Content-aware texture deformation with dynamic control

Geoffrey Guingo, Frédéric Larue, Basile Sauvage, Nicolas Lutz, Jean-Michel Dischler, Marie-Paule Cani

Computers & Graphics (2020)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Content-aware texture deformation with dynamic control

Geoffrey Guingo[a,b], Frédéric Larue[a], Basile Sauvage[a], Nicolas Lutz[a], Jean-Michel Dischler[a], Marie-Paule Cani[c]

[a] *ICube, Université de Strasbourg, CNRS, France*
[b] *Université Grenoble Alpes, CNRS (LJK), and Inria, Grenoble, France*
[c] *LIX, École Polytechnique, CNRS, France*

ARTICLE INFO

Article history:
Received July 9, 2020

Keywords: Texturing, Deformation, Real-time, Animation, GPU

ABSTRACT

Textures improve the appearance of virtual scenes by mapping visual details on the surface of 3D objects. Various scenarios – such as real-time animation, interactive texture modelling, or offline post-production – require textures to be deformed in a controllable and plausible manner. We propose a novel approach to model and control texture deformations, which is easy to implement in a standard graphics pipeline. The deformation is implemented at pixel resolution as a warping in the parametric domain. The warping is controlled locally and dynamically by real-time integration along the streamlines of a pre-computed flow field. We propose a technique to pre-compute the flow field from a simple scalar map representing heterogeneous dynamic behaviors. Moreover, to manage sampling issues arising in over-stretched areas during deformation, we provide a mechanism based on re-sampling and texture synthesis. Warping may alternatively be controlled by deformation of the underlying surface, environment parameters or interactive editing, which demonstrates the versatility of our approach.

1. Introduction

Textures are a common way to enhance the realism of virtual scenes. Their mechanism to increase user immersion is to provide an approximate solution to complex light-matter interaction related to the presence of micro- and meso-scale surface details that would be otherwise hard to explicitly simulate due to memory or computation cost. Multiple texture layers enable representing rich information about surface material and fine geometry, including color (albedo), normal, displacement, shininess, etc. Mapping textures onto the surface of 3D objects leads to great results in terms of surface appearance for static scenes.

However, texturing is much more challenging when animation or time-dependent effects are involved. Besides natural phenomena, such as ageing, weathering, or drying effects, animated objects often imply evolving visual patterns. Fluids show weakly structured patterns, that move, appear, disappear, or merge. By contrast, solids show complex patterns that are distorted, stretched or sheared. Modelling such dynamic behaviours involves the texture, the (possibly animated) geometry, and the mapping / parameterization. While these three aspects have been given a lot of attention as separate topics, the proper interplay between them remains a largely open problem.

We introduce a new model for real-time texture deformation, which is represented as a warping of the parametric domain onto itself. The novelty of our approach is to define the warping as the advection of the parametric domain in a flow field. This field is pre-computed and static. Dynamics is introduced by per-pixel integration time-steps, which makes possible to control the deformation locally. Our model comes with the following benefits:

- The deformation is content-aware. This is achieved thanks to the fact that everything is computed per-pixel in the parametric domain, on the basis of a vector field derived from the texture content itself. For instance, in Figure 1, the flowers are less deformed than the stretchable denim.

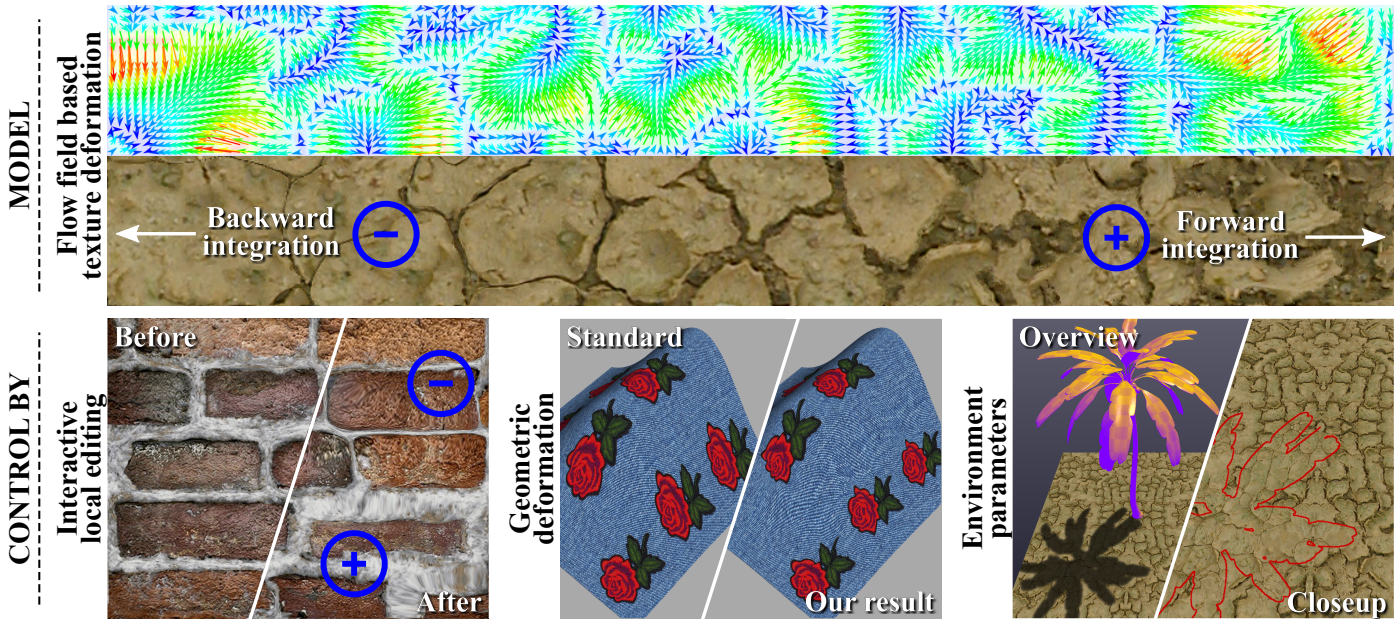

Fig. 1. Our content-aware texture deformation model improves the dynamic appearance of textured surfaces. It allows to mimic various non-uniform physical behaviors at texel resolution. Top row: the parameterization is advected in a static flow field; the resulting deformation depends on the speed and direction (forward/backward) of the integration along the streamlines. Bottom row, from left to right: the deformation can be controlled dynamically by brush painting during user interactive editing (in this example, backward integration narrows the mortar while forward integration widens it), by the geometric deformation of the underlying animated surface, or by any environment parameter (here, cracks enlarge only in regions of high sun exposure. The shadow contour is highlighted in red on the closeup view for a better visibility).

- The model is versatile. It spans multiple visual effects, such as non-homogeneous elasticity and feature shrinkage (see Figure 1). This is possible because we do not rely on a specialized physical simulation.
- The deformation is controlled locally and dynamically. It enables our model to be used within any interactive animation or editing framework. We show different scenarios, including automatic guidance from the local geometric deformation of the underlying surface, control through environment parameters, or interactive texture editing under user's control.
- Implementation within the standard rendering pipeline is straightforward and efficient. At each time step, all texels update their warping in parallel with a trivial integration scheme, while the rendering relies on standard interpolation and MIP-mapping techniques.
- Unpleasant visual distortions due to extreme local deformations are solved by combining a re-sampling technique and local detail synthesis, pre-computed and stored in a texture stack. We can thus compute a plausible appearance for thin connected structures, such as cracks or joints in cellular patterns.

Defining a flow field manually would be tedious and non intuitive. We provide a technique to derive the field from a scalar map. Such a scalar map, which is easier and more intuitive to draw for an artist, may represent, for instance, the expected texel rigidity in an elastic deformation scenario.

Our texture stack pre-integrates a warping with constant time-steps and re-samples the texture accordingly. Texture details are reintroduced in over-stretched regions by a state-of-the-art synthesis algorithm. At run-time, while the warping evolves, we keep track of the deformation magnitude at each texel so as to determine the most appropriate stack level to fetch.

In addition to the usual material texture layers, the simplest version of our model (i.e. without stack) only requires one additional map to store both the flow field and the warping. The final rendering then requires only one texture indirection to determine the actual texture coordinates from the warping, making the whole process easy to integrate to the graphics pipeline.

