,k). Then, we use them to perform three texture lookups. Finally, we linearly blend the obtained three colors using the barycentric coordinates in the triangle. This makes the texture lookup use the optimal coordinates at each vertex, and linearly interpolates the result between them. As shown in Figure 11, this hides both the poles (1) and T-Junctions (2,3) of the parameterization. Note that this can be also applied to additional attributes, such as normal maps, as done in the examples shown in Figure 13 (Right).

4. Results

The material-space tile synthesis algorithm produces satisfying results even when the input images are sightly different in terms of colors or feature shapes such as stripes, dots or more complex shapes (see figure 12). However, when the input tiles do not have well identified features, the morphing cannot outperform a simple fading between inputs.

We have developed an intuitive user interface to create and edit constraints for designing the (s,t,m) mapping. The user is provided with four different tools: the first one defines the area that will be affected by the other tools (determines the distance threshold used in subsection 2.1), the second and third ones pick a triangle and changes either its scale or its material parameter (see figure 9), and the last one (see figure 10) applies constraints to the guidance vector field by dragging the mouse over the model. Orientations are specified by the velocity vector of the mouse projected from 3D onto the model.

Our system enhances the visual richness of 3D objects at a low memory cost. Moreover, since our mapping system is totally disconnected from the material synthesis, it is easy to switch materials in real-time or to animate them. The parametric patterns can be used to morph one material into another as presented in section 1, but they can also be used for morphing with multiple key tiles. Such a morphing can be obtained by a simple concatenation of morphings between pairs of consecutive key tiles, and this yields a rich collection of patterns as shown in Figure 1. Another possibility is to use our system to hide the periodicity of a single tile. This can be done by generating a set of tiles sharing the same mask. Figure 14 shows how a 3D (s,t,m) texture with four layers suffices to remove the artifacts due to periodicity.

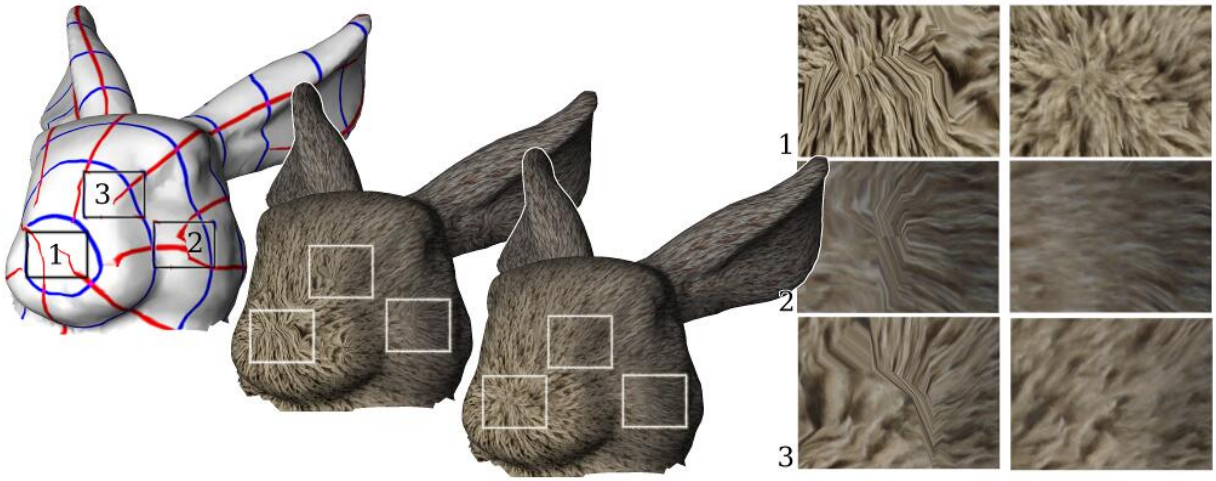

Figure 11: *The singularity (1) and T-junctions (2,3) can be fixed at rendering time by our method. Close-up images show the result without (left) and with (right) our singularity correction.*

Performance

Our system can be decomposed into three parts: material synthesis (precomputing), mapping (interactive design) and rendering. The performance of each part is given in table 1.

