By-Example Synthesis of Structurally Sound Patterns

Jérémie Dumas* Université de Lorraine, INRIA
An Lu* INRIA, T.U. München
Sylvain Lefebvre INRIA
Jun Wu T.U. München
Christian Dick T.U. München

Figure 1: *Our by-example pattern synthesis algorithm produces structurally sound patterns along the surface of an objects, resembling the input exemplar. The reinforcements are integrated* within *the pattern, by a joint optimization of appearance and structural properties.* ***Top:*** *The Standford bunny with a variety of synthesized patterns.* ***Bottom:*** *Example patterns. These objects are printed in ABS plastic on low-cost filament printers, using a dense support. The patterns are fully connected and survived the cleaning process thanks to their reinforced structure. Yet, the reinforcements are inconspicuous as they seamlessly blend within the appearance.*

Abstract

Several techniques exist to automatically synthesize a 2D image resembling an input exemplar texture. Most of the approaches optimize a new image so that the color neighborhoods in the output closely match those in the input, across all scales. In this paper we revisit by-example texture synthesis in the context of additive manufacturing. Our goal is to generate not only colors, but also structure along output surfaces: given an exemplar indicating 'solid' and 'empty' pixels, we generate a similar pattern along the output surface. The core challenge is to guarantee that the pattern is not only fully connected, but also structurally sound.

To achieve this goal we propose a novel formulation for on-surface by-example texture synthesis that directly works in a voxel shell around the surface. It enables efficient local updates to the pattern, letting our structural optimizer perform changes that improve the overall rigidity of the pattern. We use this technique in an iterative scheme that jointly optimizes for *appearance* and *structural soundness*. We consider fabricability constraints and a user-provided description of a force profile that the object has to resist.

Our results fully exploit the capabilities of additive manufacturing by letting users design intricate structures along surfaces. The structures are complex, yet they resemble input exemplars, resulting in a modeling tool accessible to casual users.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling;

Keywords: Texture Synthesis, Fabrication, By-example Modeling

1 Introduction

Additive manufacturing empowers designers and artists with an unprecedented ability to imagine and manufacture fine, intricate patterns. The patterns may be arranged in a delicate overall structure that flows in space and suggests the surface of a larger object. The sculptures *Crania Anatomica Filigre* by artist Joshua Harker [2011], the surface autoglyphs by Henry Segerman [2009] or the designs by company *Nervous System* [Rosenkrantz and Louis-Rosenberg 2007] are impressive and fascinating examples of this trend, also witnessed by the popularity of Voronoi carvings on the sharing platform *Thingiverse* (e.g. Chess Set - Voronoi Style `thing:172960`, Coral Candle Fixture `thing:32513`).

In this paper we consider the problem of automatically generating such patterns, from an input example. Our intent is to empower casual users and designers with a tool that quickly generates a compelling pattern that prints correctly and does not break in every-day use. Performing this task by hand is very difficult: the user has to be skilled in the use of CAD systems — which are often not targeted at such models — but also needs a good understanding of the limitations of additive manufacturing as well as notions of mechanical engineering to foresee when and how the object might break.

Using our approach, the user quickly obtains a solution that enforces these constraints, and resembles the input pattern. She is then able to focus on the most important task: exploiting our technique for designing interesting and intriguing objects.

We understand this problem as an instance of *by-example texture synthesis*, a long standing problem in Computer Graphics [Wei et al. 2009]. Despite the wide spectrum of available methods, only a few are capable of synthesizing a pattern along a surface, and these approaches either manipulate finely tessellated models and per-vertex colors, or synthesize a volume of colors, or operate through a planar parameterization of the surface. This does not fit our purpose: we seek to produce a pattern that flows along the surface, without suffering from the distortions or discontinuities of planar mappings. Existing volume approaches are not well suited as we seek a method capable of efficiently updating the synthesis result locally, so as to reinforce and strengthen the global structure.

[1] Joint first authors.

Most importantly, none of the existing techniques generate patterns with controlled structural properties. A simple failure case is to consider the connectivity of the synthesized pattern. Given an exemplar pattern describing a connected, single component pattern, the available surface synthesizers cannot offer any guarantee regarding the connectivity of the output. Our situation is more general as we not only seek for a connected pattern, but also for patterns that are *rigid enough* to withstand the manufacturing process and the necessary finishing steps, as well as every-day manipulation.

Contributions.

- A novel by-example on-surface texture synthesizer, working in a voxel shell around the surface. It synthesizes along the surface, without the need for a global parameterization, and yet does not resort on a computationally expensive volume definition of the texture synthesis problem.
- A graph abstraction of the synthesized patterns, allowing an approximate but fast computation of weak areas within the patterns, both by a stress analysis and by a geometric criterion.
- A pattern reinforcement strategy, that combines the synthesizer and a structural analysis to drive the pattern towards an object that can be fabricated.

This adds up to an algorithm for by-example synthesis of complex carved patterns along surfaces, that can be fabricated even on low-cost fused-filament printers.

Assumptions. We assume that the input surface shell can be printed correctly at the user-chosen shell thickness, *without the pattern*. Otherwise carving the pattern would only worsen its already failing structural properties. We also assume that the exemplar appearance allows for enough degrees of freedom to create connections. Our technique will produce a correct output in any case, but the appearance may be impossible to reconcile with the connectivity requirements, resulting in visible reinforcements (see Section 6.4).

2 Previous Work

The starting point of our work is by-example texture synthesis; please refer to Wei et al. [2009] for a survey. We focus the discussion on techniques for surfaces, and then discuss techniques producing geometric details. Our approach also strongly relates to fabrication, and we review some of the recent work in that field. Finally, we discuss how our problem relates to the field of topology optimization.

By-example texture synthesis on surfaces Turk [2001] and Wei and Levoy [2001] adapt the image-based approaches comparing small neighborhoods of colors to work along a mesh surface, considering densely tessellated meshes with per-vertex colors. Tong et al. [2002] proceed similarly but synthesize in every vertex a texton label capturing a BTF appearance. A drawback of these techniques are the dependency on tessellation and the resampling required when working with distorted neighborhoods along the surface. Ying et al. [2001] synthesize the texture in a parametric space: the surface is divided into planar charts into which synthesis is performed, and the result is mapped back to the surface. A similar methodology is used by Lefebvre and Hoppe [2006] in a scheme that performs parallel texture synthesis into the pixels of a texture atlas. These approaches are relatively complex as they have to cope with parametric distortions, discontinuities at chart boundaries, and sampling issues. Praun and Hoppe [2000] perform synthesis along a surface by applying many texture patches with irregular boundaries so has to give the illusion of a continuous texture. Soler et al. [2002] optimize a set of texture coordinates for the triangles with the objective of producing a visually seamless texture along the surface. To create texture maps of scanned 3D models, Lempitsky and Ivanov [2007] and Gal *et al.* [2010] align photographs with the geometry an then formulate a labeling problem to select one image for each triangle. Our synthesizer takes a similar labeling view, using randomly positioned texture planes and voxels.

All these approaches only generate colors along the surface and do not modify the underlying geometry of the model.

By-example geometry synthesis A number of approaches have been proposed to go further and generate geometric details along the surface and within the object.

Bhat and Turk [2004] synthesize geometric details using the by-analogy approach initially developed for images [Hertzmann et al. 2001]. The geometry is captured by voxels and a distance field, and therefore the synthesizer is free to carve and sculpt the object. The scheme is based on voxel neighborhoods and therefore requires the input exemplar to be a small 3D pattern with geometric details. Lagae et al. [2005] perform geometry synthesis of 3D patterns by comparing blocks of voxels in a distance field. Comparing voxel neighborhoods throughout the object is computationally expensive. Dong et al. [2008] synthesize a volume texture restricted to a surface: only the voxels surrounding the surface are actually computed. This still requires to process a significant amount of volume data through the multi-scale neighborhood dependencies, while our scheme maintains a one-voxel thickness across all scales.

All the aforementioned approaches employ a 3D version of the 2D neighborhood matching of texture synthesis. In contrast, while we do perform synthesis in a set of voxels representing the surface, our formulation is different and does not involve comparing neighborhoods of voxels.

Zhou et al. [2006] synthesize a detailed mesh around a guiding mesh surface by stitching together geometric elements cut out from an input example geometry. The elements are deformed, aligned and stitched to produce a continuous result which is grown in a parametric domain around the model. Impressive patterns are obtained, the main drawback being the need to define elements in an input mesh and the geometric distortion in high curvature regions which would make fabrication delicate. The approach is not designed to guarantee connectivity or structural soundness. Ma et al. [2014] transfer the style of a mesh to another, using the by-analogy framework to guide an automated copy-paste of geometric patches. This work addresses large scales models while we focus on synthesizing small scale patterns.