2. Related work

2.1. Image retargeting

Image retargeting is an editing approach that aims at producing a re-scaled version of an input image while preserving its most prominent features (which can be either detected automatically or specified by the user) by removing, duplicating or displacing some existing content [1, 2, 3]. It has been used, among other things, for the synthesis of architectural scenes [4, 5], where features of facades can be easily identified and extracted for editing purposes.

The goal of retargeting is then to preserve the visual consistency of the content regardless of the target image size, but not to deform it. Moreover, the approach is not intended for animation, which implies two crucial limitations for our case of interest. First, real-time is not viewed as a determinant factor. Secondly, no consideration is given to temporal consistency, leading to modifications that are not guaranteed to be continuous over time, reversible, or able to mimic existing dynamic

behaviors. On the contrary, real-time and temporal consistency are important in our case.

2.2. Texture advection

Texture advection has been used for texturing fluids whose dynamics are provided by an evolving geometry [6, 7, 8], a flow field [9], or a particle system [10, 11]. These methods tackle the problem of synthesising a globally and temporally coherent appearance from a small input texture exemplar. The main challenges are (i) to make weakly structured content coherently appear and disappear over time, and (ii) to handle topological changes. In contrast, our goal is to deform structured materials and we are concerned neither with run-time synthesis, nor content appearance and disappearance. We also rely on a fixed topology for both the surface and the texture. As a consequence, we use advection in a different way. While fluids advect independent patches of texture on a low resolution field, our flow field is defined at the same resolution as the texture and every texel is advected coherently with its neighbors.

An early method for texturing animated objects was introduced by Smets-Solanes [12] in the context of implicit surfaces. They develop the concept of a virtual skin, which supports the texture; the skin evolves in a vector field defined by the surface so as to stick to the surface in 3D. In our context, the vector field is defined by the texture so as to deform in the 2D parametric domain.

In the field of scientific visualization, advection of static textures has a long tradition: the texture is used as a metaphor to visualize dynamic flow fields [13]. Here, we conversely use a static flow field to control texture dynamics.

2.3. Texture warping for editing and synthesis

Warping has been used in the context of texture synthesis and editing. Brooks and Dodgson [14] make use of self-similarity measures to edit textures. One application among others consists in warping the texture so as to enlarge some features while shrinking some others. Similar results can be reproduced with our method (see Section 5.2), with the advantage that we provide local and dynamical control in real-time thanks to the use of the underlying flow field. Liu et al. [15] tackle the problem of synthesizing so-called "near-regular" textures: a regular pattern undergoes small deformations which are modelled as a random warping field. Since their goal is to synthesize random variety in static textures, they do address neither control nor dynamic effects.

2.4. Texture warping for dynamic deformations

Li et al. [16] simulate an elastic skin on an animated mesh. Since they run a physical simulation at mesh resolution, they have few control and they cannot adapt the deformation to texture content.

Content-aware deformation has been addressed in several works. Gal et al. [17] preserve the shape of some selected features when deforming texture maps. They rely on a geometric optimization process which reaches interactive frame-rate. A similar technique is used offline by Grabli et al. [18] on high-resolution textures in the context of post-production. Koniaris et al. [19] compute a non-linear geometric optimization at run time on a low resolution rectangular grid, so as to keep some texture features as-rigid-as-possible. Koniaris et al. [20] solve a position based dynamics problem to simulate heterogeneous elasticity of textures mapped on deforming surfaces. It is based on a quite heavy hierarchical solver on GPU. Their rigidity map inspired our flow field generation (Section 4).

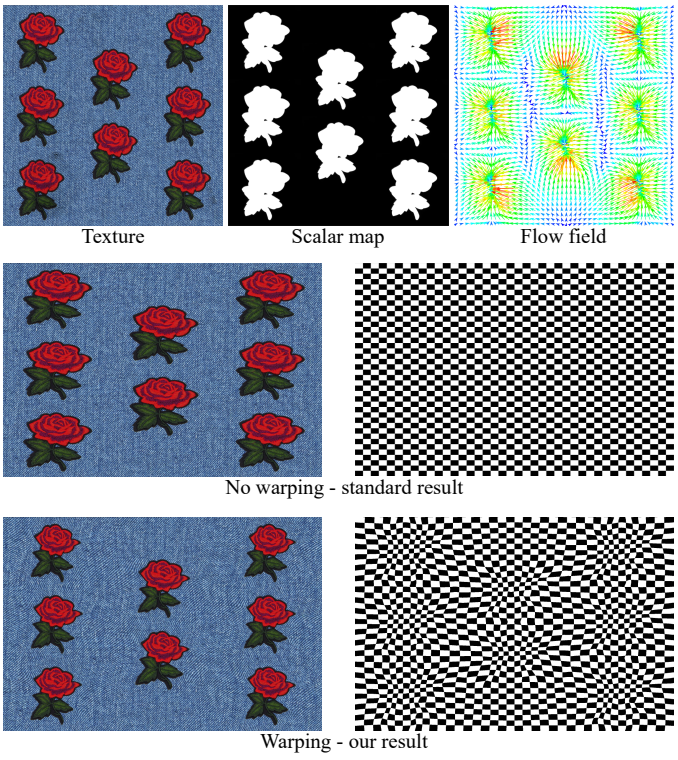

Fig. 2. A warping deforms the texture parametric domain so as to control the dynamic appearance. In this example the flowers remain nearly rigid during stretching. The warping is computed by integrating along streamlines of a static flow field. In turn, the flow field can be automatically derived from a scalar map.

These methods rely on real-time simulation or optimization, which requires to solve a global equation at each time step. While this is appropriate for interactive applications on medium-size data, it remains time consuming for high-resolution textures, and it is complex to integrate in the graphics pipeline. In addition, the deformation is only controlled indirectly through global parameters. By contrast, we provide direct and local control, and easy integration in the standard graphics pipeline. Run-time computations are purely local and confined in the fragment shader, while the global problem has been pre-processed when computing the flow field. It thus preserves the key advantage of texturing: to generate high frequency details in parallel on low frequency geometry. We are also not limited to a given physical phenomenon. Finally, our method successfully deforms thin connected features such as cracks or joints in cellular patterns, which are difficult to handle in numerical systems. This is made possible by our smooth pre-computed flow field which avoids real-time optimization.

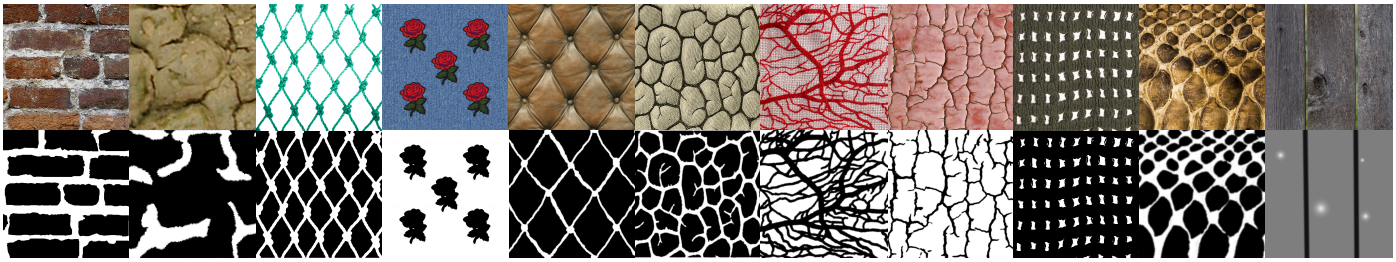

Fig. 3. Different input textures and their associated scalar maps. It can be seen that various kinds of contents can be deformed, including very thin structures. The right-most example illustrates the case of a scalar map which is not only binary.

3. Texture deformation

3.1. Motivation

Let $\mathcal{S}$ be a surface, and X its embedding into 3D, defined as :

$$p \xrightarrow{X} \mathbf{x} \tag{1}$$

where $p \in \mathcal{S}$ and $\mathbf{x} \in \mathbb{R}^3$ is a 3D position. When $\mathcal{S}$ is animated, its 3D geometry $X_t(\mathcal{S})$ depends on the time t.

