Topology-Constrained Synthesis of Vector Patterns

Shizhe Zhou[†] Changyun Jiang[†] Sylvain Lefebvre[‡] *
[†]USTC [‡]Inria

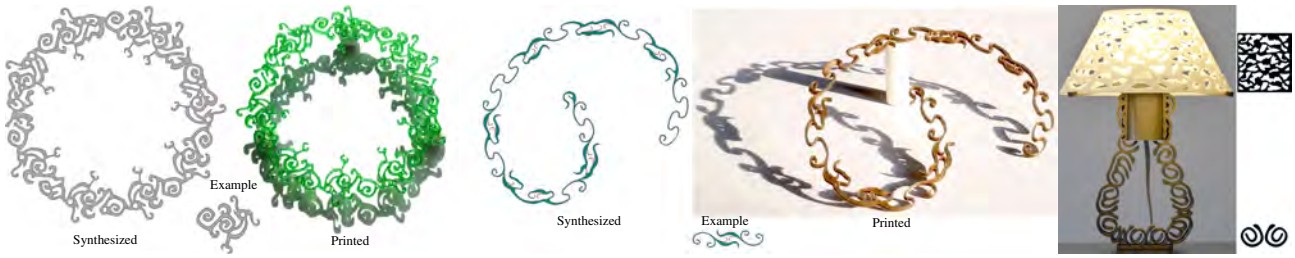


Figure 1: *We use vector shapes as inputs to synthesize patterns along curves. The resulting patterns have a constrained topology allowing them to be 3D printed as a single piece.* Left: *Pattern along a circle and its printout. After printing, the pattern is a single connected object.* Middle: *Pattern along a curve and its printout.* Right: *A printed lamp modeled with our system, the shade and the foot are synthesized.*

Abstract

Decorative patterns are observed in many forms of art, typically enriching the visual aspect of otherwise simple shapes. Such patterns are especially difficult to create, as they often exhibit intricate structural details and at the same time have to precisely match the size and shape of the underlying geometry. In the field of Computer Graphics, several approaches have been proposed to automatically synthesize a decorative pattern along a curve, from an example. This empowers non expert users with a simple brush metaphor, allowing them to easily paint complex structured decorations.

We extend this idea to the space of design and fabrication. The major challenge is to properly account for the topology of the produced patterns. In particular, our technique ensures that synthesized patterns will be made of exactly one connected component, so that once printed they form a single object. To achieve this goal we propose a two steps synthesis process, first synthesizing the topology of the pattern and later synthesizing its exact geometry. We introduce topology descriptors that efficiently capture the topology of the pattern synthesized so far.

We propose several applications of our method, from designing objects using synthesized patterns along curves and within rectangles, to the decoration of surfaces with a dedicated smooth frame interpolation. Using our technique, designers *paint* structured patterns that can be fabricated into solid, tangible objects, creating unusual and surprising designs of lamps, chairs and laces from examples.

CR Categories: I.3.0 [Computer Graphics]: General

Keywords: vector pattern synthesis, topology-constrained

Links: DL PDF WEB VIDEO

*Corresponding author: sylvain.lefebvre@inria.fr (Sylvain Lefebvre)

ACM Reference Format
Zhou, S., Jiang, C., Lefebvre, S. 2014. Topology-Constrained Synthesis of Vector Patterns. ACM Trans. Graph. 33, 6, Article 215 (November 2014), 11 pages. DOI = 10.1145/2661229.2661238 http://doi.acm.org/10.1145/2661229.2661238.

DOI: http://doi.acm.org/10.1145/2661229.2661238

1 Introduction

Our work considers modeling for the purpose of fabrication. In particular, we seek to design algorithms that allow modelers to create rich and detailed objects while ensuring that they will print correctly in 3D. Our favored approach for fabrication is Fused Filament Fabrication (FFF) which creates objects by adding layers after layers of melted plastic filament. This produces objects that are lightweight, detailed and flexible. However, other techniques such as laser cutters can be used to fabricate our designs.

In this paper we focus primarily on decorative elements applied along curves. Such curvilinear details appear ubiquitously in digital contents creation, in particular for jewelry, architecture and decorative furniture design. Our approach lets designers synthesize complex patterns along curves from an input example (see Figure 1). The example patterns are specified as vector graphics. Techniques exist in the Computer Graphics literature for this purpose (Section 2), but our approach uniquely considers the problem of synthesizing patterns *that can be printed into physical objects*. In particular, prior approaches cannot guarantee that the synthesized patterns have a single connected component or contain no holes. The absence of holes is important when patterns are carved out of a piece of material, to avoid the appearance of disconnected parts.