Finally, closer to our goals, Zhou et al. [2014] synthesize fabricable patterns along a curve, from an example. This approach is specifically designed to synthesize patterns with controlled topology. It is however a one-dimensional synthesizer and the approach does not consider the structural properties of the generated objects.

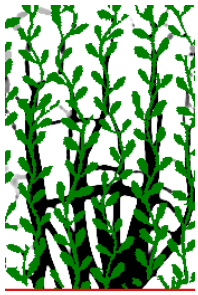
Fabrication constraints A number of recent papers address several aspects of fabrication, helping the user to produce designs that enforce specific constraints.

Umetani et al. [2013] quickly analyze the robustness of a rigid object through an analysis of the cross sectional stresses and reveal fragilities to the user. Stava et al. [2012] reinforce meshes that would otherwise be too fragile after printing. Based on a stress analysis of a tetrahedral mesh, some parts are thickened and struts are added to reinforce the object. A major difference with our work is that we do not seek to correct the synthesized patterns after the fact, but to *jointly* optimize for appearance and structural soundness: in most

cases the reinforcements seamlessly integrate within the appearance of the synthesized pattern.

Capturing all the possible forces that can be applied to a model is difficult. Zhou et al. [2013] thus propose a worst case analysis, which does not need to rely on specific forces applied to the problem. Our approach lets the user specify forces for an intended use case, but also by avoiding fragilities that could occur outside of this expected scenario. Other approaches allow the user to balance shapes [Prévost et al. 2013; Christiansen et al. 2015b], produce strong yet lightweight inner structures [Wang et al. 2013; Lu et al. 2014], or divide the object into smaller parts for easier printing [Luo et al. 2012; Hu et al. 2014].

Topology optimization One possible way to improve the structural soundness of a shape is via *topology optimization*. The Solid Isotropic Material with Penalization (SIMP) method [Sigmund and Maute 2013; Christiansen et al. 2015a] seems particularly well suited to our goal, since it considers a distribution of material in a grid and maximizes a rigidity objective under a prescribed material consumption ratio [Sigmund 2001].



However, there are challenges in using this approach for our purpose. In particular, the newly generated structures would significantly disturb the visual appearance of the original pattern: topology optimization tends to accumulate matter non-uniformly as shown in the Figure inset. In this figure, a pattern was first synthesized. The initial pattern (green) is then used as passive elements with fixed density, where a vertical force (gravity) is applied. The bottom nodes are fixed. The SIMP method is then used to distribute additional material (black) and reinforce the structure by minimizing its compliance. Matter tends to concentrate at the bottom. This is due to the accumulation of the forces: the regions below the loads have a much higher sensitivity to the overall compliance. This leads to the destruction of any details in these regions. Our approach avoids this issue by reinforcing the pattern with a small number of thin bridges between nearby structures, and uses by-example synthesis to preserve the appearance everywhere.

Finally, the performance of topology optimization currently makes it impractical for the detailed geometric structures we target (on a bunny model with 734,415 voxels, 2,814,642 degrees of freedom, a single design update requires $\geqslant 1$ hour using a multigrid solver).

3 Overview

Input Our approach starts with a user specified target surface — given as a triangulated mesh $\mathcal{M}$, a desired shell thickness and an input exemplar pattern. The exemplar pattern is an image specifying colors as well as a binary pattern map in which pixels are tagged as either solid (1) or empty (0). This is illustrated in Figure 2.

The user also specifies a radius $r_{pattern}$ which captures the scale of the features in the example pattern. We assume that this scale is larger than the minimal printable feature.

Pre-processing As a pre-processing step we voxelize the surface shell. We consider a regular grid of voxels and select only the voxels that intersect the surface. In addition, we augment the exemplar images with a distance field computed within the binary pattern map. The feature distance is used as an additional channel when comparing the values of pixels [Lefebvre and Hoppe 2006].

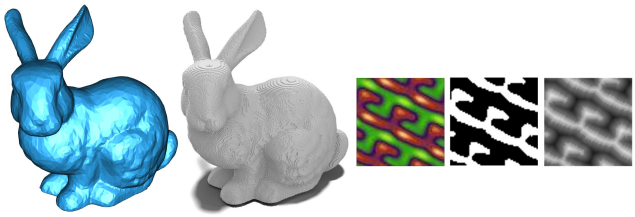


Figure 2: ***Input:*** *The original mesh, the voxelized surface, the colored pattern and its binary mask.*

Note that the voxelized surface shell has a thickness of only one voxel during the entire process. We only thicken it to the user specified thickness prior to 3D printing. Structural analysis properly takes into account the final thickness, but creating these voxels from the start would be wasteful since pattern synthesis and analysis is restricted to the surface voxels.

Algorithm Our algorithm generates a pattern along the voxelized surface shell, that is both visually similar to the example and is structurally sound. The key novelty of our approach lies in the interplay of these two objectives. Instead of reinforcing the patterns after the fact, our scheme optimizes jointly for both objectives, incorporating reinforcements within the synthesized pattern. The difference is illustrated in Figure 3.

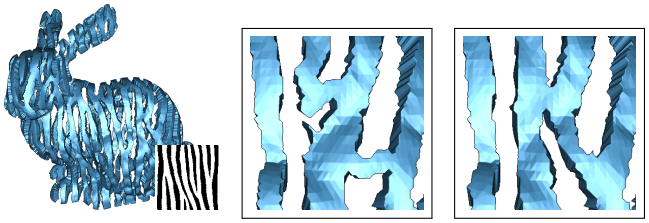


Figure 3: ***Left:*** *A result from our approach, and the corresponding exemplar (zebra).* ***Middle:*** *Closeup. Without resynthesis reinforcements are visible.* ***Right:*** *At the same location our approach seamlessly blends reinforcements within the pattern.*

Targeting a structurally sound object means that we seek for the following properties: 1) the object can be manufactured and can survive the cleaning step after the printing process and 2) the object can withstand a user-specified *force profile*, i.e. points of attachments and external forces, most typically gravity, describing in which context the object will be used.

These two objectives are slightly different in nature. Optimizing only for a specific force profile means that the object will be sound under that particular circumstance, but it might exhibit fragilities under different conditions. We therefore incorporate a secondary objective which eliminates such fragilities, without any specific knowledge of the force profile. This is described in details in Section 5.

The overall algorithm for structurally sound, by-example pattern synthesis is given in Algorithm 1. SYNTHESIZE performs a complete synthesis pass, generating an initial pattern. It is only concerned with the appearance. STRUCTURALOPTIM modifies the voxels according the structural considerations and returns the modified set of voxels and a boolean indicating whether the stopping criteria has been met (`done`). RESYNTHESIZE updates the pattern to recover its appearance, while avoiding damaging the changes made by the structural optimizer.

Algorithm 1: PATTERNSYNTHESIZER

Input: Surface mesh $\mathcal{M}$, input exemplar I
Output: Surface voxels tagged as solid or empty, so that the produced pattern resembles Iand is structurally sound.

$\mathcal{V} \leftarrow$ VOXELIZE $(\mathcal{M})$;
$\mathcal{V} \leftarrow$ SYNTHESIZE(I,$\mathcal{V}$);
while *true* **do**
  $(\mathcal{V}, done) \leftarrow$ STRUCTURALOPTIM($\mathcal{V}$);
  if *done* **then**
    return $\mathcal{V}$
  $\mathcal{V} \leftarrow$ RESYNTHESIZE(I,$\mathcal{V}$);

The synthesizer is described in Section 4 and the structural optimization in Section 5. We present our results in Section 6.

Notations We denote the input exemplar image $I : (x, y) \rightarrow (r, g, b, f)$, with f the feature distance. We denote the *state* of a pixel i by $s(i) \in \{0, 1\}$, where a value of 1 means solid and a value of 0 empty. Our goal is to synthesize a mapping from any given voxel center to a location $(x, y) \in \mathbb{R}^2$ of the exemplar image I.

We denote the list of surface voxels $\mathcal{V} = [\![1, n]\!]$. Each voxel $v \in \mathcal{V}$ encloses a (small) surface patch. We assume the small patch to be planar and going through the voxel center. The voxel center position is denoted as $\mathbf{p}(v) \in \mathbb{N}^3$, its normal — averaged over the enclosed surface — is denoted as $\mathbf{n}(v) \in \mathbb{R}^3$.

