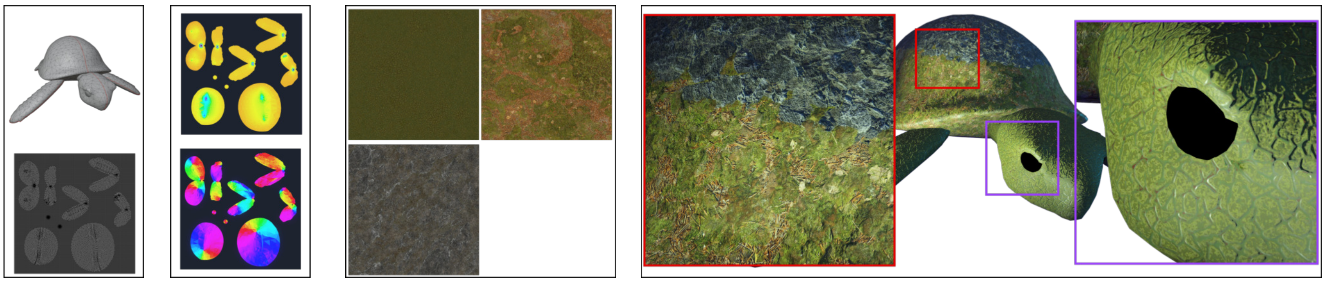
EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 2

Deformed tiling and blending: application to the correction of distortions implied by texture mapping

Q. Wendling[1], J. Ravaglia[1] and B. Sauvage[1]

[1]ICube, Université de Strasbourg, CNRS, France



(a) *geometric input* **(b)** *pre-computed deformation field* **(c)** *small texture exemplars* **(d)** *texture synthesis and rendering in real-time*

Figure 1: *Our technique synthesizes textures in real-time (d) from small exemplars (c) with local control on scaling, shearing, and orientation. It is able to both compensate distortions and hide seams from the input parameterization (a) thanks to a deformation field (b).*

Abstract
The prevailing model in virtual 3D scenes is a 3D surface, which a texture is mapped onto, through a parameterization from the texture plane. We focus on accounting for the parameterization during the texture creation process, to control the deformations and remove the cuts induced by the mapping. We rely on the tiling and blending, a real-time and parallel algorithm that generates an arbitrary large texture from a small input example. Our first contribution is to enhance the tiling and blending with a deformation field, which controls smooth spatial variations in the texture plane. Our second contribution is to derive, from a parameterized triangle mesh, a deformation field to compensate for texture distortions and to control for the texture orientation. Our third contribution is a technique to enforce texture continuity across the cuts, thanks to a proper tile selection. This opens the door to interactive sessions with artistic control, and real-time rendering with improved visual quality.

CCS Concepts
• ***Computing methodologies*** → ***Texturing; Rendering;*** *Mesh geometry models;*

1. Introduction

A typical process for designing 3D assets involves initially modeling a geometric surface, then unfolding it onto the parametric plane (UV plane) before creating texture maps. This parameterization is then used to map the textures onto the surface during 3D rendering. Except in the very special case of developable surfaces, the parameterization distorts the distances, the angles and/or the areas, which generates artifacts when mapping the textures. The distortions can be reduced by cutting the surface into pieces and unfolding them separately. However, this operation raises another issue: discontinuities (so-called seams) appear along the cuts if the texture is not meticulously designed on both sides of the cut. As the parameterization is known before the textures are created, it is possible to account for it to create proper textures, which compensate for distortions and cuts. The present work follows this idea, in the context of real-time texture generation, which is very useful in interactive settings, such as scene design or video games.

Texture generation aims at creating textures with algorithms. It alleviates the manual work of artists, and allow for generating very large and detailed textures used in virtual scenes. *Offline* algorithms perform the calculations once, and store the result for later rendering of the scene. In this setting, calculations may be heavy, while the result is limited in memory. By contrast, *real-time* algorithms compute the texture on-demand during the rendering, allowing for generating unbounded textures with a fixed memory cost. In counterpart, it comes with strict computation constraints, short time budget, and parallelization requirements. The management of parameterization's distortions and cuts is difficult in this context, as no global optimization is tractable. In this paper, we rely on the tiling and blending (T&B) technique, which takes as input a small texture exemplar, generates and renders an unbounded texture result in real-time. Our contributions are as follows:

- We introduce a deformation field in the tiling and blending algorithm (Section 3), which allows for a local control of scale, shear, and rotation. This field associates a tensor (2×2 matrix) to any position in the UV plane.
- We derive a deformation field from the parameterization of a triangle mesh (Section 4), which compensates for distortions due to texture mapping. In a nutshell, we compute the deformation between the tangent plane in ambiant space and the UV plane, and we invert it.
- We modify the T&B algorithm along texture cuts to remove discontinuity artifacts (Section 5).

Our contribution is not only useful for compensating parameterization artifacts, it also comes with additional benefits (Section 6). Firstly, it allows for an artistic control of the texture, as the deformation field can be drawn by hand, or designed according to specific artistic goals. Second, it is useful for controlling the materials in physically-based rendering. Indeed, while multi-layer textures encode rich materials, their mapping on the surface may distort the appearance, e.g. the shape of light reflections [AKDW22]. We show that our technique may help to control the micro-geometry by making the result independent of the parameterization.

2. Related work

This work is related to real-time texture synthesis, and to global parameterization of meshes.

2.1. Real-time texture synthesis.

Tiling techniques assemble tiles with regular shapes (often quads) that are precomputed to be compatible, i.e. there is no seam on the border between two tiles. Some methods can be applied to surfaces of arbitrary topology [NC99, FL05]. Direct stochastic tilings [Wei04,LD05,LD06] generate aperiodic tilings in real-time. Their main weakness is to suffer from repetitions, as they rely on a limited set of precomputed tiles.

Patch based techniques cut pieces in an exemplar texture, and assemble them to build the result. Their irregular shape is optimized to limit seams. A few methods reach real-time, by optimizing the computation of cuts in parallel [LL12], or by precomputing the cuts [VSLD13]. These methods reduce the repetitions compared to tiling, at the cost of more seam artifacts.

Bombing consist in printing small predefined kernels at random positions, using stochastic point-wise processes [SA79, PFH00, DMLG02, Gla04, LHN05, MWT11, WSCP13]. They can represent discrete distributions of repetitive and structured patterns, such as printed fabrics, pebbles, or piles of tree leaves. These methods are often compliant to parameterization-free synthesis: the kernels are printed directly on the surface.

Procedural noises form a broad family of techniques, which define the texture as the result of a procedure. Most algorithms can generate textures in parallel at arbitrary high resolution and on an arbitrary large extent, while limiting the memory footprint to the procedure's parameters. Reviewing all techniques is beyond the scope of this paper [LLC*10]. Solid textures are worth mentioning, as they need no parameterisation of the surface. They are however dedicated to volumetric materials, such as wood or stone, which are different from surface patterns we are addressing here.

Gaussian noises are also relevant to our work. They rely on Gaussian random fields, controlled in the spectral domain. The principles of sparse convolution [Lew84] and spot noise [vW91] algorithms have given rise to numerous variants [LLDD09, GLLD12, GSV*14, GLM17]. Some of them avoid geometric distortions by a synthesis directly on the surface, or by a compensation in the parametric domain [LLC*10]. The main limitation is the narrow range of unstructured stochastic patterns that can be modeled.