Using our approach, the designer specifies a curve in 2D or 3D and selects a pattern. She additionally specifies topological constraints. The pattern is automatically synthesized along the curve. The designer may locally specify an orientation to locally twist the pattern, or may apply the curve along a tessellated object surface. By applying our technique twice, our approach can also synthesize connected patterns within rectangular domains, as for the lamp shade in Figure 1. The amount of variety in the pattern is optionally controlled by penalizing autocorrelation within the result. At the end of the interactive session the synthesized pattern is given a thickness and turned into a 3D mesh that can be 3D printed. Entire objects can be fabricated by assembling multiple curvilinear parts.

Our contributions are:

- A formulation of the pattern synthesis problem for fabrication.
- The introduction of *topology descriptors* that enable the inclusion of topological properties in the synthesis process.
- A novel synthesizer optimizing for the topology and the geometry of the patterns. Results have exactly the number of connected components and holes specified as a constraint.
- A topology driven approach to automatically analyze and decompose an input exemplar pattern into a set of pieces used during the synthesis process.

2 Related Works

We focus here on works using data–driven approaches to generate patterns along curves. Such techniques have been proposed in different contexts, for instance for terrain synthesis [Zhou et al. 2007; Sibbing et al. 2010], painterly rendering [Barla et al. 2006; Lu et al. 2013] and image inpaiting [Sun et al. 2005]. We organize the works below by their core techniques rather than their applications.

Discrete elements. A natural way of enriching a curve with texture is to mimic the process of a paint brush, combining discrete paint strokes along the curve. Several works have been proposed for this purpose, distributing elements according to an example along curves but also more generally in the plane [Hurtut et al. 2009; Ma et al. 2011; Kazi et al. 2012; Landes et al. 2013; AlMeraj et al. 2013]. These approaches generally optimize the neighborhoods of discrete elements in the result so that they resemble those in the example. They do not attempt to generate a continuous pattern and are not well suited to our purpose.

Curve synthesis. Rather than synthesizing patterns along a curve, some approaches target the synthesis of the curve itself [Hertzmann et al. 2002; Merrell and Manocha 2010; Lu et al. 2012]. These approaches would be well suited for fabrication since the generated curves are continuous, have a single component and are vector shapes. However, only curves of simple topology (lines) are considered as input while our work is more general and supports arbitrary patterns as input.

Stochastic pattern synthesis. Stochastic pattern synthesis has been demonstrated successfully along curves for terrains [Zhou et al. 2007; Sibbing et al. 2010], for image inpainting [Sun et al. 2005], and for line stylization [Bénard et al. 2010; Kim and Shin 2010; Ando and Tsuruno 2010; Lu et al. 2013; Lukáč et al. 2013]. These works typically formulate the problem as assembling a number of elementary patches from the example along the curve, minimizing the transition error between two successive patches. Such approaches are designed for input images with some degree of randomness and usually do not perform equally well on structured patterns. This is a major issue when targeting fabrication as any discontinuity in the pattern leads to a broken print.

Structured pattern synthesis. Techniques for structured pattern synthesis employ a similar formulation but exact optimizers, such as shortest path searches [Lefebvre et al. 2010; Alhashim et al. 2012] or dynamic programming [Zhou et al. 2013; Lu et al. 2014]. This is necessary as the structure in the input makes good solutions much less likely, and therefore stochastic optimizers no longer easily find satisfying results. These approaches also explicitly include geometric correspondences between successive pieces as an objective, ensuring contiguous patches of geometry correspond well [Zhou et al. 2006]. While generating great visual results, the aforementioned techniques do not provide any guarantee on the topology of the final pattern and would not be suitable for fabrication. Our goal in this paper is to build upon these techniques and make them applicable to the purpose of designing from examples decorative patterns that can be fabricated, taking into account the global topology of the result.

3 Topology-Constrained Synthesis

3.1 Overview

From an input 2D vector shape (Section 3.2) our algorithm generates a pattern within a band defined around a curve. Prior to synthesis, the exemplar is divided into a number of pieces (Section 3.3),

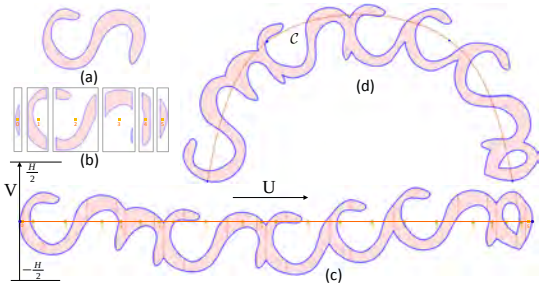


Figure 2: *(a) Input exemplar. (b) Division into six pieces. (c) Pattern synthesized in the parametric space of curve $\mathcal{C}$, within a band of height H. (d) Result mapped along the curve.*

used as building blocks positioned along the curve (Section 3.5). Synthesis is performed within the parametric domain of the curve. This is summarized in Figure 2.