4 Surface Texture Synthesizer

Our synthesizer builds upon a new formulation of texture synthesis on 3D surfaces. We consider a set of planes in $\mathbb{R}^3$, each defining an orthogonal projection of the (tiled) exemplar everywhere in space. Any voxel center can thus lookup a color from any of these planes, by projection. Our synthesizer chooses in every voxel a unique source plane so that the combinations of projected colors along the surface gives the illusion of a continuous texture resembling the exemplar image. This idea is illustrated in Figure 4.

By controlling the set of planes, we easily guide the synthesizer towards different pattern scales or orientations. The approach is efficient as we only need to encode a choice of plane in the voxels, while the other information (coordinates, projection, colors) is implicit.

The synthesizer performs a multi-resolution synthesis into a pyramid of voxelized representations of the initial voxels $\mathcal{V}$. We denote $\mathcal{V}^r$ the resolution levels, with $\mathcal{V} = \mathcal{V}^0$ the finest level and $\mathcal{V}^L$ the coarsest level. For the sake of clarity, let us for now only consider the finest resolution level $\mathcal{V}$.

4.1 Layers Around the Surface

We consider a set of planes Ψ chosen to have normals uniformly distributed in the unit sphere. The exact location of the planes does not matter, so let us assume they surround the object as illustrated in Figure 4, left.

The planes define orthogonal projections of the texture onto the surface, and the voxels will receive their color from one of these planes. By optimizing these choices, the synthesized texture will appear along the surface. This is illustrated in Figure 4, right.

We denote by ψ a plane in Ψ of normal $\mathbf{n}(\psi)$. In addition we define a set of plane transformations Γ, each $\tau \in \Gamma$ defined by an origin

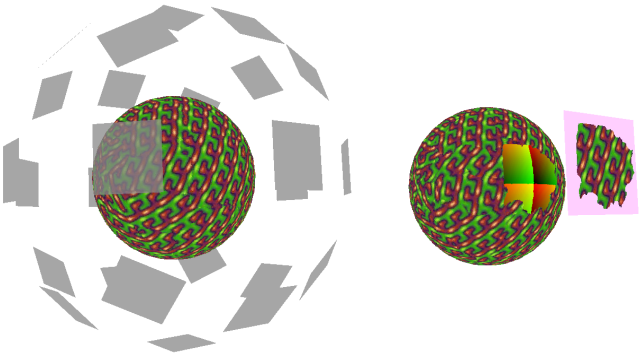


Figure 4: ***Left:*** *Planes surrounding the object, each defining an orthogonal projection of the texture onto the surface.* ***Right:*** *Each surface point selects a plane as the source for its color. The optimizer naturally grows patches. The image shows the source plane for one of the patches. The colors within the surface patch are revealing the (u, v) lookup coordinates into the exemplar image.*

point $\mathbf{o}(\tau)$ in the plane and two orthogonal vectors $\mathbf{u}(\tau), \mathbf{v}(\tau)$. Every τ maps the exemplar I to the plane in a different manner. The $\mathbf{u}(\tau), \mathbf{v}(\tau)$ vectors are not necessarily normalized so as to allow for scaling.

Each voxel $v \in \mathcal{V}$ is associated with a mapping to exemplar texture space by choosing a plane-parameterization pair $(\psi(v), \tau(v)) \in \Psi \times \Gamma$. The mapping function $M(\mathbf{x}, \psi(v), \tau(v))$ projects x on the surface of the chosen plane $\psi(v)$ via an orthogonal projection. The projected point is then mapped to image space by the planar transformation $\tau(v)$.

Our texture synthesis consists of finding a good set of choices of $(\psi(v), \tau(v))$ for voxels $v \in \mathcal{V}$ such that the resulting texture resembles the exemplar appearance.

4.2 Synthesis as an Energy Optimization

We now formalize the optimization problem to achieve texture synthesis along the surface. We start by making several observations regarding the desired properties of the result, and then give a precise formulation of the energy to optimize.

4.2.1 Desired Properties

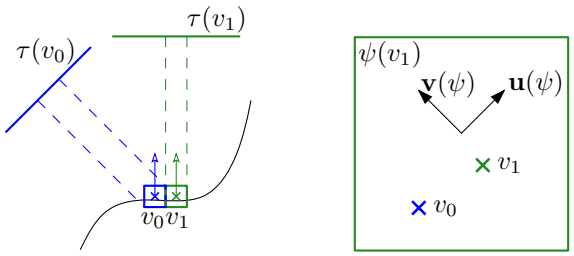
Let us consider a voxel $v \in \mathcal{V}$. It corresponds to a small surface patch, approximated locally as a plane. All the surface points are mapped to the image space by the same function $M(\mathbf{x}, \psi(v), \tau(v))$ where $(\psi(v), \tau(v))$ is the parameterization choice for v, the voxel enclosing $\mathbf{x}$.

If the normal to the plane $\mathbf{n}(\psi(v))$ and the voxel normal $\mathbf{n}(v)$ align perfectly, then the resulting texture in v is a copy of one portion of the input exemplar, uniformly scaled and rotated by $\tau(v)$. This locally reproduces the exemplar appearance. However, if the normal of the plane disagrees with the voxel normal, the texture will appear distorted along the surface enclosed within the voxel. Our energy term penalizes such cases, encouraging voxels to choose projection planes agreeing with the local surface normal.

Let us now consider that there is a negligible amount of distortion, as would be the case for a planar surface. In this case, preserving the exemplar appearance boils down to achieving inconspicuous texture transitions between the voxels, e.g. similarly to image quilting [Efros and Freeman 2001]. Let us consider $5 \times 5 \times 5$ neighbor-

hoods of voxels. We denote by $\mathcal{N}(v)$ this neighborhood for a voxel v. If the neighboring voxels perform the same choice of mapping, that is for all $w \in \mathcal{N}(v), (\psi(w), \tau(w)) = (\psi(v), \tau(v))$, then the voxels copy coherent (adjacent) texture patches, and the appearance is preserved.

If the voxel chooses different projections, color discontinuities might become visible at voxel boundaries. We measure the quality of the transition by considering the color differences between the colors that the neighborhood *expects* and the color the voxel has, and vice versa. This idea is illustrated in Figure 5.



(a) *Projection onto the planes* **(b)** *2D view of green plane*

Figure 5: *Mapping voxel centers to exemplar texture space.* ***(a)*** *Two voxels selecting different planes.* ***(b)*** *Projection of both voxel centers into the plane selected by v_1. The color of both projected points is used to determine whether the transition is visible. A similar operation is performed into the plane selected by v_0.*

4.2.2 Synthesis Energy

Based on this analysis, we derive the energy function that measures the quality of a choice of mappings $\mathcal{C} : \mathcal{V} \rightarrow \Psi \times \Gamma$. The global energy is defined as the sum of two terms, one for color transitions and one for normal alignments:

$$E(\mathcal{V}, \mathcal{C}) = \sum_{v \in \mathcal{V}} (E_{transition}(v, \mathcal{V}, \mathcal{C}) + E_{distortion}(v, \mathcal{V}, \mathcal{C}))$$

The transition error is measured as:

$$E_{transition}(v, \mathcal{V}, \mathcal{C}) = \sum_{w \in \mathcal{N}(v)} \|I(M(v, \psi(v), \tau(v))) - I(M(v, \psi(w), \tau(w)))\|^2 + \|I(M(w, \psi(v), \tau(v))) - I(M(w, \psi(w), \tau(w)))\|^2$$

The distortion error is:

$$E_{distortion}(v, \mathcal{V}, \mathcal{C}) = 1 - (\mathbf{n}(v) \cdot \mathbf{n}(\psi(v)))$$

4.3 Optimization Scheme

We optimize for $\widetilde{\mathcal{C}} = \arg\min\{E(\mathcal{V}, \mathcal{C})\}$. To make the problem tractable we select a finite set of planes Ψ and plane transformations Γ. Therefore, each pair (ψ, τ) can be seen as a label. In the following we consider the pairs to be integers indexing planes and plane transformations.

We now describe an optimization scheme quickly selecting labels. It is inspired by the upsample-jitter-correction scheme of Lefebvre and Hoppe [2005]. Note however that our formulation could also be solved by other optimizers, such as alpha-expansion on the labels (ψ, τ) in a global optimization approach [Kwatra et al. 2003; Kwatra et al. 2005; Lempitsky and Ivanov 2007]. However, for stochastic textures the greedy local improvement strategy gives good results while enabling a fast parallel update scheme [Wei et al. 2009].