Tiling and blending (T&B) is an algorithm which randomly draws tiles in an input exemplar, paves the plane with these tiles, and blend them where they overlap. T&B was first designed for Gaussian textures, and then extended to preserve local contrasts [HN18, DH19, BS19], global arrangements of patterns [LSD21, LSD23], and fine structures [FS24]. De Goes et al. [DGBD20] hinted the idea of aligning the texture on a mesh with a given orientation field. Spatial variations where recently introduced by Lutz et al. [LSG24] to model the surface of ocean. They rotate the tiles to control the orientations of waves. We drew inspiration from this approach, and extend it in two ways. First, we introduce non-rigid transformations (scaling and shearing). Second, our transformation field is finely sampled per-texel, while a coarse per-tile sampling is sufficient in their context.

2.2. Parameterization

Texturing arbitrary surfaces is a long standing problem [Tur91, WL01]. As said above, some procedural noises and bombing techniques do not need parameterization at all. Over the years, several other parameterization-free methods relied on optimizations or on machine learning [Tur01, ZZV*03, HZW*06, PXM*24]. Planar orthogonal projections can be used to map a texture on a geometric object, without need for a global parameterization. This strategy is pretty popular for its ease of implementation, but it is prompt to distortions and ghosting artifacts [Ngu07, Chapter 1]. Another strategy consists in optimizing only local parameterizations for a given texture, as Schuster et al. [STSK20] for example. In most cases, however, the use of a global parameterization remains the norm, because it is more versatile in the production and rendering pipeline.

With regard to the texture mapping, a global parameterization faces two challenges [FH05, SPR*07]: avoiding distortions, which cause texture deformations, and limiting cuts, which cause seams. To address distortions, a common strategy consists in minimizing a global energy, which penalizes area, length, and/or angle distortions. However, a low global energy does not prevent local distortions, which may cause strong texture deformations. The constraints can be relaxed by cutting the mesh before unfolding, which produce seams artifacts when mapping the texture. To mitigate seams, methods based on integer translations on quadrangular meshes have been proposed [CBK15], but they are limited to textures with regularly organised patterns. Parameterization remains a topic of interest. Recent techniques achieve low distortion [RPPSH17, CC23], at the cost of more cuts that may introduce texture discontinuity and orientation inconsistency across the cuts.

3. Deformed Tiling and Blending

3.1. Classical Tiling and Blending

Given a texture example E as input, the Tiling and Blending (T&B) algorithm generates an arbitrary large texture T that resembles E. This algorithm paves the UV plane k times: for example by dual tiling ($k = 2$) or hexagonal tiling ($k = 3$) [HN18, LSD23]. The generated value $T(\mathbf{u})$ at a texture coordinate $\mathbf{u}$ in the UV plane is a weighted average of k tiles E_i, one tile in each paving:

$$T(\mathbf{u}) = \sum_{i=1}^{k} w_i(\mathbf{u}) E_i(\mathbf{u}) \tag{1}$$

with:

$$E_i(\mathbf{u}) = E(\mathbf{u} + \mathbf{h}_i(\mathbf{u})) \tag{2}$$

where the non-negative weights $w_i(\mathbf{u})$ are given by compactly supported scalar functions. Each tile defined by Equation (2) is a piece drawn in the example texture E at a pseudo-random shift $\mathbf{h}_i(\mathbf{u})$ responsible for the randomness in the result.

3.2. Introducing deformations in Tiling and Blending

We propose to extend the standard T&B algorithm by introducing a deformation field $\mathcal{F}(\mathbf{u})$ that provides control over the content of the tiles. We modify Equation (2) as:

$$E_i(\mathbf{u}) = E(\mathcal{F}(\mathbf{u}) * (\mathbf{u} - \mathbf{c}_i(\mathbf{u})) + \mathbf{h}_i(\mathbf{u})) \tag{3}$$

where $*$ denotes the standard matrix-vector product. $\mathcal{F}(\mathbf{u})$ is a 2×2 tensor field which encodes scaling, shearing and rotation of the tiles (Figure 2). The center of deformation $\mathbf{c}_i(\mathbf{u})$ of the tile i is set to be its barycenter, which brings two advantages. First, it makes it intuitive to apprehend the effects of the deformation on the content of each tile. Second, it reduces the distance $\|\mathbf{u} - \mathbf{c}_i(\mathbf{u})\|$, which acts as an amplification factor (the lower the better) of the variations of $\mathcal{F}$.

An important feature of the proposed model is that the deformation field $\mathcal{F}$ in Equation (3) depends on $\mathbf{u}$, allowing it to vary within a tile. When deformations are defined constant per tile, as in Lutz et al. [LSG24], the tiling grid may become visible (Figure 3c). Varying the deformations within the tile makes it possible to remove these artifacts (Figure 3d).

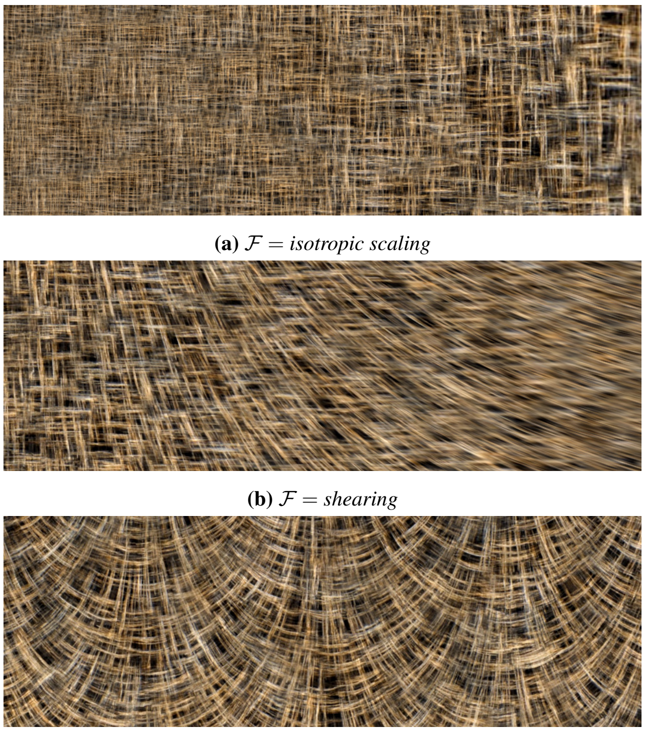

(a) $\mathcal{F}$ = *isotropic scaling*
(b) $\mathcal{F}$ = *shearing*
(c) $\mathcal{F}$ = *rotation*

Figure 2: *Results of the T&B algorithm, given by equations (1) and (3). Effects of various smooth deformation fields $\mathcal{F}$.*

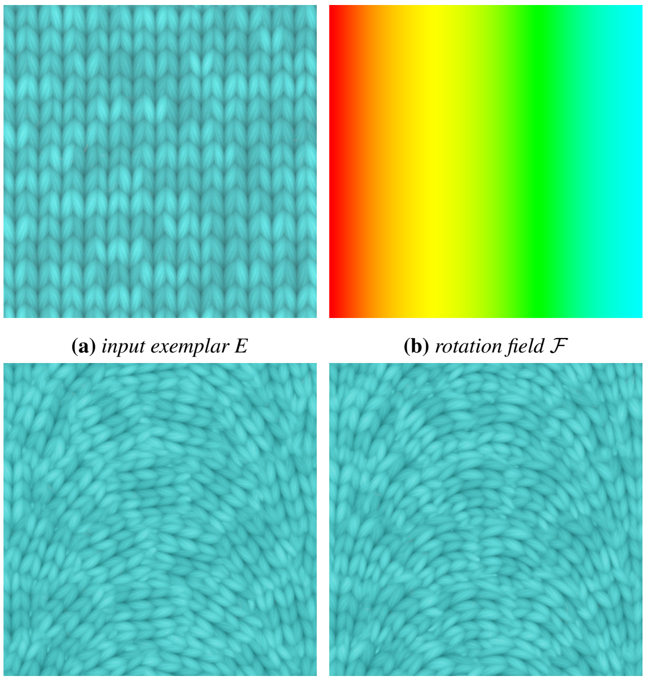
In the upcoming sections, we leverage our model in order to intricate the texture synthesis with the deformations induced by the parameterization of a 3D surface. In Section 4 we construct a deformation field $\mathcal{F}$ that compensates for the distortions induced by the parameterization while allowing control of the texture's orientation. In Section 5 we modify the hash function $\mathbf{h}_i$ and the deformation centers $\mathbf{c}_i$ to remove the discontinuities at the boundaries of texture pieces.