Standard 2D texture mapping is defined through a composition:

$$p \xrightarrow{U} \mathbf{u} \xrightarrow{C} Color(p) \tag{2}$$

where U is the parameterization, $\mathbf{u}$ is a 2D parametric coordinate and C is the texture, stored as an image, which maps the parametric domain to a color. Usually, U and C are precomputed according to X. U is thus most often time invariant. The texture then simply follows the geometry which remains the only animated part. While this is perfectly acceptable for a flapping flag, this method typically produces visual artifacts as soon as surface deformation is not isometric. Figure 2 illustrates this problem: the texture color patterns (middle row) just follow the geometric deformation, as a skin pinned to each vertex of the mesh. This produces a distortion of any rigid pattern (here the flowers) embedded within the texture. Our purpose is to make parameterization time-dependent as well, so as to automatically maintain consistency between geometry and texture. The bottom row in Figure 2 illustrates this correction: the flower patches are kept almost rigid while the supposedly extensible fabric is distorted. Although standard software allows to keyframe texture coordinates so that parameterization may also be animated, manually specifying such key-frames independently from the animation of the surface is a tedious task, given that visual consistency with the deforming geometry needs to be maintained.

The main challenge is to automatically generate visually consistent texture deformations in the form of parameterization warpings, so as to avoid any need for manual keyframing of texture coordinates. While doing so, having the deformation depend on texture contents, e.g. generating different behaviours for pixels which represent different materials in the texture, is of particular interest. Our last requirement is to design a real-time method, allowing interactive animation and user control.

3.2. Deformation as a parametric warp

Our model is defined as:

$$p \xrightarrow{U} \mathbf{u} \xrightarrow{W_t} \mathbf{w} \xrightarrow{C} Color(p,t) \tag{3}$$

where U is the initial parameterization, C is the texture, and W_t is a time-dependent warping function from the parametric domain onto itself. During the animation, U and C are kept constant while W_t evolves.

Note that in our model, W_t is sampled on texels, which form a fine and regular grid, while U is sampled on mesh vertices, often coarse and irregular. This gives us two advantages over directly animating U. First, computations are made easier by the regular grid, and the implementation on GPU is straightforward. Second, the deformations are created at fine resolution, and can be set to depend on the texture. Indeed, we define W_t according to the texture content $C(\mathbf{u})$, which makes the deformation content-aware.

3.3. Warping as an advection in a flow field

The warping advects any parameter $\mathbf{u}$ along the streamlines of a flow field:

$$W_t(\mathbf{u}) = \mathbf{u} + \int_{s=0}^{t} \vec{\mathbf{v}}(W_s(\mathbf{u}))\mathrm{d}\mu \tag{4}$$

where:

- t is the temporal variable of the animation;
- $\vec{\mathbf{v}}$ is a static flow field;
- $\mathrm{d}\mu$ is the integration step which controls the warping.

The intuition of formula (4) is that the advection speed $\vec{\mathbf{v}}$ is modulated by $\mathrm{d}\mu$. In a standard advection, one would have $\mathrm{d}\mu = \mathrm{d}s$, so every texel $\mathbf{u}$ would be advected exactly at speed $\vec{\mathbf{v}}$ without control. By taking control over the integration step $\mathrm{d}\mu$ we introduce the possibility of modulating the speed, as if the texels could "swim" forward and backward along the streamlines. $\mathrm{d}\mu$ is defined at any time t (control across time) and any position $\mathbf{u}$ (control across space). In Section 5, we present different scenarios to derive $\mathrm{d}\mu$ from the geometry, from an interactive tool, or from environment parameters.

An advantage for $\vec{\mathbf{v}}$ to be static is that it can be pre-computed (see Section 4) and loaded on the GPU. Thus the warping W_t is updated using a discretization of equation (4), computed at each

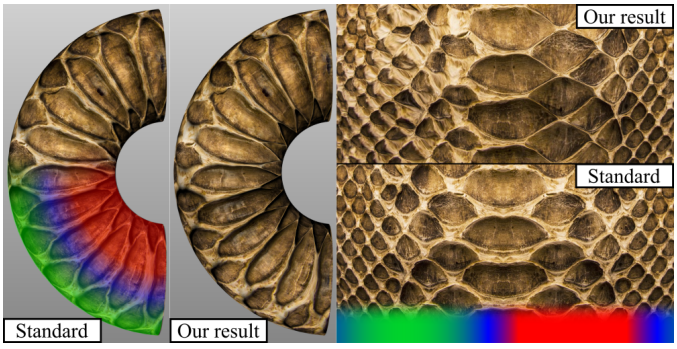

Fig. 4. Left: a bent cylinder textured with snake scales, which are deformed according to the tensor. Tensor magnitude is illustrated at the bottom (blue: no deformation; green: stretch; red: compression). Right: texture visualized in parametric space. Top: with warping. Bottom: without warping.

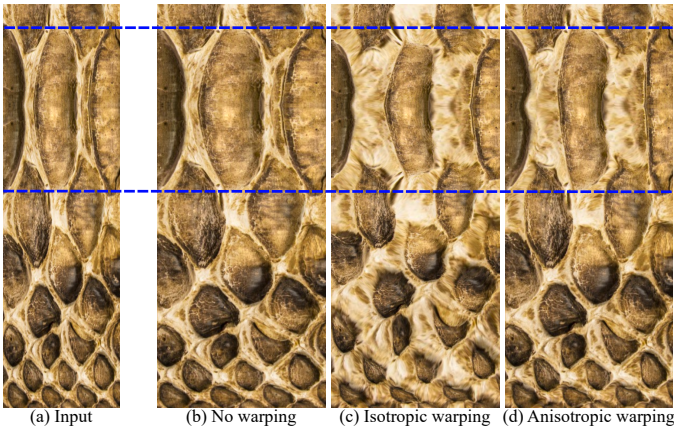

Fig. 5. An input texture (a) undergoes a geometric stretching. Results with several controls (b,c,d) are compared. (b) no warping: scales and skin are stretched identically. (c) isotropic control: skin is more stretched ; scales are deformed both horizontally and vertically. (d) anisotropic control: the deformation preferably follows the stretch direction (horizontal).

frame for all texels in parallel during a single off-screen rendering pass. This is a major advantage against real-time simulation techniques, which have to solve a global equation at each time step.

The warping is actually computed using

$$W_{t+dt}(\mathbf{u}) - W_t(\mathbf{u}) \approx \vec{\mathbf{v}}(W_t(\mathbf{u}))\mathrm{d}\mu, \tag{5}$$

which is a trivial discretization of Equation (4). It proved to be sufficient in our tests, though more sophisticated integration schemes could improve the precision.

4. Flow field generation

Designing an appropriate flow field $\vec{\mathbf{v}}$ is a delicate task. In this section, we propose a technique that avoids a painful manual drawing of a *vector* field. Instead, the user provides a *scalar* map R from which the flow field is computed automatically by solving the equation:

$$\mathrm{div}(\vec{\mathbf{v}}) = R \tag{6}$$

The intuition is driven by an analogy with fluids. Regions with positive divergence (e.g. sources) repulse the parameterization, so that the texture shrinks when advected. Conversely, a negative divergence (e.g. sinks) attracts the parameterization and stretches the texture. We ask the user to provide the scalar map R, $-1 \leq R(\mathbf{u}) \leq 1$ for every texel $\mathbf{u}$. It represents the stretched versus shrunk regions. Figure 3 shows the textures we used to illustrate this paper, as well as the corresponding scalar maps R that we drawn using standard image editing tools. Asking as input a scalar map instead of a vector field enables to benefit from many existing advanced segmentation tools, like self-similarity measures [14] for instance, to further simplify the design process.

In a scenario where an elastic deformation is driven by an animated geometry, as in Figure 2, R can be interpreted as a rigidity map, set to 1 on rigid regions and to -1 on soft regions. Indeed, if the geometry is stretched, then a standard texture mapping would stretch all texels the same way (Figure 2 middle). To mimic a more physical behavior, we would like the fabric to be extensible while the flowers remain nearly rigid (bottom row). To make patterns to appear rigid in 3D, the geometric stretch has to be compensated by a texture shrinkage, which corresponds to an expansion in the parametric domain.

In practice, we compute the field $\vec{\mathbf{v}}$ by solving Equation (6) with Dirichlet boundary conditions: we set $\vec{\mathbf{v}}(\mathbf{u}) = \vec{0}$ for all texels $\mathbf{u}$ on the boundary of the parametric domain, or of each chart of the texture atlas. Equation (6) is expressed on the dual grid : R is linearly interpolated, and $\mathrm{div}(\vec{\mathbf{v}})$ is computed with finite differences. The resulting system has less equations than unknowns, yet it has no exact solution in general – this is due to the Dirichlet conditions and the divergence theorem. We solve this system in the least-square sense using a numerical solver. This computation is done only once, as a pre-processing, since only the resulting flow field is required for our real-time deformation.