We now consider the multi-resolution pyramid of voxels, together with a Gaussian stack for the input exemplar I. The algorithm is given in Algorithm 2. UPSAMPLE propagates the choices from one level to the next, simply copying the choice of the parent in the child voxels. JITTER introduces randomness: a fraction of the voxels are forced to have random selections of labels. This percentage is exposed to the user to control how regular or random the pattern should be. OPTIMIZE performs the actual labeling, and is described in Algorithm 3.

In every optimization step, we construct a set of candidate choices for each voxel. We first include choices from the neighbors, a process inspired by the coherent-candidate mechanism [Ashikhmin 2001; Tong et al. 2002]. These choices tend to grow coherent patches on the surface. If, however, the neighboring voxels have disagreeing normals the distortion energy will quickly increase.

To allow the optimizer to discover better transitions, both between patches and in curved areas, we add R random candidates. The space of possible pairs is however extremely large, and we therefore bias the random sampling towards the most likely candidates. We pick a random direction within a cone around the voxel normal, and select the plane in Ψ having the normal that best aligns with this direction. The parameterization τ is either chosen randomly (uniformly), or

Algorithm 2: SYNTHESIZETEXTURE

Input: Pyramid of surface voxels $\mathcal{V}^0, ...\mathcal{V}^L$
Output: Choices for each resolution level $\mathcal{C}^0, ..., \mathcal{C}^L$

$\mathcal{C}^L \leftarrow \{(\psi(v), \tau(v)) = (0, 0) | v \in \mathcal{V}_0\}$;
for *l from* $L-1$ *to* 0 **do**
  $\mathcal{C}^l$ = UPSAMPLE($\mathcal{C}^{l+1}$);
  $\mathcal{C}^l$ = JITTER ($\mathcal{C}^l$);
  $\mathcal{C}^l$ = OPTIMIZE($\mathcal{C}^l$);
return $\mathcal{C}^0$

Algorithm 3: OPTIMIZE

Input: Surface voxels $\mathcal{V}$, set of choices $\mathcal{C}$, planes Ψ and plane transformations τ.
Output: Optimized set of choices $\mathcal{C}$

for $v \in \mathcal{V}$ **do**
  // Coherent candidates
  $K \leftarrow \{(\psi(w), \tau(w)) | w \in \mathcal{N}(v)\}$;
  // Random candidates
  for $i \leftarrow 1$ **to** R **do**
    (r_1, r_2) = sample a random pair in $\Psi \times \Gamma$;
    $K \leftarrow K \cup \{(r_1, r_2)\}$;
  $e_{min} \leftarrow E(v, \mathcal{V}, \mathcal{C})$; // Current energy
  $t_{best} \leftarrow (\psi(v), \tau(v))$; // Initialize best choice
  foreach $k \in K$ **do**
    $\mathcal{C}' = \mathcal{C}$ with $\psi(v) \leftarrow \psi(k), \tau(v) \leftarrow \tau(k)$;
    $e_{min} \leftarrow \min(e_{min}, E(v, \mathcal{V}, \mathcal{C}'))$;
    $t_{best} \leftarrow$ update best choice;
  $\mathcal{C}(v) = t_{best}$;
return $\mathcal{C}$

biased towards a predefined orientation to let the user control how the pattern flows along the surface.

The candidate with lowest energy becomes the choice for the current voxel. Voxels are updated in parallel, using a sub-pass update mechanism [Lefebvre and Hoppe 2005].

4.4 Pattern Synthesis

The synthesizer is used to directly produce colors into the voxelized surface shell. By considering the solid attribute in the exemplar image, we obtain a carved pattern. Of course, this pattern generally cannot be fabricated — we will discuss these aspects in Section 5.

As can be seen in Algorithm 1 there are two different calls to the synthesizer: SYNTHESIZE and RESYNTHESIZE. The first performs the initial synthesis, and the second adapts the patterns to the changes made by the structural optimizer.

Initial synthesis Initial synthesis is performed using our synthesis algorithm without any change, with optional user controls such as a pattern orientation.

Re-synthesis After structural optimization, a new set of voxels has been marked as solid to reinforce the pattern. These changes might disagree with the pattern itself. Let us consider the case of an anisotropic pattern. Since it tends to produce elongated contiguous features, the structural optimizer will very likely introduce connections between the features, orthogonal to their main orientation.

It is therefore necessary to locally recover the appearance. However, to have any hope for the process to converge we need to guarantee that the changes will not be removed entirely. We therefore perform a local update of the pattern, that is constrained in two ways. First, the voxels which have been forced as solid for structural concerns may only select candidate labels marking them as solid through the projection onto the exemplar. Second, only the surrounding voxels will be locally resynthesized. These are free to adapt to their surroundings. We employ a local constraint scheme which propagates through a few resolution levels. All parent voxels having one child voxel tagged as solid are also tagged as solid. Synthesis then resumes from the coarser resolution level to the finest, restricting the set of updated voxels to a local region located below the coarsest parent tagged as solid. The coarser resolution level is selected by going back $\log_2(r_{pattern})$ levels, where $r_{pattern}$ is expressed in voxels of finest resolution. We illustrate the process in Figure 6, where a red color shows the solid voxels introduced by structural optimization and the green color shows voxels which are locally re-synthesized to adapt to the constraint.

5 Structural Optimization

After the first synthesis step the voxel states (solid/empty) obtained from the projected exemplar do not, in general, define a structurally sound pattern, and in most cases not even a single connected component. The goal of our structural optimization is to correct these issues, working jointly with the optimizer towards the final pattern.

We give in Algorithm 4 the pseudo code for the structural optimization step. GENERATESURFACEGRAPH and GENERATEABSTRACTGRAPH are described in Section 5.1, and REINFORCEMENTBRIDGES is described in Section 5.2.

We assume that the full surface shell, without the carved pattern, is a strong enough object. Under this assumption, the worst case scenario of the structural optimization would be to fill all empty voxels, completely removing the synthesized pattern.

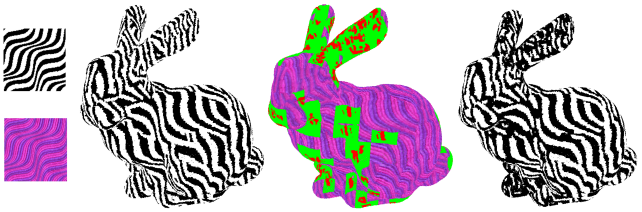


Figure 6: ***From left to right:*** *1) Exemplar pattern at the top (white is solid), color version at the bottom. 2) Initial synthesis result. 3) The constraint map of a single iteration shown on the colored version of the exemplar. The red areas can only select candidates with a solid tag, while green areas are free to adapt. The rest of the surface is not allowed to change. 4) Final pattern after all iterations are completed. Note that the ears are now connected to the body, and that the neck has been strengthened.*

Algorithm 4: STRUCTURALOPTIM

Input: Surface voxels $\mathcal{V}$ tagged as solid or empty
Output: Surface voxels $\mathcal{V}$ tagged as solid or empty with improved structural properties

1 $\mathcal{G}_{surface} \leftarrow$ GENERATESURFACEGRAPH($\mathcal{V}$);
2 $\mathcal{G}_{abstract} \leftarrow$ GENERATEABSTRACTGRAPH($\mathcal{V}, \mathcal{G}_{surface}$);
3 $\mathcal{V} \leftarrow$ REINFORCEMENTBRIDGES($\mathcal{G}_{abstract}$,$\mathcal{V}$);
4 **return** $\mathcal{V}$;

5.1 Surface Graph and Abstract Graph

We now need a data-structure on which to perform analysis. The number of surface voxels is generally high — in the order of 10^6 — and it is computationally prohibitive to directly manipulate this data, especially for the computation of structural properties. We therefore propose to define an abstraction of the synthesized pattern.

We define the *surface voxels graph* as the weighted undirected graph $G_{surface} = (\mathcal{V}, E)$, where $(u, v) \in E$ is an edge if and only if voxels u and v are contiguous, that is they touch by a corner: $\|\mathbf{p}(u) - \mathbf{p}(v)\|_2^2 \leqslant 3$. The edges are weighted by the euclidean distance between the voxels they connect. A closeup on a surface graph is shown in Figure 7 (left). Note that this graph is not impacted by the choice of solid/empty voxels and does not depend on the synthesized pattern. It is thus generated only once. For the sake of clarity we assume next that $G_{surface}$ is fully connected: the input surface mesh $\mathcal{M}$ is a single object. We otherwise treat each component separately.

The first part of our analysis process is to abstract the surface graph into a graph of lower complexity — between 500 and 1000 vertices — while still retaining the same global connectivity as the synthesized pattern. Figure 7 (right) shows an example of the abstract graph.