4. Compensating for the distortions due to the parameterization

We now illustrate how to construct a deformation field $\mathcal{F}(\mathbf{u})$ that compensate for the deformations induced by a given, piece-wise linear, parameterization of an input 3D triangular mesh. The first step is to compute the linear deformation undergone by each triangle caused by the parameterization. The shearing and scaling components of this deformation are then used to compensate for the distortions induced by the parameterization.

In the following, we distinguish points (or vectors) $\mathbf{x}$ in the 3D geometrical space, $\mathbf{u}$ in the 2D UV plane, and $\tilde{\mathbf{u}}$ the extension of $\mathbf{u}$ in 3D:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3, \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \in \mathbb{R}^2, \text{and } \tilde{\mathbf{u}} = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \in \mathbb{R}^3.$$



(a) *input exemplar E* **(b)** *rotation field $\mathcal{F}$*

(c) *result T : deformation per tile* **(d)** *result T : deformation per texel*

Figure 3: *To obtain smooth results (d) with no grid artifacts (c), it is required to finely sample the deformation field $\mathcal{F}$. In our model, the deformation is defined per texel, i.e. it varies within each tile.*

4.1. Computing the distortions

Each triangle of the input mesh undergoes a transformation while transitioning from the 2D UV plane to the 3D geometrical space. This transformation F is given by the *deformation gradient*, a standard concept in finite element methods.

Consider a triangle of the input mesh along with its vertices coordinates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$ in the 3D geometric space, its normal $\mathbf{x}_n$, and its vertices coordinates $\tilde{\mathbf{u}}_0$, $\tilde{\mathbf{u}}_1$, $\tilde{\mathbf{u}}_2$ in the extended UV space. Denote the geometric edges of the triangle as $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and their corresponding extended UV edges as $\tilde{\mathbf{u}}_{ij} = \tilde{\mathbf{u}}_j - \tilde{\mathbf{u}}_i$. The deformation gradient F is then defined as

$$F = \begin{bmatrix} | & | & | \\ \mathbf{x}_{01} & \mathbf{x}_{02} & \mathbf{x}_n \\ | & | & | \end{bmatrix} \begin{bmatrix} | & | & 0 \\ \tilde{\mathbf{u}}_{01} & \tilde{\mathbf{u}}_{02} & 0 \\ | & | & 1 \end{bmatrix}^{-1}. \tag{4}$$

It is a linear mapping $\mathbb{R}^3 \mapsto \mathbb{R}^3$ that maps each extended UV vector $\tilde{\mathbf{u}}_{ij}$ onto its equivalent geometric vector $\mathbf{x}_{ij}$, i.e. $\mathbf{x}_{ij} = F * \tilde{\mathbf{u}}_{ij}$.

The rigid (rotation, R) and and non-rigid (combining scaling and shearing, S) parts of F are then separated using singular value decomposition $F = U\Sigma V^T$:

$$F = RS = \left(UV^T\right)\left(V\Sigma V^T\right) \tag{5}$$

Computing F for every triangle of the input mesh leads to two 3×3 tensor fields, $R(\mathbf{u})$ and $S(\mathbf{u})$, which are constant per triangle. These tensor fields are the key to compensate for the distortions induced by the parameterization.

4.2. Compensating for the distortions

To compensate for the texture distortions induced by the mesh parameterization, a 2×2 tensor field $\mathcal{S}(\mathbf{u})$ is defined based on $S(\mathbf{u})$ from Equation (5). Texture distortion only happens between the texture plane and the tangent plane of each triangle. Therefore, the normal component in $S(\mathbf{u})$ is ignored: $\mathcal{S}(\mathbf{u})$ is defined as the top left 2×2 block of $S(\mathbf{u})$.

Setting $\mathcal{F}(\mathbf{u}) = \mathcal{S}(\mathbf{u})$ in Equation (3) effectively compensate for the scaling and shearing distortions as shown in Figure 4b. In such a case, discontinuities appear at the edges of each triangle since $\mathcal{F}(\mathbf{u})$ is defined per-triangle, and therefore discontinuous. Moreover, $\mathcal{F}$ does not contain any rotational component, and the texture is only oriented on the surface by the isoparametric lines.

4.3. Controlling the orientation

We propose a technique to control the texture orientation by aligning it with a tangent orientation field. Let $\mathbf{o}(\mathbf{u}) \in \mathbb{R}^3$ be a tangent vector field, i.e. $\mathbf{o}(\mathbf{u})$ lies in the plane of the geometric triangles. Orienting the texture with $\mathbf{o}(\mathbf{u})$ amounts to rotating the texture by the deviation angle between $\mathbf{o}(\mathbf{u})$ and the UV iso-parametric lines. We define this angle $\theta(\mathbf{u})$ as

$$\theta(\mathbf{u}) = \mathit{angle}\left(R^{-1}(\mathbf{u}) * \mathbf{o}(\mathbf{u}), (1,0,0)^T\right). \tag{6}$$

Let $\mathcal{R}(\theta(\mathbf{u}))$ denote the 2D rotation of angle $\theta(\mathbf{u})$. Setting $\mathcal{F}(\mathbf{u}) = \mathcal{R}(\theta(\mathbf{u}))\mathcal{S}(\mathbf{u})$ aligns the texture with $\mathbf{o}(\mathbf{u})$, as can be seen in Figure 4c compared to 4b: the tangent vector field $\mathbf{o}(\mathbf{u})$ is aligned with the parallels instead of the meridians, and so is the synthesised texture. Remember that the parameterization does not change, only the texture synthesis is modified. We can see, though, that visual discontinuities remain on the edges, because $\mathcal{F}$ is only defined per-triangle without any guarantee about its continuity.