Assembling pieces along a single dimension is now typical for pattern synthesis. However enforcing the topology constraints brings new challenges. Rather than optimizing for the final pattern by only considering the matching cost of successive pieces; we optimize for results in two steps: topology and geometry.

We first optimize for the topology of the result, choosing a sequence of pieces that enforces the user selected number of components and holes (Section 3.6). To properly capture how the topology of the pattern being synthesized evolves, we introduce *topology descriptors* (Section 3.4). The topology of the final pattern is *guaranteed*, as opposed to be simply encouraged through an objective function. After this step, the geometry of the pattern is not yet resolved. We solve for the geometry in a second optimizer, which employs vector shape deformation techniques to obtain a smooth, gap–free geometry (Section 4).

3.2 Exemplars

The input exemplars of our algorithm are 2D vector shapes. These shapes are assumed to properly define interior/exterior areas in the plane, but may have multiple disconnected components (see for instance Figures 1, 21, 22).

In our work input 2D shapes are described by simple general polygons with holes (SGPH). Each polygon is a non-intersecting contour line which can be convex or concave. The polygons do not overlap. Polygons with a counter clockwise orientation describe exterior boundaries, while polygon with counter clockwise orientation describe holes. The orientation can be automatically recovered since the contours are closed and non–intersecting.

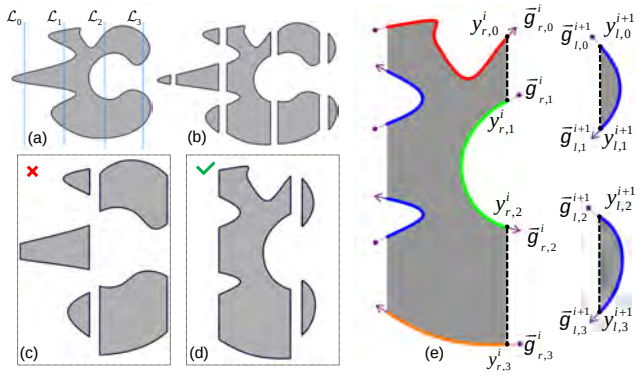


Figure 3: *(a,b) Building pieces from an input exemplar. (c,d) Our algorithm only considers matching two pieces with the same number of portals (split lines marked by black dash lines in (e)).*

3.3 Splitting into pieces

Our synthesis process forms a new pattern by putting together pieces extracted from the examplar. We use t parallel slicing lines to cut the input shape into $t+1$ separated pieces denoted $P_0, ..., P_t$, as shown in Figure 3. We denote each piece width by $w_0, ..., w_t$. The width of each piece may vary. The selection of lines is either uniform or uses our topological analysis described in Section 5. We assume for now that the choice of split lines is given as input.

Each split piece has two sides (left/right) with a number of *portals* along each: a portal is a segment along which the pattern interior has been sliced. We denote by $M_{l,i}$ and $M_{r,i}$ the number of portal endpoints for the left and right side of piece i (there are two endpoints per portal). We denote by $y^j_{l,i}$ and $y^j_{r,i}$ the vertical position of the j-th portal endpoint of piece i for respectively the left and right sides. The endpoints are ordered from top to bottom, so that $y^0_{r,i}$ is the highest portal endpoint on the right side. We denote by $g^j_{l,i}$ and $g^j_{r,i}$ the tangent of the piece geometry at the j-th portal endpoint for respectively the left and right sides of piece i.

3.4 Topology descriptors

To ensure that the result can be printed as a single object, the synthesized pattern has to form a single connected component. As illustrated in the inset it is not enough to match the number of portals on each sides of successive pieces. In the shown example, the pattern has five components instead of a single one, even though no open portal remains.

To faithfully capture the topology of the synthesized patterns, we introduce *topology descriptors*. Each piece has a topology descriptor which captures how portals on both sides are connected through the pattern within. The topology descriptor contains for each side a small array with one entry per portal. Each entry stores the ID of the connected component to which the corresponding portal belongs. The descriptor also stores two counters. The first tracks the number of closed components enclosed inside, while the second tracks the number of inner holes.

When two pieces are joined, the topology descriptor of the obtained larger piece reflects the (potential) merge of connected components inside. The left/right descriptor of the result is obtained using the two descriptors being joined: the component IDs are rewritten, merging those which are connected through a portal and keeping others unique. If the joint produces a component without any remaining portal, then it lies entirely inside and the connected component counter is incremented. Similarly, if two (left) portals with same ID are merged with two (right) portals with same IDs, the hole counter incremented. Figure 4 illustrates this process. The merging operation takes $O(n \log n)$ time with n the number of portals (small in practice).