We denote by $d_G(u, v)$ the graph geodesic distance between u and v in a graph G. The *abstract surface graph* is denoted $G_{abstract} = (\mathcal{S}, \mathcal{F})$, where $\mathcal{S} \subset \mathcal{V}$. Each edge $f \in \mathcal{F}$ is tagged with a state $\rho(f) \in \{0, 1\}$ indicated whether it is considered empty or solid.

The subset of voxels $\mathcal{S}$ in $G_{abstract}$ is selected by down-sampling from the set of *solid* voxels $\mathcal{V}$ in $G_{surface}$. This is done following a Poisson disk sampling strategy, with a binary search on the radius to reach a target number of voxels (1000 in our examples) [Bowers et al. 2010]. We then connect these voxels by edges so as to reproduce the connectivity of the synthesized pattern. This is done by algorithm CONNECTSAMPLES (Algorithm 5).

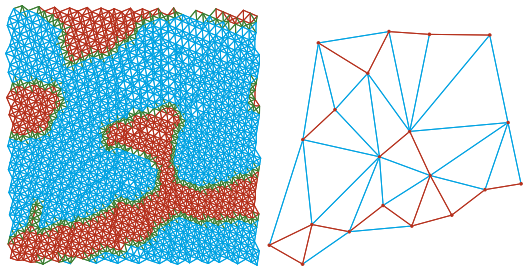


Figure 7: *The initial surface graph $G_{surface}$ (left) is used to produce the abstract pattern graph $G_{abstract}$ (right), having a greatly reduced number of vertices but still capturing the connectivity of the pattern. Red edges are solid and blue edges are empty.*

Algorithm 5: CONNECTSAMPLES

Input: The surface and solid voxels graphs and subsets $\mathcal{S} \subseteq \mathcal{V}$
Output: A graph $G_{abstract} = (\mathcal{S}, \mathcal{F})$, which is a subset of $G_{surface}$. A function $\rho : \mathcal{F} \to \{0, 1\}$ for the edge type.

1 Compute shortest path forest $\mathfrak{F}$ from $\mathcal{S}$ in G_{solid} (via DIJKSTRA);
2 For each pair of tree $T(u), T(v) \in \mathfrak{F}$ that have a vertex incident to the same edge $e \in E(G_{solid})$, connect their roots ($\mathcal{F} \leftarrow (u, v) \cup \mathcal{F}$), and set $\rho(u, v) = 1$;
3 Compute shortest path forest $\mathfrak{F}'$ extending $\mathfrak{F}$ in $G_{surface}$;
4 For each pair of tree $T(u), T(v) \in \mathfrak{F}'$ s.t. $(u, v) \notin \mathcal{F}$ and both trees have a vertex incident to the same edge $e \in E(G_{surface})$, connect their roots ($\mathcal{F} \leftarrow (u, v) \cup \mathcal{F}$), and set $\rho(u, v) = 0$;
5 **return** $(G_{abstract} = (\mathcal{S}, \mathcal{F}), \rho)$

CONNECTSAMPLES proceeds as follows: starting from sources in $\mathcal{S}$, it initially grows regions in G_{solid} — the graph $G_{surface}$ restricted to solid voxels. Whenever two regions are connected by the growth, the two source voxels are connected by a new edge in $G_{abstract}$. The newly added edge $f \in \mathcal{F}$ is marked as solid ($\rho(f) = 1$). In a second pass, the same regions are grown in the whole graph $G_{surface}$. For each adjacent regions that were not previously connected a new edge is added in $\mathcal{F}$. It is marked as empty since the connecting path in $G_{surface}$ goes through empty voxels. The paths connecting source voxels through $G_{surface}$ are recorded. They are used later to produce reinforcements.

5.2 Reinforcement Bridges

We now proceed to reinforcing the pattern. This step performs a number of local changes, adding new voxels. The algorithm iteratively selects edges of $G_{abstract}$ marked as empty and changes them to become solid. The corresponding voxels in $\mathcal{V}$ are in turn switched to become solid. This information is fed back to the synthesizer.

We rely on two complementary approaches to compute the scores in the edge selection process. The first and main score is based on an analysis of stresses under a certain force profile in a simplified beam geometry capturing the pattern (Section 5.2.1). The second is a purely geometric criterion based on pattern geodesic distance in $G_{abstract}$ (Section 5.2.2).

The rationale for using two criteria is that the force profile only predicts a single scenario. Therefore, under a different set of circumstances some fragile configurations might exist. The second score acts as a worst-case criterion. Note that the user is in charge of deciding how safe he wants the print to be. Strengthening the second criterion will ultimately make the force profile have less impact. We generally allow only a few changes from the worst-case criterion, so as to obtain interesting effects from the force profile without extreme fragilities under other conditions.

A primary goal of the edge selection process is to form a single connected component in $G_{abstract}$, considering solid edges only. To this end, edges are added iteratively, choosing the next empty edge of highest score which connects two different components, using the force profile criterion. A number of additional edges are then added using the force profile criterion again, until the maximum stress of all edges is below a material-dependent threshold. We currently manually fix this threshold based on ABS/PLA plastic elastic properties. After adding edges with the force profile, we consider the second, worst-case criterion. We again mark edges as solid until no edge considered as weak remains.

The algorithm finally modifies the set of voxels. Each edge in $G_{abstract}$ corresponds to a voxel path — the one that connected the regions as described in Section 5.1. Adding only these voxels would not be sufficient, as it would produce a very thin bridge between two parts of the pattern. Instead, we dilate these paths and add all voxels which are within distance $r_{pattern}$ of the voxels along the path. Let us re-emphasize that adding edges in $G_{abstract}$ does not necessarily produce a straight line in $\mathcal{V}$ as it follows the shortest path between the solid regions, across empty voxels.

5.2.1 Score Based on Force Profile

We now exploit the abstract graph to build a simplified mechanical model of the pattern. Intuitively, each edge will become a beam whose stiffness is determined by whether it is solid or empty.

We cannot directly use truss or beam elements from the finite element method (FEM) literature, as these are 1D elements that feature a free rotation at their endpoints. This pivot joint behavior does not properly capture the printed pattern. Instead, given the abstract surface graph, we generate a 3D geometry based on hexahedral and wedge elements, which we then simulate with the FEM to obtain a structural analysis of the pattern.

To achieve a proper geometric construction we make the assumption that the surface around each voxel is locally planar — which is correct given a smooth mesh as input. This makes it possible to represent every graph voxel by wedge elements, considering the 1-ring of neighbors projected into the local plane, as illustrated in Figure 8. The elements are extruded along the voxel normal, by an amount that corresponds to the desired shell thickness. Each neighbor connects to the current graph voxel by an hexahedral element incident to the face of a wedge.

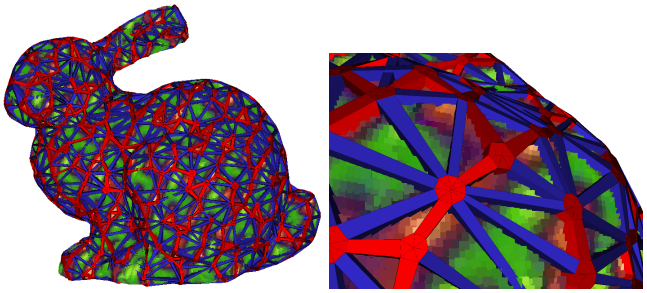


Figure 8: *Each vertex of the abstract graph becomes a set of wedge elements connecting to its neighbors through hexahedral beams. Solid elements are shown in red, while soft elements are shown in blue. Each beam corresponds to an edge in the graph, letting us derive a stress tensor for each edge through finite element analysis.*

Note that we do not take into account flips or self-intersections that may occur if the curvature is too high. None of the models we tested created such cases.

Finite element analysis The geometry constructed from the abstract graph is composed of wedges and hexahedral elements, onto which we apply the finite element method. We simulate the displacement of each node (vertex) of the beam-wedge geometry. The shape functions for interpolation within the elements are given in supplemental material. We simulate an elastic material having the properties of the printing material, typically ABS or PLA plastic.

Each element is associated with a stiffness $\rho(e)$. For wedges it is always 1, and for hexahedral elements — which correspond to graph edges — it is set to either 1 or $\rho_{\min}$ depending on whether the corresponding edge in $G_{abstract}$ is marked as, respectively, *solid* or *empty*. The stiffness matrix of the element is then derived as $\widetilde{\mathbf{K}}_e = \rho(e)\mathbf{K}_e$.