Notice that the interval $[-1; 1]$ for R is arbitrary. Scaling R would scale $\vec{\mathbf{v}}$, which can be compensated by inverse scaling of $\mathrm{d}\mu$. In practice we adjust manually the scaling of $\mathrm{d}\mu$.

5. Warping control

The advection – and thus the deformation – is controlled by the integration time-steps $\mathrm{d}\mu$ which are defined per pixel. This allows for local and dynamic control. We propose several ways to control texture deformation, motivated by various applications, and discuss results. In an animation scenario, we compute the time-steps from the underlying geometry. In an editing scenario, we define the time-steps by a simple brush. Lastly, we illustrate control by environment parameters.

5.1. Control by geometry deformation

Consider a common situation: a finely detailed texture is mapped on a coarsely triangulated surface which is animated. We want to derive $\mathrm{d}\mu$ from the time-varying geometry X_t.

Surface deformation can be expressed locally by the tensor matrix, which captures the geometric strains. Let T_t be the 2×2 matrix describing the deformation of a triangle between consecutive frames $t - \mathrm{d}t$ and t (see Appendix A for details). To

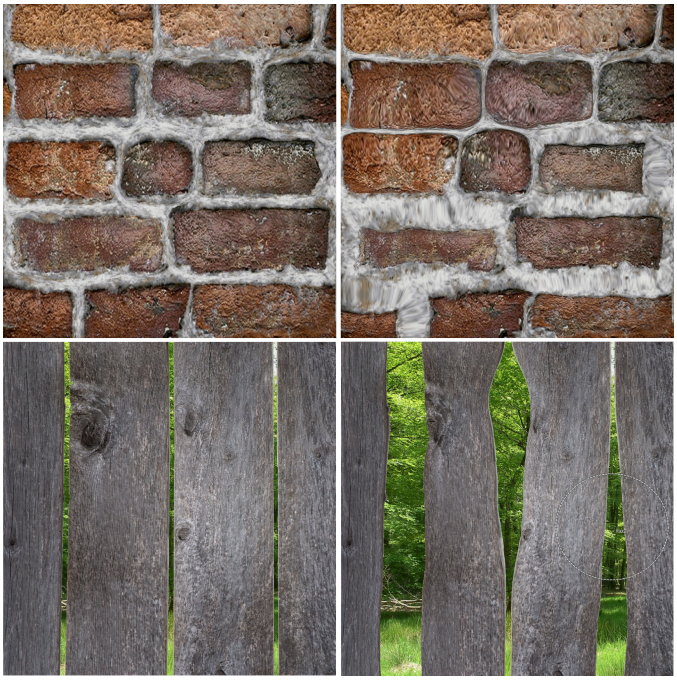

Fig. 6. Texture deformation controlled by a brush during an interactive mapping editing session. Left: reference images. Right: results after editing. First row: The mortar has been either stretched or compressed, in order to produce a user-desired, space-varying effect. Second row: the editing process is applied to the foreground texture, a plank wall, which lets appear a background image.

take into account this tensor, we experimented with various formulas. In the following we propose two of them, which have different properties.

Isotropic control. We use the differential norm of the accumulated tensors as a measurement of the stretch magnitude. Let $A_t = T_t \circ T_{t-\mathrm{d}t} \circ \ldots \circ T_0$ be the accumulated tensor at frame t, we define the control parameter as:

$$\mathrm{d}\mu = (\|A_t\| - \|A_{t-\mathrm{d}t}\|)\,\mathrm{d}t \tag{7}$$

where $\|\ \|$ represents the Frobenius norm. This expression was set to meet the following properties :

1. A static mesh ($T_t = Id$) implies no texture deformation ($\mathrm{d}\mu = 0$).
2. The warping is invariant under time step decomposition. In other words, the accumulation of many small steps is equivalent to one large step.
3. The resulting warping is independent of the geometric path. That is, two triangle trajectories that end at the same position produce the same warping. In particular, this ensures reversibility: if the mesh gets back to its rest shape, then the warping gets back to its initial value.

Figure 4 illustrates our method on a snake skin texture: with a standard mapping (no warping), skin and scales are stretched identically ; with our method, the soft skin stretches a lot while the rigid scales deform as little as possible. This can be seen also in the accompanying video.

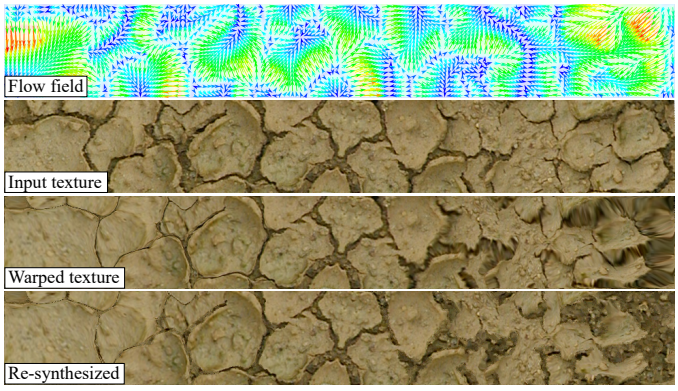

Fig. 7. The flow field $\vec{\mathbf{v}}$ associated to an input texture is integrated, resulting in a warped texture with shrinking or stretching cracks, depending on the sign of the integration steps $\mathrm{d}\mu$. These distortions are removed by the use of our re-synthesised texture stack.

While this produces satisfactory results for nearly isotropic deformations, this control has a limitation: different geometric deformations with the same area change produce the same warping, as $\mathrm{d}\mu$ is not sensitive to the stretching direction. This can be seen on Figure 5-c: while the geometric stretch is only horizontal, the scales slightly shrink vertically. As shown next, introducing anisotropic behavior is possible but at the price of losing the above property 3.

Anisotropic control. Sensitivity to the direction can provide more physically plausible behavior. For instance, when stretching along a given direction, one may expect the texture to warp in that direction while remaining rigid in the orthogonal one. The following expression for $\mathrm{d}\mu$ achieves this:

$$\mathrm{d}\mu = \log\left(\bar{\mathbf{v}}^T T_t \bar{\mathbf{v}}\right)\mathrm{d}t, \quad \text{with } \bar{\mathbf{v}} = \frac{\vec{\mathbf{v}}}{\|\vec{\mathbf{v}}\|}. \tag{8}$$

The dot product between the normalized flow $\bar{\mathbf{v}}$ and $T_t\bar{\mathbf{v}}$ transformed by the geometric tensor produces a stronger deformation along the tensor main stretch direction. The log function is used to guarantee the above properties 1 and 2. As shown in Figure 5-d, the anisotropic behavior is improved.

This type of parameter control comes, however, with a drawback. Since $\mathrm{d}\mu$ depends on the flow $\vec{\mathbf{v}}(W_t(\mathbf{u}))$ at the current warped parameter, the result depends on the trajectory and property 3 is lost. As a consequence, the texture deformation achieved this way is not reversible.

Note that we make no assumption on how the animation is computed, e.g. skinning, key-frames, simulation, or optimization. Whatever the geometric model, the control is unchanged. Thus our model could be combined with triangle-based simulations such as the skin model of Le et al. [16].

5.2. Control by texture mapping editing

Let us consider the following content creation scenario. A mesh has been parameterized, and a texture mapped onto it. Since the mesh is not developable, distortions of color patterns are introduced. Our model provides a powerful tool for artists to improve the texture mapping at texel resolution without changing mesh unfolding and texture coordinates.

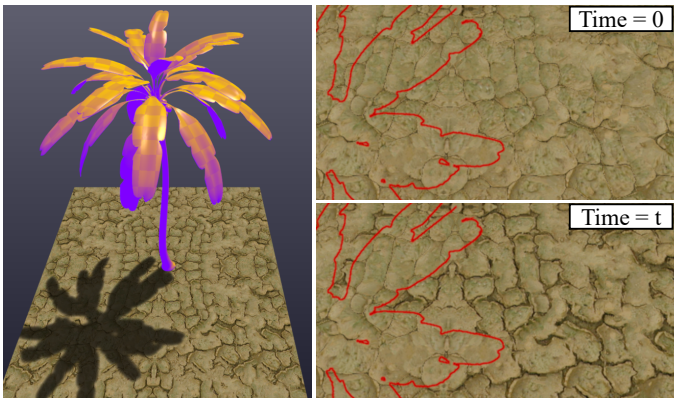

Fig. 8. Example of control by environment parameter. The ground texture flow field is integrated everywhere but in the palm tree shadow, making cracks to enlarge only in areas of high sun exposure, mimicking a drying ground. *Right*: comparison between the initial texture state (top) and the warped texture after several integration steps (bottom). The shadow contour is highlighted in red.