Texture synthesis: The texture synthesis takes 5 minutes for 8 layers at resolution 256×256 with 5×5 patch size for colors (reduced to a 5 dimensions vector), 3×3 patch size for feature (reduced to a 3 dimensions vector) and with 3 levels of resolution. Each pixel is updated 20 times for the coarser level and 4 times for the other levels. Most of the examples presented in the paper have 3 key image inputs, and such examples require synthesis of two morphs, giving a total time of 10 minutes.

Model	Dragon	Bunny	Dragon 2
# Triangles	11K	24K	50K
Interpolate orientation	1.2s	4.1s	8.9s
Initialize (u,v)	1.3s	3.7s	7.4s
Material constraint	0.5s	0.7s	1s
Orientation constraint	1s	2s	4s
Scale constraint	0.8s	1.7s	3.7s
Memory usage	3Mb	3Mb	3Mb

Table 1: *The first three rows give the timings for computing the initial material-space coordinates. The next three rows give the average time for interactions (ROI with approx. 25 percent of the model triangles). The last row gives the amount of graphic RAM used by our representation.*

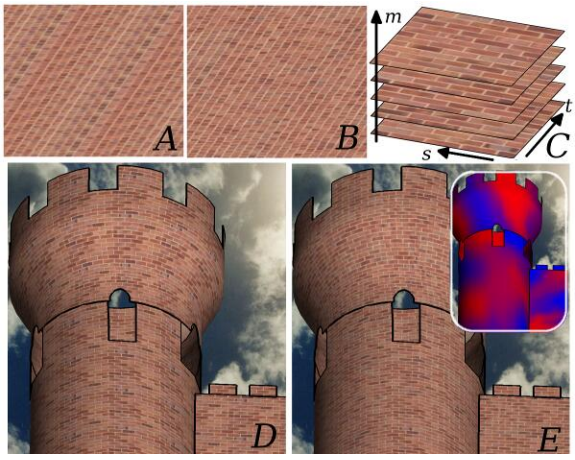

Figure 14: *A: when mapping a brick pattern in repeat mode, the tile is noticeable; B: our technique hides the periodicity by modulating the mapping along the extra "material" axis (m); C: four layers are sufficient to hide the pattern; We show another example mapped with the same tile, without (D) and with (E) our technique. The small image shows the "material" coordinate.*

Mapping design: The initialization of the (s,t) coordinates can take up to 30 seconds for models with approximately 50K triangles. Adding new constraints is then much faster thanks to the local updates. A typical effect area usually contains approximately 10K triangles, which allows the model to be updated interactively, as shown in the video.

Rendering: Our representation is especially well suited to interactive rendering on the GPU. In terms of computation, we only need one $3D$ texture lookup for non-singular regions and three $3D$ texture lookups around singularities (5 to 10 percent of the triangles). Using a normal map requires one additional lookup in the regular regions (and three lookups around singularities). In terms of memory usage, our representation is between 10 to 15 times smaller than an equivalent texture atlas. Using a texture atlas would require roughly 36 Mb to 48 Mb for our examples, and even more if we count the empty space lost in the atlas.