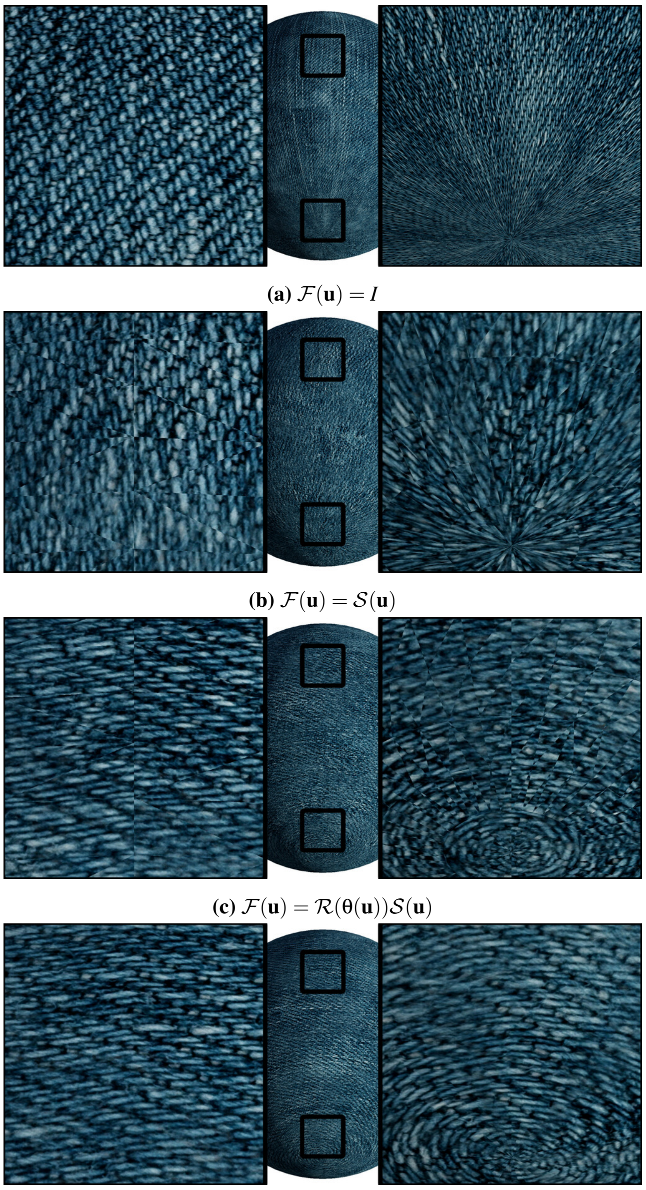
4.4. Smoothing of the deformation field

The visual artifacts (Figures 4b and 4c) due to the discontinuity of $\mathcal{F}$ can be removed (Figure 4d) by introducing an additional smoothing:

$$\mathcal{F}(\mathbf{u}) = \mathcal{R}\left(\mathcal{L}\left(\theta(\mathbf{u})\right)\right) * \mathcal{L}\left(\mathcal{S}(\mathbf{u})\right), \tag{7}$$

where $\mathcal{L}$ is a smoothing operator. Note that because the group of rotation matrices is not stable by linear combinations, $\mathcal{L}$ is applied to the angles $\theta(\mathbf{u})$, not to the matrices $\mathcal{R}$.

Several smoothing operators are good candidates for $\mathcal{L}$, including Gaussian filters, barycentric coordinates, and finite elements. In our model, we used finite elements defined over regular grid: the fields θ and $\mathcal{S}$ are sampled at the vertices of the grid, and linearly interpolated in-between. The finite element operator is straightforward to implement, interpolates the original fields, and avoids visual artifacts (Figure 4d). The resolution of the grid controls the balance between smoothness and faithfulness to the data, and it



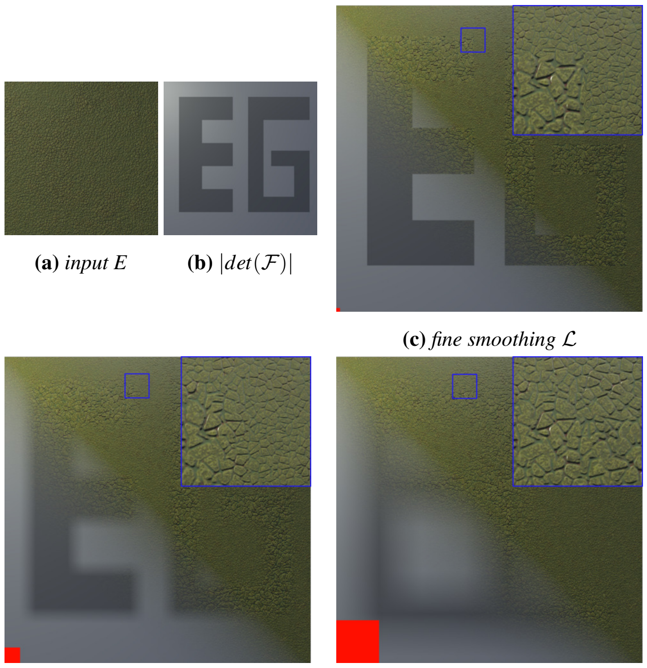
(a) $\mathcal{F}(\mathbf{u}) = I$

(b) $\mathcal{F}(\mathbf{u}) = \mathcal{S}(\mathbf{u})$

(c) $\mathcal{F}(\mathbf{u}) = \mathcal{R}(\theta(\mathbf{u}))\mathcal{S}(\mathbf{u})$

(d) $\mathcal{F}(\mathbf{u}) = \mathcal{R}(\mathcal{L}\theta(\mathbf{u})) * \mathcal{L}(\mathcal{S}(\mathbf{u}))$

Figure 4: *Effects of the choice of deformation field $\mathcal{F}(\mathbf{u})$ on the synthesised texture. (a) Without deformation field, the result is identical to the classical T&B algorithm. (b) The tensor field $\mathcal{S}(\mathbf{u})$ compensates for the distortions induced by the mesh parameterization: the important distortions around the poles are reduced. (c) Integrating $\theta(\mathbf{u})$ provides an additional control over the texture's orientation: the texture is now oriented along the parallels instead of the meridians. (d) Smoothing the deformation fields by integrating $\mathcal{L}$ avoids the appearance of discontinuities around the faces of the mesh.*



(a) *input E* **(b)** $|det(\mathcal{F})|$ **(c)** *fine smoothing* $\mathcal{L}$ **(d)** *medium smoothing* $\mathcal{L}$ **(e)** *large smoothing* $\mathcal{L}$

Figure 5: *Setting of the smoothing resolution (the red squares are cells of the finite elements grid). The deformation field $\mathcal{F}$ is an isotropic scaling, the color in (b) represents the area distortion $|det(\mathcal{F})|$. (c) The resolution is too fine, discontinuities appear. (d) The resolution is a good compromise. (e) The resolution is too large, features in $\mathcal{F}$ disappear.*

must be set according to the frequency of variation of $\mathcal{F}$. Using too fine resolution may interpolate large deformation changes in a small area, making abrupt changes visible (Figure 5c). On the opposite, with too large resolution the algorithm will fail to capture the variations of the deformation field inside the elements of the grid (Figure 5e). As of now, it is left as a user parameter (Figure 5d is a good compromise).

5. Texture continuity across cuts

A cut is a sequence of edges which are embedded twice in the UV plane: a *left* edge, and a *right* edge. Ensuring the visual continuity across the cuts is necessary to avoid visible seams when the texture is mapped onto the surface.

Problem statement. To avoid the appearance of visible seams in T&B approaches, two coherence criteria must be met across the cuts: coherence of the visual aspect of the texture, and coherence of the content of the synthesised texture. The first coherence is ensured, even across cuts, thanks to the deformation field $\mathcal{F}$ that leads to no visual changes of distortion once the texture is mapped onto the surface mesh. The coherence of content however is not guaranteed across cuts: a pair of corresponding left and right triangles across a cut are not neighbours in the UV plane; therefore the tiles blended during synthesis are different in the left and right parts of

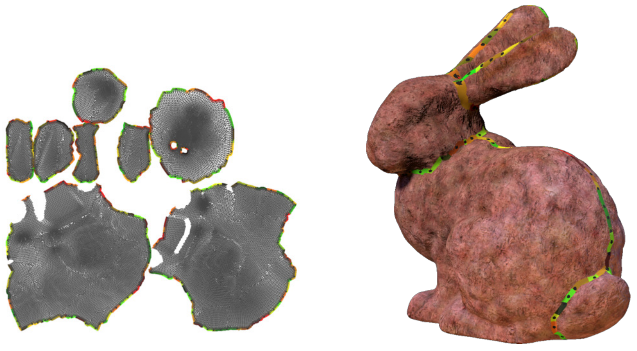


Figure 6: *Visualisation of the boundary cells in the UV plane (left), and on the surface embedded in the 3D space (right). The boundary cells are identified with a unique colour and the black dots are located at the center of deformations points* $\mathbf{b}_l$ *and* $\mathbf{b}'_l$.

the cut. Our solution to enforce this coherence is to select the exact same tiles on the left and on the right.