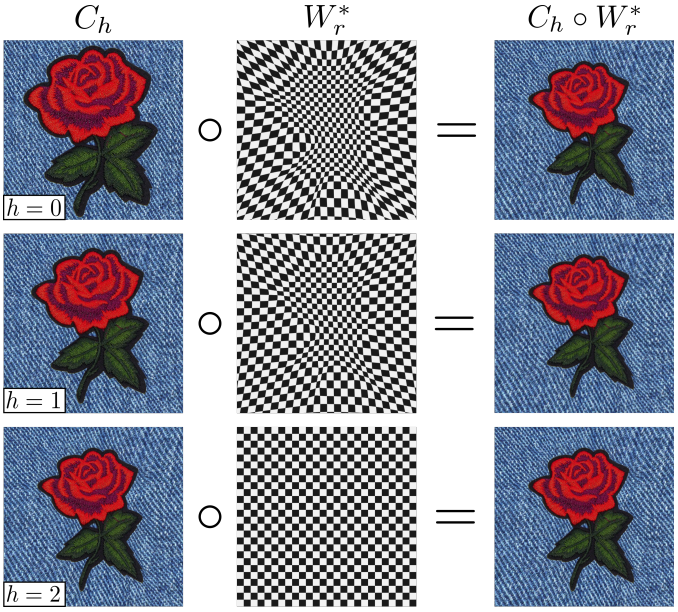

Fig. 9. Left: texture stack C_h. Middle: residual warping W_r^*. Right: resulting $C_h \circ W_r^*$. See how changing h reduces the residual warping while providing an identical result.

To illustrate this use of our method, we designed a tool that modifies warping on-the-fly thanks to an interactive brush. The value of $d\mu$ is then a simple Gaussian weight centered around the cursor (see Figure 6 and the accompanying video). Similarly to the aforementioned anisotropic control, other heuristics that account for the current direction of $\vec{\mathbf{v}}$ in the computation of $d\mu$ might be considered as well, in order to add directional effects to the editing brush.

Approaches have been proposed to optimize mapping with respect to texture content [21], but since they act on texture coordinates only, correction is only possible at triangle level. Combining them with our method, which provides manual control at texel resolution, might thus ease the task of texture mapping by allowing fine editing or the achievement of artistic effects by the user.

5.3. Control by environment parameters

The warping can be controlled by any kind of external parameters. To illustrate this, let us consider the example shown in Figure 8, where an occluder (the palm tree) is used to produce a shadow on a ground plane by shadow mapping. Flow field integration is directly guided by the shadow map, using its value to set the step $d\mu$ and making the ground texture to deform with respect to sun exposure: cracks enlarge only in regions out of the shadow. It must be noticed that control parameters can be changed interactively: moving the light and changing the shadow makes the deformation to immediately adapt to the new configuration. This example illustrates also the easiness of implementation and integration of our model in a GPU rendering pipeline.

6. Re-sampling and detail synthesis

The model presented so far behaves well for smooth deformations, which requires both $\vec{\mathbf{v}}$ and $d\mu$ to be smooth over the domain. This is acceptable for data such as the flowers in Figure 2. However we want to treat more challenging data, such as the mortar between the bricks in Figure 6 or the cracks in Figure 7. In these examples, thin features undergo extreme local deformations, which causes two problems: the sampling is irregular and the details are over-stretched. Our goal is to get the fourth row in Figure 7 instead of the third one. We address these problems by pre-computing a stack, which stores a small set of textures (about 10), using re-sampling and local detail synthesis.

6.1. Texture re-sampling

The warping W_t is arbitrary, resulting in an uneven sampling of the texture on the surface. This may produce artifacts when using a standard anisotropic filtering based on MIP-maps, because the footprint of a projected pixel may encompass texels of various sizes. Our idea is to re-sample $C \circ W_t$ on a regular grid. The difficulty comes from W_t, which is controlled pixel-wise in real time. On one hand, *on-the-fly* re-sampling is too time consuming. On the other hand, *pre-computed* re-sampling is not tractable since $d\mu$ is not known a priori – only $\vec{\mathbf{v}}$ is known.

Our solution consists in pre-integrating the warping at discrete time steps, re-sampling the result and storing it in a texture stack. This stack is then used at run-time to reduce sampling unevenness: for each texel, the warping is decomposed into a pre-integrated warping (the closest stack level) plus a much lower residual warping. We now show how to decompose the warping into these two parts (pre-integrated and residual), and provide the algorithm for run-time handling.

A reference warping W^* that is context-independent, i.e. with no external parameter in $d\mu$ is thus pre-integrated at discrete time steps and stored as a texture stack $\{C_h\}_{0 \le h \le H}$, where $C_0 = C$ and every texture

$$C_h(\mathbf{u}) = C_0 \circ W_h^*(\mathbf{u}) \tag{9}$$

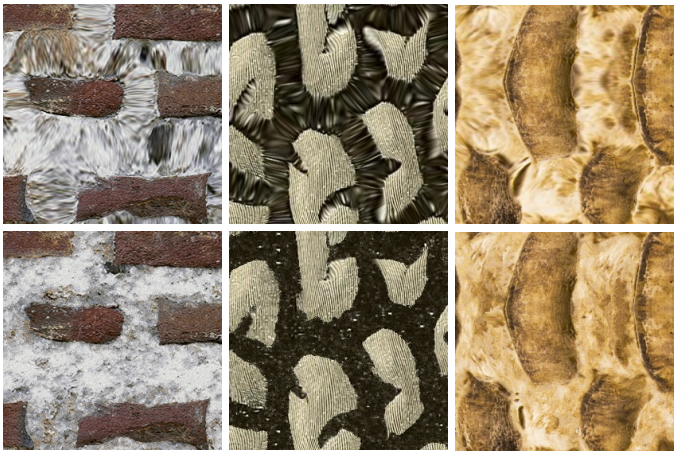

Fig. 10. Close-up on a stretched surface area with a warped texture. *Top*: standard warping with resampling ($C_h \circ W_r^*$) *Bottom*: warping with resampling and synthesis in the stretched regions. Details are improved in stretched regions thanks to texture synthesis.

is sampled on a regular grid, as shown in Figure 9 left. We achieve this by integrating according to time only, i.e. $\mathrm{d}\mu = \mathrm{d}s$:

$$W^*_{t^*}(\mathbf{u}) = \mathbf{u} + \int_{s=0}^{t^*} \vec{\mathbf{v}}(W^*_s(\mathbf{u}))\mathrm{d}s \tag{10}$$

Here, t^* behaves like a "reference time" which is related to the "real time" t by

$$t^*(t) = \int_{s=0}^{t} \mathrm{d}\mu, \quad \text{and} \quad W^*_{t^*(t)} = W_t \tag{11}$$

Without loss of generality, we assume that the interval between h and $h+1$ is one unit of the reference time, ie. $h = t^*$. Then, during animation at time t, we decompose the warping into a pre-integrated part $W^*_{h(t)}$ and a residual warping part $W^*_{r(t)}$ such as:

$$W_t = W^*_{t^*(t)} = W^*_{h(t)} \circ W^*_{r(t)} \tag{12}$$

where $t^*(t) = h(t) + r(t)$. By combining Equations (9) and (12), we can rewrite

$$Color(p,t) = C \circ W_t \circ U(p) \tag{13}$$

enabling the color to be computed as:

$$Color(p,t) = C_{h(t)} \circ W^*_{r(t)} \circ U(p) \tag{14}$$

where $C_{h(t)}$ has been pre-computed in the stack, and $W^*_{r(t)}$ is a much smaller warping than W_t. As shown in Figure 9, by moving up and down in the stack, we are able to reduce the residual warping.

The challenge is now to compute $W^*_{r(t)}$ in real time during an animation or modeling session. Our insight to achieve this is as follows. For every texel:

- We maintain a level h in the stack, which is incremented if $t^* \geq h+1$, decremented if $t^* \leq h-1$.
- $W^*_{r(t)}$ is updated similarly to W_t, except that it is reset when the level is incremented or decremented.

Fig. 11. On-the-fly correction of under-sampling occurring in the over-stretched regions of a deforming geometry. *Left*: no correction. *Right*: the re-synthesized texture stack is used.