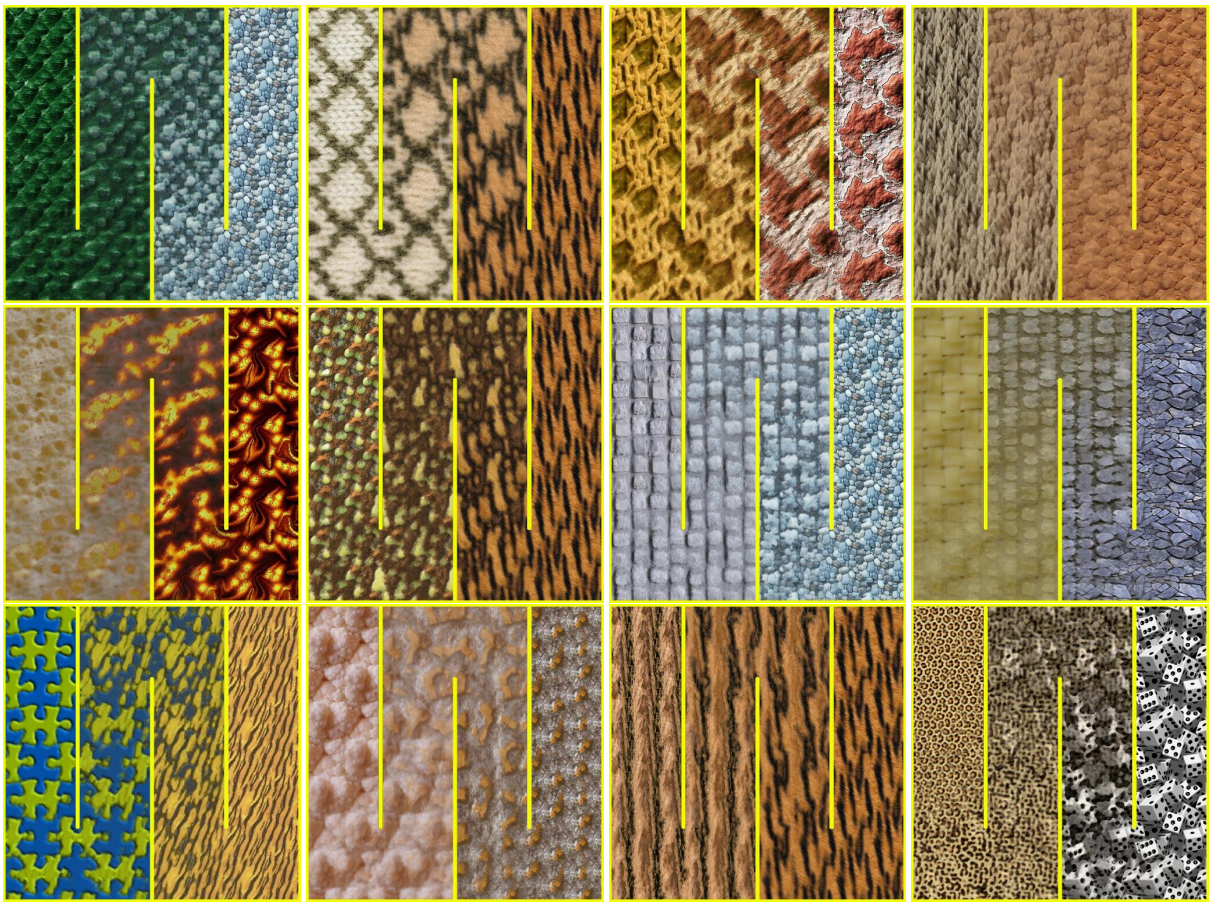

Figure 12: *The synthesis algorithm is able to morph between different type of feature mask (stripes, dots, others), but fails when meaningful features cannot be defined in the input image (lower right).*

Conclusion

We have presented a material representation that is simple, compact and independent from its mapping onto the object. This makes it possible to generate visually rich patterns at a minimal memory cost. These characteristics make it especially well suited for interactive user editing and for real-time rendering. The main limitation of our method is that a single material m axis is represented. In future work, we will investigate the possibility of creating and compressing 4D (or nD) tiles by using PCA. Moreover, our results could be improved by future advances in textures synthesis and by using richer material definitions (e.g. multiscale PRT).

References

[BJEYLW01] BAR-JOSEPH Z., EL-YANIV R., LISCHINSKI D., WERMAN M.: Texture mixing and texture movie synthesis using statistical learning. *IEEE TVCG 7*, 2 (2001). 2

[BW01] BREEN D. E., WHITAKER R. T.: Level-set approach for metamorphosis of solid models. *IEEE TVCG 7*, 2 (2001). 4

[COSL98] COHEN-OR D., SOLOMOVIC A., LEVIN D.: Three-dimensional distance field metamorphosis. *ACM TOG 17*, 2 (1998). 3

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *ACM TOG 22*, 3 (2003). 2

[DFS05] DODGSON N.-A., FLOATER M.-S., SABIN M.-A.: Surface parameterization: a tutorial and survey. *Advances in Multiresolution for Geometric Modelling* (2005). 2