Solution. The content of a tile is fully defined by two parameters in Equation (3): the pseudo-random shift $\mathbf{h}_i$, and the center of deformation $\mathbf{c}_i$. Therefore, it is necessary to make these parameters coherent between the left and right sides of the cuts to ensure that the same tiles are blended on each side.

To do so, a seam band is first defined around the edge cuts to delineate the area where the coherence must be enforced. This band is further divided into cells indexed by l, in which a unique central point $\mathbf{b}$ is defined (Figure 6). Each cell and each central point is embedded twice in the UV plane, leading to left-right pairs of boundary cells, and two points $\mathbf{b}_l$ and $\mathbf{b}'_l$ respectively on the left and on the right. The coherence is finally ensured by setting the same pseudo-random shift of all tiles within the cell to the same value $\mathbf{h}_l$, and setting the center of deformation to either $\mathbf{b}_l$ or $\mathbf{b}'_l$ whether if the tile falls in the left or the right part of the cut. That is, **if and only** if a tile's center lies within a cell l (see Figure 7 bottom), we replace Equation (3) with:

$$E_i(\mathbf{u}) = E\left(\mathcal{F}(\mathbf{u}) * (\mathbf{u} - \mathbf{b}_l(\mathbf{u})) + \mathbf{h}_l(\mathbf{u})\right) \qquad (8)$$

on the left, and similarly with $\mathbf{b}'_l(\mathbf{u})$ on the right side. Note that the shift $\mathbf{h}_l$ is the same on both sides, as the cells are given the same index l.

Computing the boundary cells. The pairs of boundary cells must satisfy the three following properties: the edges on both sides match in 3D; the points $\mathbf{b}_l$ and $\mathbf{b}'_l$ lie on a boundary edge, and match in 3D; and the size of the cells must exceed the size of the tiles. We have implemented a simple and effective algorithm to compute boundary cells satisfying these criteria (Figure 7):

1. A pair of quad cells is built for any pair of boundary edges. Two quad edges are parallel to the boundary edge, the two others being the bisectors with the adjacent boundary edges. The points $\mathbf{b}_l$ and $\mathbf{b}'_l$ are set to the midpoints of boundary edges.
2. The cells are merged in a breadth-first manner, until they exceed the size of the T&B tiles in UV space.

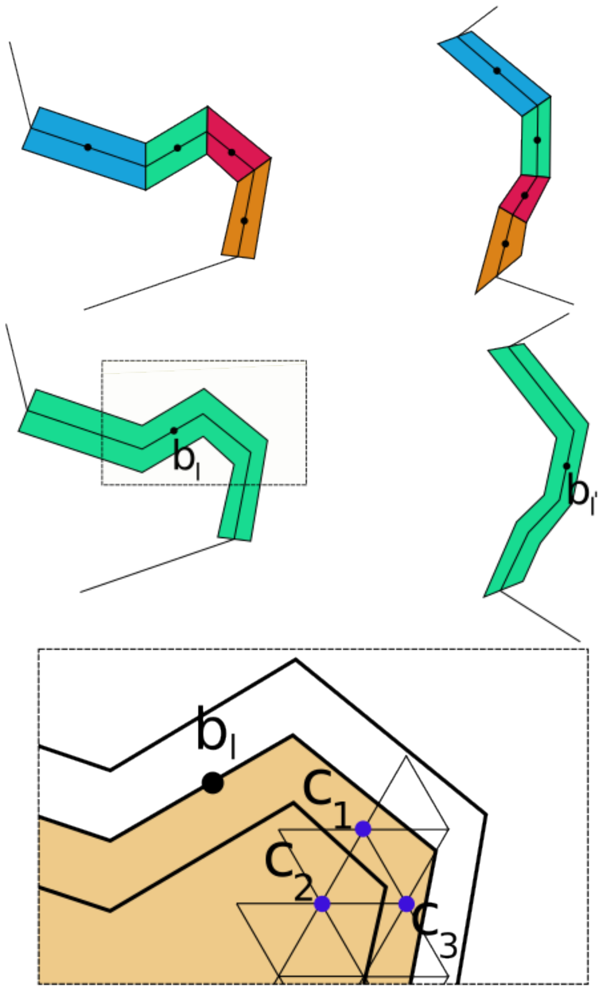


Figure 7: *Computation of the boundary cell. Top: all pairs of boundary edges are given a pair of quad cells. Middle: cells are merged to reach a critical size. Bottom: the centers of tiles (*$\mathbf{c}_1$ *and* $\mathbf{c}_3$*) are replaced by the center of cells (*$\mathbf{b}_l$*) as in equation 8.*

3. For each pair of cells, the most central $\mathbf{b}_l$ and $\mathbf{b}'_l$ are saved, other points are discarded.

A result of this algorithm is shown in Figure 6.

6. Discussion

6.1. Technical details

We used a finite element operator based on a regular grid to smooth the deformation fields. This operator samples the deformation fields at the grid vertices before interpolating their value. Since the deformation fields are only defined inside each UV triangles, samples outside of these triangles are undefined: in such cases, we set the sample value to that of the closest UV triangle instead.

Throughout this article, the illustrated results were rendered using hexagonal tiling, and tiles were blended with the Mix-Max

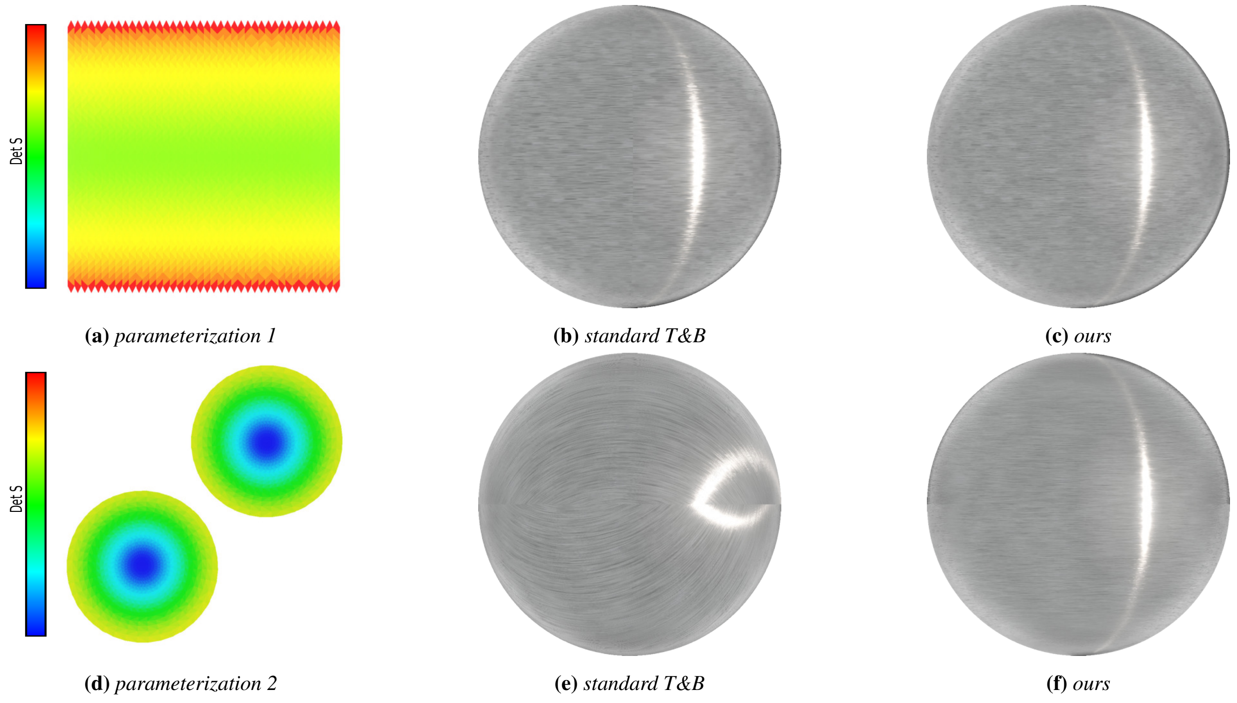


Figure 8: *Synthesis of an anisotropic micro-surface (brushed metal). Two parameterizations (a) and (d) are colored by $|det(\mathcal{F})|$. Compared to the standard T&B (b and e), our algorithm (c and f) controls the orientations and hides texture cuts. The typical elongated highlight of brushed metal is robustly reproduced, and continuous across the cut.*