The detailed algorithm is given in Appendix B. An advantage of our technique is to be compliant with hardware-based anti-aliasing. Indeed, MIP-map is turned on for all color maps C_h. Then, for every surface point p, the colors of the four neighbors of $\mathbf{u}(p)$ are computed independently using Equation (14) and bi-linearly interpolated.

6.2. Synthesizing details

The texture sampling is now quite regular whatever the deformation, so we can address the problem of stretched details. Visual artifacts appear in over-stretched areas during the forward integration (i.e. from C_0 to C_H) due to an enlargement of pixel contents. Compressed areas on the contrary gather pixel contents and thus are free from these artifacts. To solve this, we propose to synthesize details in the stretched parts of the stack (Figure 10). The key point is to maintain spatial coherence between stretched and compressed parts of each level, as well as the temporal coherence across levels.

Our insight is to use the backward integration (i.e. from C_H to C_0), where stretched areas behave as compression. We first create a bottom-up stack using forward integration, we re-synthesize the stretched parts of the very last level (C_H), then we create a second top-down stack using backward integration from the new C_H. Each slice of texture stacks resulting from the backward and forward integration are then linearly blended, so as to keep from each only the regions corresponding to compressed area. We thus guarantee that the maximum details are preserved at each stack level. We provide the details of our algorithm in a supplemental material.

The choice of the synthesis algorithm may vary depending on the input and the desired quality. In our case, we used a modified version of [22], which has the advantage of being fast and able to preserve sufficiently well the structure and spatial organization of our inputs.

7. Results

We have shown many results throughout the paper involving color maps (Figures 1 to 10). Figure 12 shows two other textures mapped on an animated bouncing cube (see also the accompanying video). As can be seen, the texture deforms in

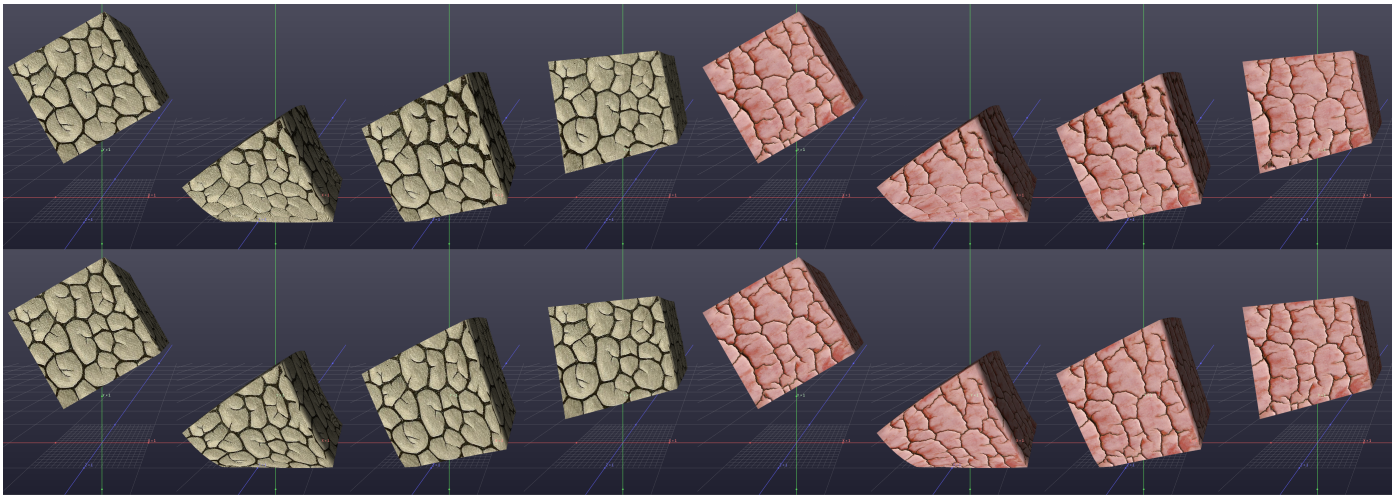

Fig. 12. A few frames of a soft bouncing cube with texture deformation guided by the underlying surface animation. Top row: our texture deformation model. Bottom row: standard texture mapping. Dark parts, defined as elastic, shrink on impact with the ground and stretch after the rebound, when the object tends to recover its rest shape.

Texture	Geometry	Standard	Warping		+ Resynth	
			1 step	10 steps	1 step	10 steps
Bricks (0.32M texels)	Plane (2 triangles)	0.53	0.63	0.65	0.67	0.69
	Cylinder (3.2K tri.)	0.54	0.64	0.67	0.68	0.72
	Inflating (4.9K tri.)	0.70	0.79	0.83	0.84	0.88
Snake (1M texels)	Plane	1.04	1.20	1.23	1.34	1.36
	Cylinder	1.08	1.18	1.23	1.29	1.33
	Inflating	1.41	1.56	1.58	1.60	1.71
Denim (3.24M texels)	Plane	2.72	3.03	3.09	---	---
	Cylinder	2.81	2.97	3.02	---	---
	Inflating	3.53	3.79	3.77	---	---

Table 1. Rendering time for different textures and different geometries (in milliseconds). "Standard" refer to a standard rendering without texture deformation, "Warping" to the basic version of our model, and "+Resynth" to the version using the texture stack for detail synthesis. "Steps" indicates the number of integration steps performed per-pixel in the fragment shader at each frame.

a consistent manner, elastic texels being shrunk on geometry compression and stretched on extension. Our method is oblivious to the type of data stored in the texture: in Figure 6 we present a texture with transparency; in Figure 13 the deformation is also applied to a displacement map.

7.1. Detail synthesis

As shown in Figure 3, some of the textures to which we applied our method are critical for deformation scenarios, since the stretched parts are made of very thin structures separating big rigid cellular patterns. In such cases, strong sampling artifacts appear, as a very few number of pixels cover large surface areas, as illustrated in the upper row of Figure 10.

Such textures are not treated by previous methods, which mainly focus on textures that exhibit small rigid features within large elastic regions (as in Figure 15). One reason is that they rely on physical simulations: fine features may not be captured by the simulation grid, or it may cause numerical problems. Another reason is that sampling issues are not addressed.

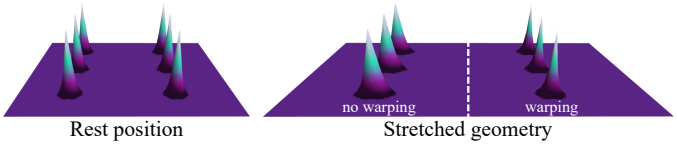

Fig. 13. A deformation is applied to both the color and displacement map channels. As it can be noticed on the right side of the stretched geometry, the warping allows to preserve the spikes thickness while stretching the supporting geometry.

Conversely, the use of our re-synthesized stack enables to correctly manage this critical type of textures, by preserving details either in the compressed or stretched regions, as shown in Figure 10 bottom. A comparison of our method on a deforming geometry with and without the use of the stack is illustrated in Figure 11.

7.2. Performances

Rendering times are given in Table 1, obtained on an Intel(R) Core(TM) i7-7700K, 4.20GHz, with a GeForce GTX 1060 6GB. Measurements have been made by controlling the warping with an animated geometry, and include the time required for rendering as well as for updating the warping, averaged over 1000 frames. Both are performed entirely on GPU, using a shader programming language. We compared three different cases: a standard rendering without texture deformation, and two variants of our flow field based model: with and without the re-synthesized texture stack. It must be mentioned that for numerical accuracy reasons, we subdivide the time-steps used during the flow field integration of each rendered frame into smaller steps (usually 10). This is done directly in the fragment shader by consecutive fetches of the flow field texture. We compared timings with and without this subdivision.