Once the global stiffness matrix is assembled we need to decide on a set of external forces and locked nodes. Let us briefly assume these are given. We can then solve the static equilibrium equation $\mathbf{Ku} = \mathbf{f}$, which we compute using the Eigen library [Guennebaud et al. 2010] and the sparse solver CHOLMOD [Chen et al. 2008]. This computes a small displacement of the element nodes, which in turn allows us to estimate a stress tensor in the beam of each edge: $\sigma = \{\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz}, \tau_{xz}\}$, given by $\sigma_e = \mathbf{B}(\xi, \eta, \zeta)\mathbf{Eu}_e$. For more details on computing stresses with the FEM, the reader is referred to [Cook et al. 2007]. We use the sum of the norm of the principal stresses as the score for the edges.

Solving for equilibrium is relatively expensive. We therefore insert multiple edges at once. To avoid accumulation in areas of high stress, we forbid edges close to an already inserted edge, in the manner of the dart-throwing process. The radius of cancellation is twice the length of the already inserted edge.

External forces and locked nodes (boundary conditions) In the standard setting we only consider gravitational forces applied on the system. For each vertex $i \in G_{abstract}$, we apply a constant pressure in the direction of the gravity at position $\mathbf{p}(i)$, with an arbitrary but constant value. We fix all nodes that are within the 10% first layers in Z. This corresponds both to the fabrication process and the case where the model is standing upright and the points on the ground are fixed. Note that our approach makes no assumption about the forces and locked nodes and it is for instance possible to consider other scenarios such as pinch grips [Stava et al. 2012].

5.2.2 Score Based on Geometric Criterion

The geometric score is used to detect poor configurations that lead to fragilities under conditions diverging from the specified force profile, without having to resort on a mechanical simulation.

We observe that the worst fragilities consists in elongated structures that are disconnected from their surroundings. We again exploit $G_{abstract}$ to detect and suppress such configurations.

Let us consider a voxel in $G_{abstract}$ belonging to an elongated structure. This voxel is connected to a few voxels of the same structure by solid edges, and to neighboring structures by empty edges. Since at this stage the structure is fully connected, there exists a geodesic path *within the pattern* between the two voxels at each extremity of an empty edge. This is illustrated in Figure 9.

We consider the length of the geodesic path in G_{solid} as the score for an edge. We iteratively add all edges with a score higher than a user-defined threshold (typically 1.5 times the extent of the volume in our examples). The score of all edges is updated after each addition.

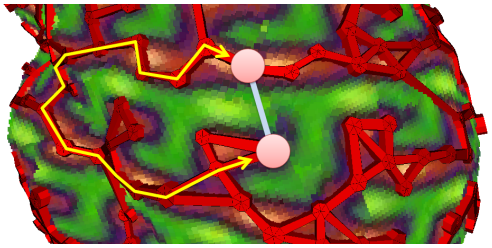


Figure 9: *Geodesic distance criterion. The view is a closeup of a pattern with the solid edge of the abstract graph overlaid. The outlined blue edge is an empty edge of the abstract graph being considered for reinforcement. Its extremities are connected by a shortest path through the solid edges, outlined by the yellow broken line. The score of the edge is its geodesic distance in the abstract graph $G_{abstract}$.*

6 Results

6.1 Texture synthesis

We first present results from our on-surface texture synthesizer. Figure 10 shows results, and Table 1 describes performance numbers. In all results, we use the following parameters: synthesis at 256^3 resolution (8 levels) for all but the skull result in Figure 14 and Figure 15 which uses 512^3 (9 levels). We use planes evenly distributed on the unit sphere in all 26 directions, with 16×16 translations and 64 rotations per direction. We use $5 \times 5 \times 5$ neighborhoods and 128 candidates in total, starting with coherent candidates given by defined neighbors, and filling the rest with random candidates. All shown results orient the pattern with a naive vector field around the 'up' direction. This could however be controlled by the user. We support non-tilable exemplars by penalizing pixels near boundaries [Lefebvre and Hoppe 2005].

In terms of quality it is roughly equivalent to existing on-surface synthesizers, but we have not added all the improvements from the state of the art (e.g. appearance space transform [Lefebvre and Hoppe 2006], interpolation of overlapping neighborhoods) which could further improve quality on color textures.

Our scheme is however conceptually simpler than the existing on-surface synthesis techniques (see [Wei et al. 2009], Section 4, for a survey) and fits our needs perfectly: the output is a thin shell of voxels and it enables fast local updates thus tightly integrating with our geometry modeling pipeline.

One drawback of our synthesizer is that it works under the assumption that the surface is smooth. Therefore, we can expect quality to degrade across sharp edges. As can be seen Figure 11 synthesis quality remains reasonable even in challenging cases such as the skull front where there is a concentration of sharp edges.

6.2 Pattern Synthesis and Fabrication

6.2.1 Preparing Synthesized Patterns for 3D Printing

Enlarging small features After the main loop of the program exits, the pattern we obtain is globally connected and structurally sound in regards of the abstract graph. However the abstract graph only captures the global connectivity of the pattern, and ignores its

Figure 10: *Texture synthesis results.*

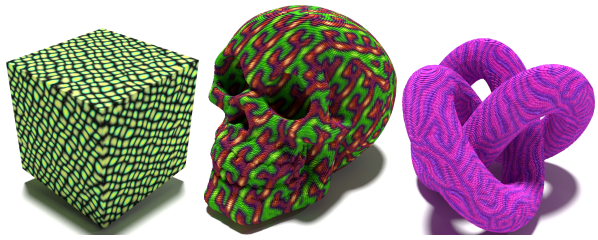


Figure 11: *Texture synthesis results on objects with sharp edges, high curvature areas, and complex topology. Models: Skull (thing:168602), Knot (thing:5506).*

local thickness. We therefore further improve the pattern by recovering a consistent feature size in all places that are too thin.

We would like to enlarge the pattern directly *on* the surface, and not *out* of it. We propose to compute a geodesic skeleton of the solid surface pattern, and perform a dilation of the skeleton by the minimal printable feature radius, restricted to the surface shell. The final result is the union of the dilated skeleton and the original pattern.

Our approach bears similarities to the work of Liu et al. [2010], whereby the geodesic curve skeleton of the surface is computed. However, our algorithm is more relaxed because it does not preserve the exact topology of the surface. This is desired since the input pattern may contain small loops which appear through synthesis. We exclude these from the skeleton.

Our technique starts by performing a watershed transform of the graph to detect large regions and connect local maxima located within these regions [Haumont et al. 2003]. We next prune the leaves and small loops from the resulting graph. Figure 12 illustrates the cleaning process.

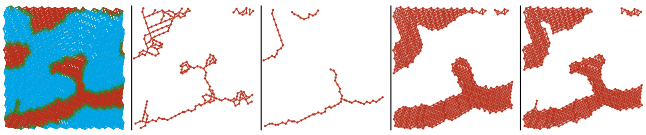


Figure 12: ***From left to right:*** *Initial pattern, noisy skeleton from the watershed transform, skeleton after pruning, after dilation, and after union with initial pattern. Small features have been enlarged.*

Final mesh The contiguous solid voxels form a thin pattern along the surface shell. We thicken this shell as a post-process, by a user-specified amount. The resulting set of voxels is then meshed by extracting its orthogonal polygon which is then remeshed to obtain a smoother model.

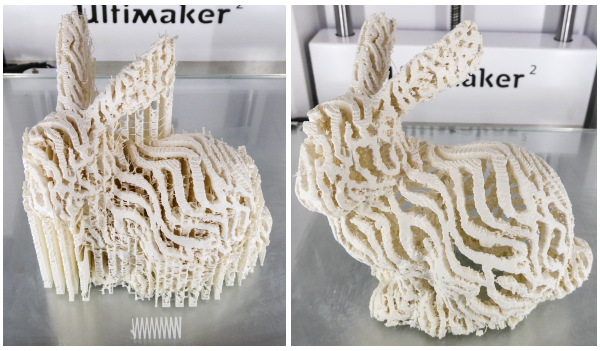


Figure 13: *Result before and after cleanup of the support structure.*

Support structures We produce our models both on filament printers (*Replicator 1, Ultimaker 2*) and powder based ink jet printers (*ZCorp 450*).

Our patterns are of course challenging prints on filament printers and they require support structures. We use simple supports from the slicing software, but more elaborate approaches could be used to facilitate cleanup [Dumas et al. 2014; Vanek et al. 2014; Schmidt and Singh 2010]. Figure 13 shows an object on the print bed, just after printing and the cleaned up version. Our structurally sound patterns survive the cleaning process even in such extreme cases.