operator [FS24]. However, our work is fully compatible with any tiling, and any blending weights.

Neither offline rendering nor pre-computed textures were used for generating our images: the textures were synthesized at each frame in a rendering engine. Orthographic projection is used (except Figure 9) to avoid additional perspective distortions. Our additional video shows live interactive rendering. Our Figure 2 has been implemented in lightweight *shader toy* demos available at rotation URL, scaling URL, and shearing URL. We also provide the 3D rendering code as additional material.

In addition to the texture examples for T&B, we store two textures: one containing $\mathcal{F}$, and one containing the cell indices l and the centers $\mathbf{b}_l$. The standard T&B algorithm with k tilings requires k texture fetches per layer for each texel (Equation (1)). Our contribution requires $k+1$ additional fetches: k fetches are necessary to get the index and central point of the border (l and $\mathbf{b}_l$ in Equation (8)), and an other one to get the deformation $\mathcal{F}$. The additional cost of the multiplication by $\mathcal{F}$ is negligible compared to the texture fetches.

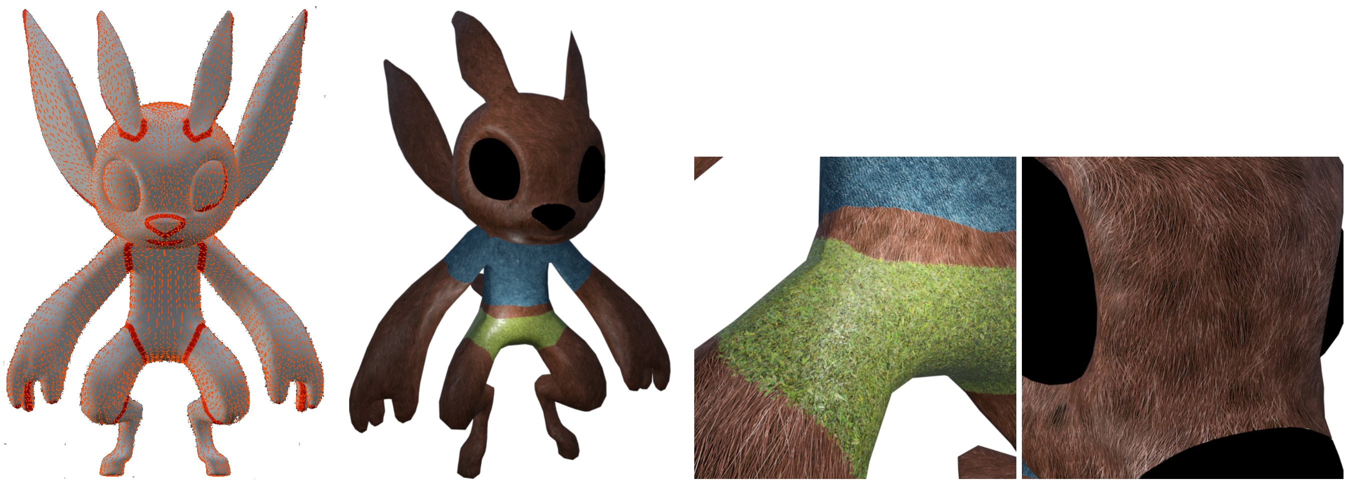
Anti-aliasing is crucial in real-time rendering. The textures synthesized by T&B can be filtered by MIP-mapping the exemplars [DH19, GSDT22, FS24]. The deformation field $\mathcal{F}$ deforms the tiles, and so it deforms the pixel footprint. We adapt the shader program as follows: given a footprint defined by its center u, and the duv/dx and duv/dy vectors, we simply multiply each of these vectors by $\mathcal{F}(u)$.

Compared to the techniques that bake the texture, a major advantage of our procedural approach is that it can produce ultra-high resolution details, as the result is computed in constant-time in the fragment shader, according to the rendering needs. As an example, storing the full-resolution texture rendered in Figure 9 would require about 10^{10} texels. This is several orders of magnitude above what non-procedural method can produce, and above current 4K textures. The actual memory footprint for this result is 37MB (for the 3 multi-layer texture examples, the deformation field, and the map of cuts, all at 1024×1024 resolution).

6.2. Physically based rendering

The introduction of deformation fields in the T&B is fully compatible with physically based rendering of complex materials. This includes the rendering of materials defined by several maps as illustrated in Figures 1, 8 and 9: albedo, normal map, metallic map, height map, and ambient occlusion map.

In the context of microfacets theory, linear deformations change the BRDF, the resulting surface appearance, and the shape of specular lobes [AKDW22]. We show in Figure 8 the synthesis of an anisotropic micro-surface (brushed metal). We compare two different parameterizations of the same mesh. When using the standard T&B algorithm (8b and 8e), the result strongly depends on the parameterization, and the cuts are visible. Conversely with our technique (8c and 8f), the results are very similar and the cuts are not visible. The typical elongated highlight of brushed metal is perfectly continuous across the cut in 8f. This is a challenging situa-



(a) *geometry and orientation field; cuts are shown in red.* **(b)** *textured result* **(c)** *zoom* **(d)** *zoom*

Figure 9: *Results of texture synthesis using our deformed T&B algorithm on a character. The high-resolution texture aligns with the orientation field, without any distortion. The potential resolution is equivalent to* 10^{10} *texels (if it were baked).*

tion, as the specular highlights tend to reinforce continuity defects. In addition, the rendering is stable, filterable, and coherent across time (see the additional video). This is a clear advantage for realistic rendering in real-time.

6.3. Artistic Control

The proposed deformed T&B algorithm offers major advantages for 3D artists and modelers when it comes to texturing a mesh from a single piece of example. It retains all benefits from classical T&B algorithm while providing an accurate artistic control over the generated texture (using orientation and deformation fields) that was missing in previous algorithms.

The end-user has total control over the orientation of the texture when designing a tangent field $\mathbf{o}(u)$ (Figure 9a). The design of such orientation field can be assisted with simple tools [DGBD20]. One intuitive approach is to use the gradient of the geodesic distance from certain vertices on the mesh. In Figure 9, the orientation field was designed to naturally orient the hair of the character (Figure 9c), and to align the texture along the arms direction (Figure 9d). Together with orientation control, the end-user can also also apply additional local texture deformation such as scaling, shearing, rotation, or any defined transformation (Figure 11).