Compared to the standard rendering, the additional cost induced by our approach is mainly due to the fact that the deformation for the whole texture must be updated at each frame: every texel has thus to be processed at least once to integrate

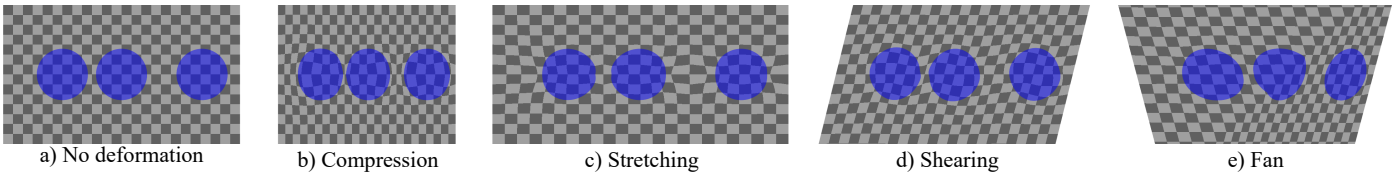

Fig. 14. Illustration of the behaviour of our algorithm (anisotropic control). The texture (a) is pre-processed with a map $R = 1$ inside the blue disks, $R = -1$ outside. It is mapped on a quad made of two large triangles, which undergoes pure compression (b), stretching (c), and shearing (d). The fan deformation (e) is a mix of shearing and stretching.

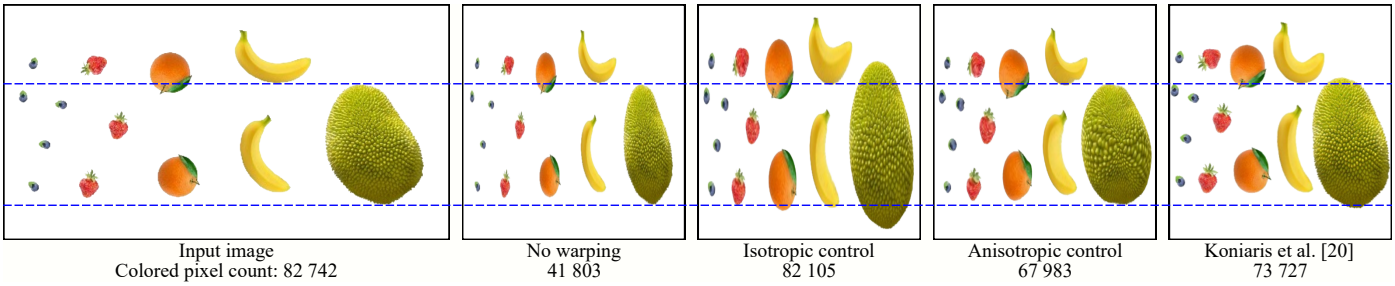

Fig. 15. Comparison with Koniaris et al. [20]. From left to right: input texture; standard deformation (50% horizontal compression); our warping (isotropic control); our warping (anisotropic control); [20] results.

the flow field, conversely to the standard rendering, where only visible texels are processed. This is the minimal expected cost for a per-pixel texture deformation. Considering the time-step subdivision, one can note that it has a very little impact on performances, despite the much more important texture fetch count it requires. This is due to cache-coherency: since our time-steps are very small, texture fetches are located very close to each other in the parametric space. It can be seen that the use of our stack for detail re-synthesis has almost no impact on the global rendering time. The number of stack levels is relatively low (10 in our examples) which avoids excessive cache-misses due to frequent texture switches, and only one is used by each texel to define the reference pre-integrated warping. We did not include timings for the denim example using re-synthesis because of their irrelevancy: texture resolution is high and the deforming areas do not consist in thin structures that need to be re-sampled. The relative overhead is about 10 to 20%, and it decreases as the texture size grows.

In terms of memory, instead of a single texture (C) for the standard rendering, our model requires an additional $\mathbb{R}^4$ texture to store both the flow field ($\vec{\mathbf{v}}$) and the warping (W_t). In the case where the re-synthesized stack is used, we need as many textures as there are stack levels, as well as an additional one to manage the stack (h and W_r^*). More precisely, if N is the texel count of the input texture and L the number of stack levels, the memory footprint of our model is of $19N$ bytes without stack, and $(24 + 3L)N$ bytes when using the stack, against $3N$ for a classical albedo rendering. As an example, the snake skin texture of 992×992 texels requires $18Mb$ without stack and $50Mb$ with a stack of 10 levels against $3Mb$ for a standard rendering.

7.3. Discussion

Figure 14 illustrates the behaviour of our algorithm and its limitations. The texture is pre-processed with a scalar map $R = 1$ inside the blue disks (rigid) and $R = -1$ outside (stretchable) so as to keep the disks as rigid as possible. Pure compression, stretching and shearing (b, c, d) show the interaction between rigid regions: the two left-most disks, which are closer to each other, are more deformed. This is due to the computation of $\vec{\mathbf{v}}$, which optimizes a global equation over the domain. The fan deformation (e) makes apparent the underlying geometry made of two triangles. Each triangle undergoes a different deformation, which implies a different warping. The discontinuity on the edge is due to the parameterization – U in equation (3) – which is not smooth. To improve this result, it could be useful to smooth U, possibly depending dynamically on the geometry X. Another option would be to account for U and X while computing the vector field.

Figure 15 shows a comparison between our method and the physical simulation of Koniaris et al. [20], which also deforms textures at fine resolution. Even if our isotropic control well preserves the area of the fruits, it changes their shape. Anisotropic control, in turn, strictly preserves the vertical sizes but internal micro-patterns are distorted compared to [20]. As expected, we cannot guaranty physical realism but only plausible deformations.

However, our model comes with several advantages over Koniaris et al. [20]:

1. Our method is easy to integrate in the graphics pipeline because it avoids numerical simulation or optimization at run-time.
2. Our method has a constant cost per texel. Conversely, in [20], the cost depends on the resolution of the simulation grid, which may not coincide with the texture resolution. Therefore, it may hinder a correct management of very thin deforming structures at texel level.
3. All examples derived from [20] (including Figure 15) exhibit rigid features on smooth stretchable backgrounds.

Thus, in the case of thin structures and/or strong deformations, the effect of deformations on the background are totally hidden. As already shown in Figures 7 and 10, our method successfully handles such challenging cases.

4. Control in [20] is possible only through geometry deformation, contrarily to our approach, where texture deformation can be controlled locally, by any kind of parameters, as illustrated by Sections 5.2 and 5.3.

8. Conclusion

We presented a novel model for content-aware texture deformation at texel resolution, based on the advection of the parametric domain in a pre-computed flow field. The deformation can be controlled locally and dynamically using various criteria, such as geometry, interactive editing, or environment parameters. Integration in the graphics pipeline is easy, and both memory and computation loads are kept low. We showed various examples, including challenging thin structures with strong deformations. To improve the appearance of over-stretched features we proposed a technique for re-sampling and detail synthesis, based on a texture stack. To ease the computation of the flow field, we derive it automatically from a scalar map.

In the future we would like to explore alternative techniques for building the flow field, such as texture key-frames combined with an inverse warping problem or offline simulations. It could also be interesting to consider boundary constraints other than the Dirichlet ones, so as to allow dynamic behaviours such as sliding, which could lead to visual effects similar to the ones presented in [16].

We believe that our model can inspire future research about dynamic phenomena including both space and time variations. It would be interesting to enhance our deformation model with the appearance and disappearance of new features (such as cracks), with topology changes (e.g. to represent lava flows), or with weathering techniques [23, 24] to represent progressive changes for static objects (e.g. rust, moist, or dust accumulation).

Another direction is on-the-fly dynamic texture synthesis, i.e. including our deformation model into a real-time synthesis framework. This could be useful to texture infinite non parameterized geometry, such as an on-the-fly procedurally generated landscape.

Acknowledgements

This work has been partially funded by the project HDWorlds from the Agence Nationale de la Recherche (ANR-16-CE33-0001) and by the advanced grant 291184 EXPRESSIVE from the European Research Council (ERC-2011-ADG 20110209).

References

[1] Avidan, S, Shamir, A. Seam carving for content-aware image resizing. ACM Transactions on Graphics 2007;26(3).

[2] Cho, TS, Butman, M, Avidan, S, Freeman, W. The patch transform and its applications to image editing. IEEE Conference on Computer Vision and Pattern Recognition 2008;32.

[3] Pritch, Y, Kav-Venaki, E, Peleg, S. Shift-map image editing. Proceedings of the IEEE International Conference on Computer Vision 2009;:151 – 158.

[4] Lefebvre, S, Hornus, S, Lasram, A. By-example synthesis of architectural textures. ACM Transactions on Graphics (SIGGRAPH Conference Proceedings) 2010;.

[5] Cabral, M, Lefebvre, S, Dachsbacher, C, Drettakis, G. Structure-preserving reshape for textured architectural scenes. Computer Graphics Forum 2009;28:469–480.

[6] Bargteil, AW, Sin, F, Michaels, JE, Goktekin, TG, O'Brien, JF. A texture synthesis method for liquid animations. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation. 2006, p. 345–351.