[GY03] GU X., YAU S.-T.: Global conformal surface parameterization. In *SGP Proceedings* (2003). 2

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *SIGGRAPH Conf. Proc.* (2000). 6

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.: Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH Conf. Proc.* (1998). 5

[KdE*03] KWATRA V., DL A. S., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM TOG, SIGGRAPH 22*, 3 (July 2003). 3

[KSG03] KRAEVOY V., SHEFFER A., GOTSMAN C.: Matchmaker: constructing constrained texture maps. *ACM TOG 22*, 3 (2003). 2

Figure 13: *Three examples of our method applied to different models.*

[Lev01] LEVY B.: Constrained texture mapping for polygonal meshes. In *SIGGRAPH Conf. Proc.* (2001). 2

[Lev05] LEVY B.: Numerical methods for digital geometry processing. In *Israel Korea Bi-National Conference* (2005). 6

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM TOG 25*, 3 (2006). 2, 5

[LLH04] LIU Y., LIN W.-C., HAYS J.: Near-regular texture analysis and manipulation. In *ACM TOG, SIGGRAPH* (2004). 2

[LLSY02] LIU Z., LIU C., SHUM H.-Y., YU Y.: Pattern-based texture metamorphosis. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications* (2002). 2, 3

[LRL06] LI W.-C., RAY N., LEVY B.: Automatic and interactive mesh to t-spline conversion. In *EG/ACM SGP* (2006). 2, 6

[MK03] MAGDA S., KRIEGMAN D.: Fast texture synthesis on arbitrary meshes. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 82–89. 2

[MYV93] MAILLOT J., YAHIA H., VERROUST A.: Interactive texture mapping. In *SIGGRAPH Conf. Proc.* (1993). 1

[MZD05] MATUSIK W., ZWICKER M., DURAND F.: Texture design using a simplicial complex of morphable textures. In *ACM TOG, SIGGRAPH* (2005). 2, 3

[NC99] NEYRET F., CANI M.-P.: Pattern-based texturing revisited. In *SIGGRAPH Conf. Proc.* (1999). 2

[NGH04] NI X., GARLAND M., HART J. C.: Fair morse functions for extracting the topological structure of a surface mesh. In *SIGGRAPH Conf. Proc.* (2004). 6

[OF02] OSHER S. J., FEDKIW R. P.: *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002. 3

[PT92] PAYNE B. A., TOGA A. W.: Distance field manipulation of surface models. *IEEE Computer Graphics and Applications 12*, 1 (1992). 3

[RLL*06] RAY N., LI W. C., LEVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM TOG* (2006). 1, 2, 6, 7

[RVLL08] RAY N., VALLET B., LI W. C., LÉVY B.: N-symmetry direction field design. *ACM Trans. Graph. 27*, 2 (2008), 1–13. 6

[SCA02] SOLER C., CANI M.-P., ANGELIDIS A.: Hierarchical pattern mapping. In *SIGGRAPH Conf. Proc.* (2002). 2

[Set99] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999. 3

[Sta97] STAM J.: Aperiodic texture mapping. In *ERCIM Research Report R046* (1997). 2

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *SGP Proceedings* (2006). 2

[TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 335–342. 3

[Tur01] TURK G.: Texture synthesis on surfaces. In *SIGGRAPH Conf. Proc.* (2001). 2

[WL01] WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH Conf. Proc.* (2001). 2

[WY04] WU Q., YU Y.: Feature matching and deformation for texture synthesis. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 364–367. 2

[YHBZ01] YING L., HERTZMANN A., BIERMANN H., ZORIN D.: Texture and shape synthesis on surfaces. In *EG Workshop on Rendering Techniques* (2001). 2

[YI86] YAMASHITA M., IBARAKI T.: Distances defined by neighborhood sequences. *Pattern Recogn. 19*, 3 (1986). 3

[ZG04] ZELINKA S., GARLAND M.: Jump map-based interactive texture synthesis. *ACM TOG 23*, 4 (2004). 2, 7

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