6.4. Limitations and future work

Our algorithm performs well when the deformation field has smooth variations. The quality of the synthesized texture declines when confronted to strong variations in the deformation field within a tile. Such a situation is shown figure Figure 10. Note that if such case occurs across a cut then the result is not guaranteed to be seamless. A first solution could be to use smaller tiles. However, it is known from previous work that the tiles must remain large enough to preserve the structure of patterns. A second solution could be to smooth the field strongly, the drawbacks of which we have shown in Figure 5. Hence, the two solutions are limited, and a compromise must be found in difficult cases.

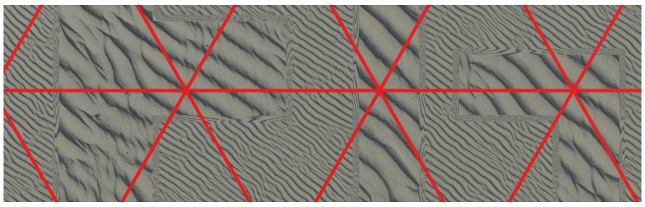


Figure 10: *Strong variations of the deformation field within a tile (in red) may distort the texture patterns.*

The smoothing technique we have chosen operates independently on the left and right sides of a cut. Therefore, strong changes in orientations across the cut may create visual artifacts, even though the texture is continuous. In our experience, shiny PBR materials are more sensitive than matt albedo textures, because specular lobes are distorted. For now, we expect the orientation field to be continuous across the cut. It would be worthwhile to adapt the smoothing operator to handle this.

The smoothing operator chosen in Section 4.4 has a fixed resolution, manually set to balance smoothness and faithfulness to the data (Figure 5). This can be a limiting factor if the deformation field has different scales of variations across the texture. A promising solution to this problem is to use an adaptive smoothing. However, this may require to adapt the resolution of the synthesis tiling as well, and no adaptive T&B is available as of today.

There remains a theoretical problem, which is the potential appearance of cracks within a tile if the deformation field varies so quick that $\mathcal{F}(\mathbf{u}) * (\mathbf{u} - \mathbf{c}_i(\mathbf{u}))$ folds over itself (i.e. is not injective).

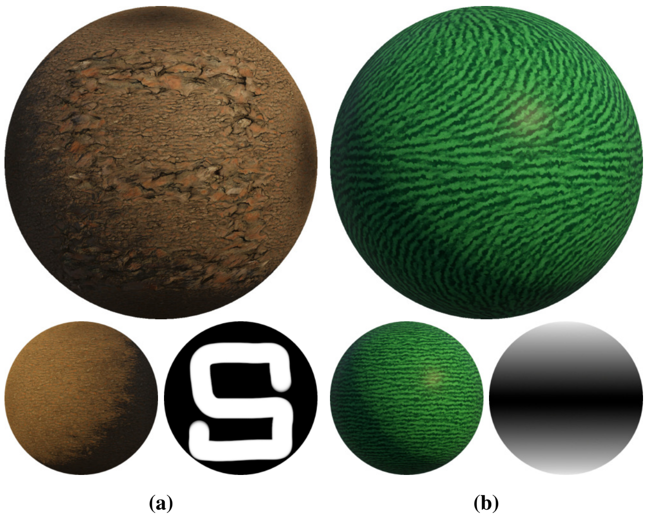


Figure 11: *Examples of additional artistic control over scaling (a) and orientation (b). Bottom left: homogeneous synthesis (distortions are compensated). Bottom right: additional artistic deformations drawn on the mesh. Top: result of the synthesis including the additional deformation.*

Up to now we haven't observed this phenomenon in practice, however we plan to study the conditions required for injectivity in the future.

Local structures in the exemplar which correspond to clearly delineated texture elements (such as leaves or stones) are well reproduced by our method, provided that the blending weights are appropriate [FS24]. But global structures that correspond to the organisation of elements (such as alignments of bricks), are not currently addressed by our method. Lutz et al. [LSD21,LSD23] demonstrate that T&B can generate this type of textures, but the combination with our approach is not straightforward.

A perspective is to explore the behavior of our method in dynamic contexts. In a first scenario, the geometry is deformed dynamically. In this case, the texture is currently stretched and sheared, exactly as with pre-computed texture maps. It would be interesting to explore complex elastic behaviors [GLS*20]. In a second scenario, a dynamic orientation field would affect the texture, such as wind blowing on a surface of water or on a fur.

An other promising avenue for future research is to use the deformation field for mimicking perspective, and generate texture in the image plane [DBP*15].

7. Conclusions

The presented work is about real-time by-example texture synthesis and rendering. Our three contributions (introduction of deformation fields in the tiling and blending algorithm, automatic compensation of distortions due to the parameterization, handling of texture continuity across cuts) solve several issues commonly encountered during texture mapping and/or synthesis as illustrated by the results. We show that it is possible to compensate the distortions induced by the parameterization of a given mesh during texture synthesis. The deformed T&B algorithm is compatible with all kinds of texture maps and materials, provided that they can be linearly interpolated and filtered, making it suitable for physically based rendering. It benefits from the advantages of the procedural approach: the result is computed and anti-aliased on-demand, in parallel in the fragment shader, at a fixed time cost per fragment. The result may have a ultra-high resolution, while the memory cost remains bounded (the texture examples, plus two maps).

The present work may be useful for 3D artists and modelers as it strongly reduces the care that needs to be taken to cutting and unfolding a mesh. While texturing both sides of cuts requires a careful management in many existing algorithms, it is naturally handled in this work. The result is a synthesised texture that exhibits a clear homogeneity when wrapped onto a mesh (pattern size and orientation), while still retaining the artist's ability to control it.

8. Acknowledgments

This work was supported by the project POSTURE (ANR-22-CE38-0010-01). We thank Romain Fournier for the Mix-Max implementation. All textures used in this paper are from the following sites: ambientCG, TextureCan, freePBR and Poly Haven. The meshes are from Sketchfab (Figure 1) and Free3D (Figure 9).

References

[AKDW22] Atanasov A., Koylazov V., Dimov R., Wilkie A.: Microsurface transformations. *Computer Graphics Forum 41*, 4 (2022), 105–116. doi:10.1111/cgf.14590. 2, 7

[BS19] Burley B., Studios W. D. A.: On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques 8*, 4 (2019), 31–53. 2

[CBK15] Campen M., Bommes D., Kobbelt L.: Quantized global parametrization. *Acm Transactions On Graphics (tog) 34*, 6 (2015), 1–12. doi:10.1145/2816795.2818140. 3

[CC23] Coiffier G., Corman E.: The Method of Moving Frames for Surface Global Parametrization. *ACM Transactions on Graphics 42*, 5 (Oct. 2023), 1–18. doi:10.1145/3604282. 3

[DBP*15] Diamanti O., Barnes C., Paris S., Shechtman E., Sorkine-Hornung O.: Synthesis of complex image appearance from limited exemplars. *ACM Transactions on Graphics 34*, 2 (Mar. 2015), 22:1–22:14. doi:10.1145/2699641. 9

[DGBD20] De Goes F., Butts A., Desbrun M.: Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics (TOG) 39*, 4 (2020), 110–1. doi:10.1145/3386569.3392389. 2, 8

[DH19] Deliot T., Heitz E.: *GPU Zen 2: Advanced Rendering Techniques*. Black Cat Publishing Inc., Encinitas, CA., 2019, ch. Procedural Stochastic Textures by Tiling and Blending. 2, 7

[DMLG02] Dischler J.-M., Maritaud K., Lévy B., Ghazanfarpour D.: Texture particles. *Computer Graphics Forum 21*, 3 (2002), 401–410. doi:10.1111/1467-8659.t01-1-00600. 2

[FH05] Floater M. S., Hormann K.: Surface parameterization: a tutorial and survey. *Advances in multiresolution for geometric modelling* (2005), 157–186. doi:10.1007/3-540-26808-1_9. 3