[7] Kwatra, V, Adalsteinsson, D, Kim, T, Kwatra, N, Carlson, M, Lin, MC. Texturing fluids. IEEE Transactions on Visualization and Computer Graphics 2007;13(5):939–952.

[8] Gagnon, J, Guzmán, J, Vervondel, V, Dagenais, F, Mould, D, Paquette, E. Distribution update of deformable patches for texture synthesis on the free surface of fluids. Computer Graphics Forum 2019;38:491–500.

[9] Kwatra, V, Essa, I, Bobick, A, Kwatra, N. Texture optimization for example-based synthesis. ACM Transactions on Graphics 2005;24(3):795–802.

[10] Yu, Q, Neyret, F, Bruneton, E, Holzschuch, N. Lagrangian texture advection: Preserving both spectrum and velocity field. IEEE Transactions on Visualization and Computer Graphics 2011;17(11):1612–1623.

[11] Gagnon, J, Dagenais, F, Paquette, E. Dynamic lapped texture for fluid simulations. The Visual Computer 2016;32(6-8):901–909.

[12] Smets-Solanes, JP. Vector field based texture mapping of animated implicit objects. Computer Graphics Forum 1996;15(3):289–300.

[13] Laramee, RS, Hauser, H, Doleisch, H, Vrolijk, B, Post, FH, Weiskopf, D. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. Computer Graphics Forum 2004;.

[14] Brooks, S, Dodgson, N. Self-similarity based texture editing. ACM Transactions on Graphics 2002;21(3):653–656.

[15] Liu, Y, Lin, WC, Hays, J. Near-regular texture analysis and manipulation. ACM Transactions on Graphics 2004;23(3):368–376.

[16] Li, D, Sueda, S, Neog, DR, Pai, DK. Thin skin elastodynamics. ACM Transactions on Graphics 2013;32(4):49:1–49:10.

[17] Gal, R, Sorkine, O, Cohen-Or, D. Feature-aware texturing. In: Proceedings of the 17th Eurographics Conference on Rendering Techniques. 2006, p. 297–303.

[18] Grabli, S, Sprout, K, Ye, Y. Feature-based texture stretch compensation for 3d meshes. In: ACM SIGGRAPH 2015 Talks. 2015, p. 71:1–71:1.

[19] Koniaris, C, Cosker, D, Yang, X, Mitchell, K, Matthews, I. Real-time content-aware texturing for deformable surfaces. In: Proceedings of the 10th European Conference on Visual Media Production. ACM; 2013, p. 11.

[20] Koniaris, C, Mitchell, K, Cosker, D. Real-time variable rigidity texture mapping. In: Proceedings of the 12th European Conference on Visual Media Production. 2015, p. 5:1–5:10.

[21] Jin, Y, Shi, Z, Sun, J, Huang, J, Tong, R. Content-aware texture mapping. Graphical Models 2014;76:152–161. Proceedings of Computational Visual Media Conference 2013.

[22] Lefebvre, S, Hoppe, H. Parallel controllable texture synthesis. ACM Transactions on Graphics 2005;24(3):777–786.

[23] Mérillou, S, Ghazanfarpour, D. A survey of aging and weathering phenomena in computer graphics. Computers & Graphics 2008;32(2):159–174.

[24] Lu, J, Georghiades, AS, Glaser, A, Wu, H, Wei, LY, Guo, B, et al. Context-aware textures. ACM Transactions on Graphics 2007;26(1):3.

[25] Pennec, X, Fillard, P, Ayache, N. A riemannian framework for tensor computing. International Journal of Computer Vision 2006;66(1):41–66.

Appendix A. Computing the geometric tensor

We need to represent the geometric deformations consistently over both time and space. To achieve this, we compute a tangent frame $(\tilde{\mathbf{f}}_x, \tilde{\mathbf{f}}_y)$ for each triangle, based on its parameterization. Then the 2D deformation tensors T_t are computed in this basis.

Let us consider a mesh triangle, with u_0, u_1, u_2 the texture coordinates in $\mathbb{R}^2$ and $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ the positions in $\mathbb{R}^3$ of each of its three vertices. The mapping from the parametric space to the (tangent plane of) the 3D triangle is then given by $\mathbf{x} = Fu$ where:

$$F = [\mathbf{x}_1 - \mathbf{x}_0,\ \mathbf{x}_2 - \mathbf{x}_0]\,[u_1 - u_0,\ u_2 - u_0]^{-1} \tag{A.1}$$

The columns $(\mathbf{f}_x, \mathbf{f}_y)$ of F form a basis of the tangent plane, which is the mapping of the canonical basis of the parametric domain. This basis is orthonormalized using the Gram-Schmidt algorithm, resulting in $\tilde{\mathbf{f}}_x$ and $\tilde{\mathbf{f}}_y$, which represent the tangent and bi-normal vectors.

Let e_1^t and e_2^t denote the coordinates in $(\tilde{\mathbf{f}}_x, \tilde{\mathbf{f}}_y)$ of the edge vectors $(\mathbf{x}_1 - \mathbf{x}_0)$ and $(\mathbf{x}_2 - \mathbf{x}_0)$ at animation frame t. The tensor T_t representing the triangle deformation between consecutive frames $t - \mathrm{d}t$ and t is given by:

$$T_t = \left[e_1^t,\ e_2^t\right]\left[e_1^{t-\mathrm{d}t},\ e_2^{t-\mathrm{d}t}\right]^{-1} \tag{A.2}$$

For rendering purposes, we need tensors linearly interpolated along triangles so as to avoid discontinuities at edges during deformation. Thus, tensors at vertices are first computed from neighboring triangle tensors, either by a naive averaging approach or by a more elaborated scheme [25]. Linear interpolation along triangles is then achieved by graphics hardware.

Appendix B. Computing the residual warping

The algorithm below details the computation at run time of the residual warping $W^*_{r(t)}(\mathbf{u}) = \mathbf{u} + \vec{\mathbf{w}}^*_{r(t)}(\mathbf{u})$. It is based on the equations of Sections 5.1 and 6.

A key observation here is that, when the level changes, the residual warping integration along $\vec{\mathbf{v}}$ restarts from the non-warped parameter $\mathbf{u}$, not from $W^*_{h(t)}(\mathbf{u})$. Indeed, to write Equation (12) we need $W^*_{r(t)}$ and $W^*_{h(t)}$ to commute:

$$W^*_{t^*(t)} = W^*_{r(t)} \circ W^*_{h(t)} = W^*_{h(t)} \circ W^*_{r(t)}. \tag{B.1}$$

This is true because the integration step does not depend on geometry or any time dependent parameter. This would be incorrect for W_t in general.

Data: flow $\vec{\mathbf{v}}$ and geometric deformation T_t for all pixels u
Result: $h(t)$ and $\vec{\mathbf{w}}^*_{r(t)}$ for all pixels u

$\vec{\mathbf{w}}[u] = 0$ ▷ stores $\vec{\mathbf{w}}_t(u)$
$\vec{\mathbf{w}}^*[u] = 0$ ▷ stores $\vec{\mathbf{w}}^*_{r(t)}(u)$
$t^*[u] = 0$ ▷ stores $t^*(t)$
$h[u] = 0$ ▷ stores $h(t)$
for any step $t \to t + \mathrm{d}t$ **do**
  $\mathrm{d}\mu[u]$ = function of $\vec{\mathbf{v}}(u + \vec{\mathbf{w}}[u])$ and $T_t(u)$
  $t^*[u] \mathrel{+}= \mathrm{d}\mu[u]$
  $\vec{\mathbf{w}}[u] \mathrel{+}= \mathrm{d}\mu[u]\vec{\mathbf{v}}(u + \vec{\mathbf{w}}[u])$
  $\vec{\mathbf{w}}^*[u] \mathrel{+}= \mathrm{d}\mu[u]\vec{\mathbf{v}}(u + \vec{\mathbf{w}}^*[u])$
  if $t^*[u] \geq h[u] + 1$ **then**
    $h[u] \mathrel{+}= 1$
    $\vec{\mathbf{w}}^*[u] = (t^*[u] - h[u])\,\vec{\mathbf{v}}[u]$
  else if $t^*[u] \leq h[u] - 1$ **then**
    $h[u] \mathrel{-}= 1$
    $\vec{\mathbf{w}}^*[u] = (t^*[u] - h[u])\,\vec{\mathbf{v}}[u]$
end