Powder based printers, such as the ZCorp printers present different challenges. While support is not necessary the print is extremely fragile when removed from the powder bed — it is later strengthened by dipping it into cyanoacrylate. Figure 14 shows a print created on this technology.

Printing on SLS machines (laser on polyamide) would allow us to produce even thinner results without support, and would directly produce stronger parts.

6.2.2 Prints

Figure 1, Figure 14 and Figure 15 present 3D printed results. Note how the patterns remain easily identifiable along the surfaces. It is worth noting that isotropic patterns are generally easier to handle as their initial synthesis is often well connected, with the exception of highly curved areas, e.g. the ears of the Standford bunny model. Anisotropic patterns are more challenging as the appearance tends to conflict with the strengthening of the model. Nevertheless our technique produces natural results on different patterns.

Computation times and other statistics are summarized in Table 1. Table 2 gives performance breakouts between texture synthesis and structural analysis for the bunny model on various exemplars.

Model/Texture	Grid	# Voxels	# Iter	t_{init}	t_{total}
Bunny/Keyboard	256	67651	4	1.11	14.6
Bunny/Bluebrown	256	84292	10	1.11	34.8
Bunny/Greencells	256	80917	3	1.11	11.4
Bunny/Hooks	256	55356	11	1.11	40.0
Bunny/Waves	256	107038	10	1.11	52.4
Bunny/Animalskin	256	89110	24	1.30	76.2
Kitten/Hooks	256	38626	8	0.9	24.3
Skull/Hooks	512	241652	4	5.9	40.3

Table 1: *Computation time for the different models shown in the paper, showing the extent of the voxelization used, number of voxels in the final geometry, number of iterations, timings for the first synthesis pass, and timings for the whole process (in s).*

Texture	Synthesis	Structural Analysis	Total
Keyboard	8.6	4.6	13.2
Bluebrown	23.8	12.1	35.9
Greencells	8.3	4.9	13.2
Hooks	28.4	10.5	38.9
Waves	37.3	13.9	51.2
Animalskin	45.5	26.9	72.4

Table 2: *Computation times in seconds on the bunny model for the synthesis, structural analysis, and both.*

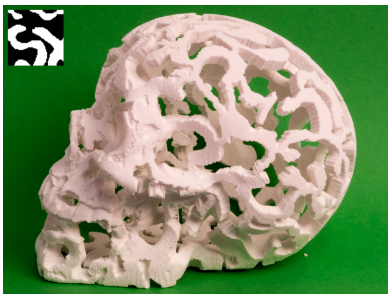


Figure 14: *Result printed on a ZCorp 450. On this type of machines the printed objects are very fragile and have to be extracted from the powder bed before dipping into cyanoacrylate. Our thin patterns nevertheless print successfully.*

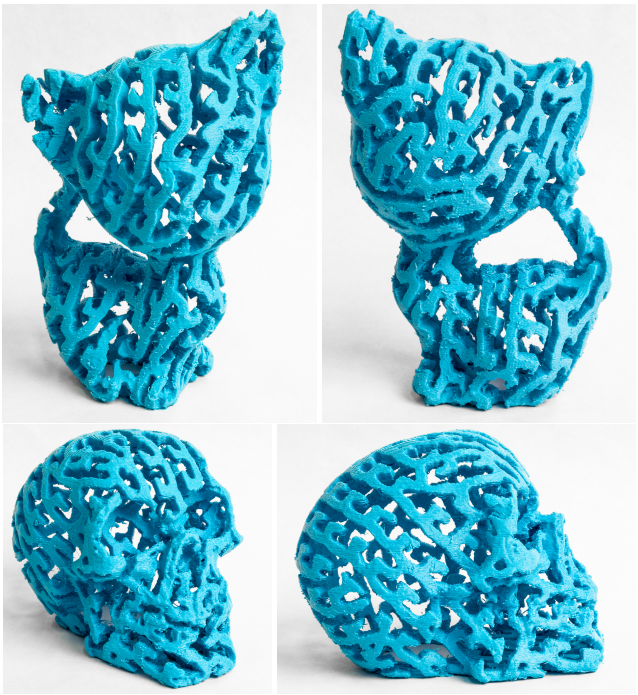


Figure 15: *Kitten and skull model carved with the hooks pattern.*
Models: Kitten (`thing:12694`), Skull (`thing:168602`).

6.3 Validation

We validate our approach by simulating our output pattern with a full-scale high resolution finite element analysis. Each voxel of the thickened pattern becomes a cubic (hexaheadral) element. We use the same boundary conditions as for our optimizer: the object is fixed to the ground and subject to gravity. The material parameters are those of ABS plastic.

Scheme	Connected	Forces only	Forces + Geodesic
Stress	Expected force profile: weight along $\vec{\mathbf{g}} = (0, 0, -1)g$		
Average	16388.5	5680.4	5418.16
Median	3961.5	3686.61	3444.95
99th%	153947	31720.7	**30486.6**
Stress	Unexpected force profile: weight along $\vec{\mathbf{g}} = (1, 0, 0)g$		
Average	87039.1	23509.8	20111.5
Median	20770.5	10374.6	9762
99th%	693773.5	185775	**145854**

Table 3: *Stresss for a model 100mm long, using a grid of extent 256. The score shown is $|\sigma_1| + |\sigma_2| + |\sigma_3|$, in Pascal. Note that with the unexpected force profile, the 99th percentile is further decreased by the last optimisation scheme.*

It is worth noting that the FEA solver requires two to five minutes to solve for the equilibrium equation. Our system does several iterations of structural optimization, each solving several times the equilibrium equation (see Section 5.2.1 and Table 1). It would thus be impractical to rely on the full simulation of the pattern directly.

The full FEA solution lets us validate our approach of using the abstract graph and simplified beam geometry. We first simulate the fully connected structure, before any additional edges are added. This is the first structure that can be simulated as disconnected structures lead to an under-determined system in FEA simulation. Second, we simulate the structure reinforced with only the force profile criterion (Section 5.2.1). Finally, we simulate the structure reinforced with our complete approach, that is force profile and geodesic criteria (Section 5.2.2). We repeat the FE analysis but this time change the force profile — the weight is applied in a direction orthogonal to the gravity used for the synthesis. This puts the pattern in an unexpected situation, for which it was not designed.

Table 3 and Figure 16 summarizes the results. As can be seen, in the expected scenario (top rows) the force profile criterion strongly reduces stresses in the structure. The geodesic criterion only marginally improves this result. In the unexpected scenario however, the geodesic criterion further improves the result as its reinforcements compensate for the now incorrect force profile the pattern was optimized for. Note that the number across both rows cannot be compared directly as the boundary conditions differ. Only numbers within a same row are comparable.

We show in Figure 17 the effect of using different force profiles on the bridges added by the structural optimizer.

6.4 Limitations

Our technique works well as long as the scale of the pattern is relatively small compared to the object. This is in general the intended use, but combined to the print size limitation it would sometimes be desirable to generate coarser pattern. High curvature areas also pose multiple difficulties: texture synthesis becomes more challenging, the local planarity assumption for building the finite elements might be violated (Section 5.2.1), and thickening to obtain the final mesh may lead to fold-overs. Thus, final quality can degrade on surfaces with highly curved features (e.g. front of the skull in Figure 15).

Not all patterns can be used to define meaningful reinforcements, e.g. if the features of the input pattern are too thin to be printed at the selected scale. Additionally, when the input pattern is completely disconnected the appearance severely conflicts with the structural

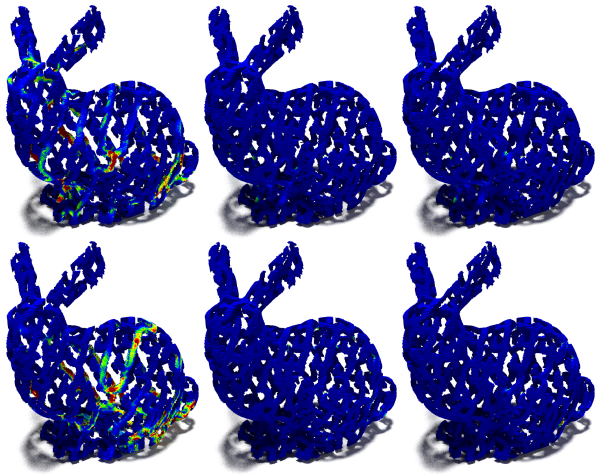


Figure 16: ***Top:*** *Color coded stresses for the first row of Table 3. The scale is the same for all three images.* ***Bottom:*** *Color coded stresses for the second row of Table 3. The scale is the same for all three images.*

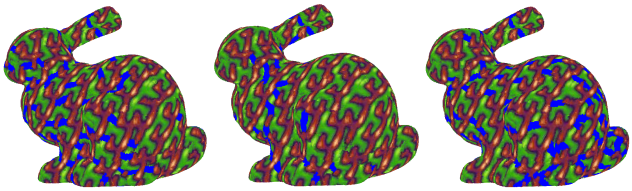


Figure 17: ***From left to right:*** *Different force profiles, with gravity to the left, to the bottom, and to the right of the bunny. Note how the reinforcements suggested by the structural optimizer adapt.*

Figure 18: *This pattern is fully disconnected and therefore the appearance cannot be reconciled with the connectivity requirements of a sound structure. The produced object nevertheless prints correctly, and does capture some of the original features.*

objectives. This still produces correct models, but the reinforcements are less inconspicuous as they do not blend within the pattern. Such a failure case is shown in Figure 18.