[FL05] Fu C.-W., Leung M.-K.: Texture tiling on arbitrary topological surfaces using wang tiles. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques* (2005), EGSR '05, Eurographics Association, pp. 99–104. doi:10.2312/EGWR/EGSR05/099-104. 2

[FS24] Fournier R., Sauvage B.: Mix-max: A content-aware operator for real-time texture transitions. *Computer Graphics Forum 43*, 6 (2024). doi:10.1111/cgf.15193. 2, 7, 9

[Gla04] Glanville S.: Texture bombing. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Fernando R., (Ed.). Pearson Higher Education, 2004, ch. 20, p. 323–338. 2

[GLLD12] Galerne B., Lagae A., Lefebvre S., Drettakis G.: Gabor noise by example. *ACM Transactions on Graphics 31*, 4 (July 2012), 73:1–73:9. doi:10.1145/2185520.2185569. 2

[GLM17] Galerne B., Leclaire A., Moisan L.: Texton noise. *Computer Graphics Forum 36*, 8 (2017), 205–2018. doi:10.1111/cgf.13073. 2

[GLS*20] Guingo G., Larue F., Sauvage B., Lutz N., Dischler J.-M., Cani M.-P.: Content-aware texture deformation with dynamic control. *Computers Graphics 91* (2020), 95–107. 9

[GSDT22] Grenier C., Sauvage B., Dischler J.-M., Thery S.: Color-mapped noise vector fields for generating procedural micro-patterns. In *Computer Graphics Forum* (2022), vol. 41, Wiley Online Library, pp. 477–487. 7

[GSV*14] Gilet G., Sauvage B., Vanhoey K., Dischler J.-M., Ghazanfarpour D.: Local random-phase noise for procedural texturing. *ACM Transactions on Graphics (ToG) 33*, 6 (2014), 1–11. doi:10.1145/2661229.2661249. 2

[HN18] Heitz E., Neyret F.: High-performance by-example noise using a histogram-preserving blending operator. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 1–25. doi:10.1145/3233304. 2, 3

[HZW*06] Han J., Zhou K., Wei L.-Y., Gong M., Bao H., Zhang X., Guo B.: Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer 22*, 9 (Sep 2006), 918–925. doi:10.1007/s00371-006-0078-3. 2

[LD05] Lagae A., Dutré P.: A procedural object distribution function. *ACM Transactions on Graphics 24*, 4 (Oct. 2005), 1442–1461. doi:10.1145/1095878.1095888. 2

[LD06] Lagae A., Dutré P.: An alternative for wang tiles: Colored edges versus colored corners. *ACM Transactions on Graphics 25*, 4 (Oct. 2006), 1442–1459. doi:10.1145/1183287.1183296. 2

[Lew84] Lewis J.-P.: Texture synthesis for digital painting. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 245–252. doi:10.1145/964965.808605. 2

[LHN05] Lefebvre S., Hornus S., Neyret F.: Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)* (April 2005), ACM SIGGRAPH, ACM Press. doi:10.1145/1053427.1053454. 2

[LL12] Lasram A., Lefebvre S.: Parallel patch-based texture synthesis. In *High Performance Graphics conference proceedings* (2012). doi:/10.2312/EGGH/HPG12/115-124. 2

[LLC*10] Lagae A., Lefebvre S., Cook R., DeRose T., Drettakis G., Ebert D. S., Lewis J. P., Perlin K., Zwicker M.: A survey of procedural noise functions. *Computer Graphics Forum 29*, 8 (2010), 2579–2600. doi:10.1111/j.1467-8659.2010.01827.x. 2

[LLDD09] Lagae A., Lefebvre S., Drettakis G., Dutré P.: Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics 28*, 3 (July 2009), 54:1–54:10. doi:10.1145/1531326.1531360. 2

[LSD21] Lutz N., Sauvage B., Dischler J.-M.: Cyclostationary Gaussian noise: theory and synthesis. *Computer Graphics Forum (Proceedings Eurographics) 40*, 2 (2021), 239–250. doi:10.1111/cgf.142629. 2, 9

[LSD23] Lutz N., Sauvage B., Dischler J.-M.: Preserving the autocovariance of texture tilings using importance sampling. *Computer Graphics Forum (Proceedings Eurographics) 42*, 2 (May 2023), 347–358. doi:10.1111/cgf.14766. 2, 3, 9

[LSG24] Lutz N., Schoentgen A., Gilet G.: Fast orientable aperiodic ocean synthesis using tiling and blending. In *High-Performance Graphics* (2024). doi:10.1145/3675388. 2, 3

[MWT11] Ma C., Wei L.-Y., Tong X.: Discrete element textures. *ACM Transactions on Graphics 30*, 4 (July 2011), 62:1–62:10. doi:10.1145/2010324.1964957. 2

[NC99] Neyret F., Cani M.-P.: Pattern-based texturing revisited. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co. doi:10.1145/311535.311561. 2

[Ngu07] Nguyen H.: *Gpu gems 3*, first ed. Addison-Wesley Professional, 2007. 2

[PFH00] Praun E., Finkelstein A., Hoppe H.: Lapped textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 465–470. doi:10.1145/344779.344987. 2

[PXM*24] Pajouheshgar E., Xu Y., Mordvintsev A., Niklasson E., Zhang T., Süsstrunk S.: Mesh neural cellular automata. *ACM Trans. Graph. 43*, 4 (July 2024). doi:10.1145/3658127. 2

[RPPSH17] Rabinovich M., Poranne R., Panozzo D., Sorkine-Hornung O.: Scalable locally injective mappings. *ACM Transactions on Graphics 36*, 2 (Apr. 2017), 16:1–16:16. doi:10.1145/2983621. 3

[SA79] Schachter B., Ahuja N.: Random pattern generation processes. *Computer Graphics and Image Processing 10*, 2 (June 1979), 95–114. doi:10.1016/0146-664X(79)90044-3. 2

[SPR*07] Sheffer A., Praun E., Rose K., et al.: Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision 2*, 2 (2007), 105–171. doi:10.1561/0600000011. 3

[STSK20] Schuster K., Trettner P., Schmitz P., Kobbelt L.: A three-level approach to texture mapping and synthesis on 3d surfaces. *Proc. ACM Comput. Graph. Interact. Tech. 3*, 2 (2020), 1–1. 2

[Tur91] Turk G.: Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph. 25*, 4 (July 1991), 289–298. doi:10.1145/127719.122749. 2

[Tur01] Turk G.: Texture synthesis on surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, Association for Computing Machinery, p. 347–354. doi:10.1145/383259.383297. 2

[VSLD13] Vanhoey K., Sauvage B., Larue F., Dischler J.-M.: On-the-fly multi-scale infinite texturing from example. *ACM Transactions on Graphics 32*, 6 (November 2013), 208:1–208:10. (Proceedings of Siggraph Asia'13). doi:10.1145/2508363.2508383. 2

[vW91] van Wijk J. J.: Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph. 25*, 4 (July 1991), 309–318. doi:10.1145/127719.122751. 2

[Wei04] Wei L.-Y.: Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2004), HWWS '04, ACM, pp. 55–63. doi:10.1145/1058129.1058138. 2

[WL01] Wei L.-Y., Levoy M.: Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01. doi:10.1145/383259.383298. 2

[WSCP13] Wang L., Shi Y., Chen Y., Popescu V.: Just-in-time texture synthesis. *Computer Graphics Forum 32*, 1 (2013), 128–136. doi:10.1111/cgf.12003. 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 22*, 3 (July 2003), 295–302. doi:10.1145/882262.882266. 2