7 Conclusion

We have introduced the first method to synthesize patterns along a curved surface from an example, while ensuring that the pattern is printable and withstands a user specified force profile. We believe our work opens interesting questions regarding the joint optimization of shape structural soundness and shape appearance. By-example approaches offer unique advantages in this context: our method is easily accessible to casual users, and simple to use through the specification of a 3D model and an image of a pattern.

There are several directions of future work. A first direction is to transpose what we applied on surfaces into a volume synthesis context. The challenges are different: in 3D patterns have more opportunities to connect without violating the appearance specified in a 2D exemplar. However, the computational cost grows dramatically when dealing with volumes. A second direction is to further explore the possible controls. We currently under-exploit the ability of our on-surface synthesizer to orient and scale the synthesized textures. It is also possible to locally change the texture, introducing progressive variations along a surface (e.g. [Zhang et al. 2003]). Both controls can be combined with structural optimization, e.g. orienting the pattern locally to maximally absorb stress.

Acknowledgements

This work was funded by ERC grant ShapeForge (StG-2012-307877). We thank the anonymous reviewers for their helpful comments, and the Lorraine region for providing us with equipment.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ACM, New York, NY, USA, I3D '01, 217–226.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, ACM, New York, NY, USA, SGP '04, 41–44.

BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. 2010. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph. 29*, 6 (Dec.), 166:1–166:10.

CHEN, Y., DAVIS, T. A., HAGER, W. W., AND RAJAMANICKAM, S. 2008. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35*, 3 (Oct.), 22:1–22:14.

CHRISTIANSEN, A. N., BÆRENTZEN, J. A., NOBEL-JØRGENSEN, M., AAGE, N., AND SIGMUND, O. 2015. Combined shape and topology optimization of 3d structures. *Computers & Graphics 46*, 0, 25 – 35. Shape Modeling International 2014.

CHRISTIANSEN, A. N., SCHMIDT, R., AND BÆRENTZEN, J. A. 2015. Automatic balancing of 3d models. *Computer-Aided Design 58*, 0, 236 – 241. Solid and Physical Modeling 2014.

COOK, R. D., MALKUS, D. S., PLESHA, M. E., AND WITT, R. J. 2007. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons.

DONG, Y., LEFEBVRE, S., TONG, X., AND DRETTAKIS, G. 2008. Lazy solid texture synthesis. *Computer Graphics Forum 27*, 4, 1165–1174.

DUMAS, J., HERGEL, J., AND LEFEBVRE, S. 2014. Bridging the gap: Automated steady scaffoldings for 3d printing. *ACM Trans. Graph. 33*, 4 (July), 98:1–98:10.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 341–346.

GAL, R., WEXLER, Y., OFEK, E., HOPPE, H., AND COHEN-OR, D. 2010. Seamless montage for texturing models. *Computer Graphics Forum 29*, 2, 479–486.

GUENNEBAUD, G., JACOB, B., ET AL., 2010. Eigen v3. http://eigen.tuxfamily.org.

HARKER, J., 2011. Crania Anatomica Filigre: Me to You. https://www.kickstarter.com/projects/joshharker/crania-anatomica-filigre-me-to-you.

HAUMONT, D., DEBEIR, O., AND SILLION, F. 2003. Volumetric cell-and-portal generation. *CFG 22*, 3, 303–312.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 327–340.

HU, R., LI, H., ZHANG, H., AND COHEN-OR, D. 2014. Approximate pyramidal shape decomposition. *ACM Trans. Graph. 33*, 6 (Nov.), 213:1–213:12.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (July), 277–286.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (July), 795–802.

LAGAE, A., DUMONT, O., AND DUTRE, P. 2005. Geometry synthesis by example. In *Proceedings of the International Conference on Shape Modeling and Applications 2005*, IEEE Computer Society, Washington, DC, USA, SMI '05, 176–185.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July), 777–786.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. *ACM Trans. Graph. 25*, 3 (July), 541–548.

LEMPITSKY, V., AND IVANOV, D. 2007. Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 1–6.

LIU, L., CHAMBERS, E. W., LETSCHER, D., AND JU, T. 2010. A simple and robust thinning algorithm on cell complexes. *Computer Graphics Forum 29*, 7, 2253–2260.

LU, L., SHARF, A., ZHAO, H., WEI, Y., FAN, Q., CHEN, X., SAVOYE, Y., TU, C., COHEN-OR, D., AND CHEN, B. 2014. Build-to-last: Strength to weight 3d printed objects. *ACM Trans. Graph. 33*, 4 (July), 97:1–97:10.

LUO, L., BARAN, I., RUSINKIEWICZ, S., AND MATUSIK, W. 2012. Chopper: Partitioning models into 3d-printable parts. *ACM Trans. Graph. 31*, 6 (Nov.), 129:1–129:9.

MA, C., HUANG, H., SHEFFER, A., KALOGERAKIS, E., AND WANG, R. 2014. Analogy-driven 3d style transfer. *Computer Graphics Forum 33*, 2, 175–184.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 465–470.

PRÉVOST, R., WHITING, E., LEFEBVRE, S., AND SORKINE-HORNUNG, O. 2013. Make it stand: Balancing shapes for 3d fabrication. *ACM Trans. Graph. 32*, 4 (July), 81:1–81:10.

ROSENKRANTZ, J., AND LOUIS-ROSENBERG, J., 2007. http://n-e-r-v-o-u-s.com/about_us.php.

SCHMIDT, R., AND SINGH, K. 2010. Meshmixer: An interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, ACM, New York, NY, USA, SIGGRAPH '10, 6:1–6:1.

SEGERMAN, H., 2009. Surface autoglyphs. http://www.segerman.org/autologlyphs.html.

SIGMUND, O., AND MAUTE, K. 2013. Topology optimization approaches. *Struct. and Mult. Optimization 48*, 6, 1031–1055.

SIGMUND, O. 2001. A 99 line topology optimization code written in matlab. *Struct. and Mult. Optimization 21*, 2, 120–127.

SOLER, C., CANI, M.-P., AND ANGELIDIS, A. 2002. Hierarchical pattern mapping. *ACM Trans. Graph. 21*, 3 (July), 673–680.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graph. 31*, 4 (July), 48:1–48:11.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph. 21*, 3 (July), 665–672.

TURK, G. 2001. Texture synthesis on surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 347–354.

UMETANI, N., AND SCHMIDT, R. 2013. Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, ACM, New York, NY, USA, SA '13, 5:1–5:4.

VANEK, J., GALICIA, J. A. G., AND BENES, B. 2014. Clever support: Efficient support structure generation for digital fabrication. *Computer Graphics Forum 33*, 5, 117–125.

WANG, W., WANG, T. Y., YANG, Z., LIU, L., TONG, X., TONG, W., DENG, J., CHEN, F., AND LIU, X. 2013. Cost-effective printing of 3d objects with skin-frame structures. *ACM Trans. Graph. 32*, 6 (Nov.), 177:1–177:10.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 355–360.

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

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. In *Rendering Techniques 2001*, S. Gortler and K. Myszkowski, Eds., Eurographics. Springer Vienna, 301–312.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (July), 295–302.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2006. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3 (July), 690–697.

ZHOU, Q., PANETTA, J., AND ZORIN, D. 2013. Worst-case structural analysis. *ACM Trans. Graph. 32*, 4 (July), 137:1–137:12.

ZHOU, S., JIANG, C., AND LEFEBVRE, S. 2014. Topology-constrained synthesis of vector patterns. *ACM Trans. Graph. 33*, 6 (Nov.), 215:1–215:11.