Topology descriptors are used during the optimization to keep track of the topology of intermediate results: each time a new piece is added to an intermediate result their descriptors are merged. When no portal remains the pattern cannot grow further and the counters indicate the numbers of components and holes of the final result.

3.5 Synthesis

We synthesize a new pattern along a band of width H, defined around a parametrized smooth curve $\mathcal{C}$ of arc length L. We perform synthesis within the parameterized domain of $\mathcal{C}$ – denoted by the U and V axis next. Therefore, synthesis is always performed

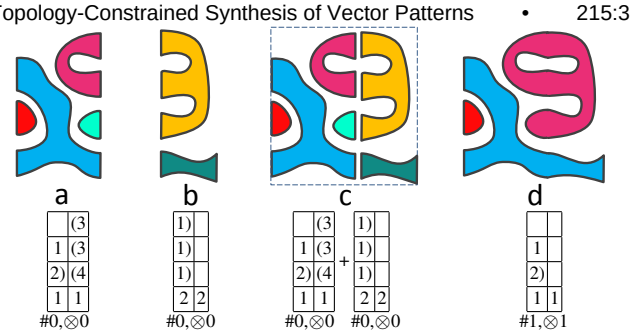


Figure 4: *Illustration of our topology descriptors. (a) A piece with four components, one ending and two starting inside. (b) A piece with two components, one ending inside. (c) Combining (a) and (b) merges their topology descriptors at their four common portals. (d) The result after merging, with only three remaining connected components and only one portal at the right. Note that the 'g' shaped component is closed and will no longer change. The number of enclosed components is therefore incremented to '#1'. Closing the 'g' also adds a hole when merging component ID 3 (left) with component ID 1 (right), incrementing the hole counter to '⊗1'.*

along a straight line, parallel to the U axis. The V axis represents the axis orthogonal to $\mathcal{C}$, as illustrated Figure 2. We allow pieces to slide orthogonally to $\mathcal{C}$ inside the band; i.e. each piece can be offset by a value $\delta \in (-\frac{H}{2}, \frac{H}{2})$.

Our goal is to find a sequence of pieces and offsets forming a new pattern. We denote the sequence $\Pi = (\eta_0, \delta_0), ..., (\eta_N, \delta_N)$ where (η_i, δ_i) indicates that piece P_{η_i} is to be used at location i in the sequence with vertical offset δ_i. Π therefore contains two subsequences, one for the choice of pieces $\eta = (\eta_0, ..., \eta_N)$ and one for vertical positioning of the pieces $\delta = (\delta_0, ..., \delta_N)$ along the V axis.

The best sequence is optimized under the following requirements:

1. **[Constraint]** Successive pieces are joined by boundaries having strictly the same number of portals, i.e. $M_{r,\eta_i} = M_{l,\eta_{i+1}}$, and the final pattern must be closed (no open portal).
2. **[Constraint]** (optional) The number of connected components in the pattern has to match a user specified target.
3. **[Constraint]** (optional) The number of holes in the pattern has to match a user specified target.
4. **[Objective]** The length of the resulting pattern $l = \sum_{i=0}^{N} w_{\eta_i}$ is close to the curve length L. I.e. $|l - L| < \epsilon$. The initial value of ϵ is $0.05 \cdot L$.
5. **[Objective]** The result is a pattern visually similar to the input example content, i.e. the seams from one piece to the next are as inconspicuous as possible.

The constraints are strictly enforced, while the objectives are optimized for. The sequence Π contains terms η and δ which have a different nature. The choice of terms η fixes the topology of the final pattern, by deciding how pieces connect together. The terms δ determine the actual geometry of the final pattern. We optimize in two steps: dynamic programming with topology descriptor for η (Section 3.4) and global geometry optimization for δ (Section 4).

If curve $\mathcal{C}$ is open (e.g. not a loop), we use piece P_0 and piece P_t as the left and right extremities (i.e. $\eta_0 = 0$, $\eta_N = t$) as they are the natural choice for starting/ending the pattern. If $\mathcal{C}$ is a closed curve, we optimize for one more virtual piece η_{N+1} constrained by $\eta_0 = \eta_{N+1}$ and set $\delta_0 = 0$ to place the first piece at the band center.

3.6 Topology solver

We optimize for the best sequence of pieces η by dynamic programming (DP). While this has been classically done for pattern synthesis, the novelty of our approach is to use the DP solver for the topology only, through the use of our topology descriptors.

The topology optimizer assumes that the precise geometry of the pieces will be later optimized through the δ terms, and therefore could entirely ignore the geometry. However it is desirable to include some geometric aspects as a secondary objective to simplify the geometry optimization. For instance mismatching tangents or severe deviation from the synthesis curve could make the geometry optimization much more difficult. Our objective function for these secondary objectives is described next.

3.6.1 Objective function

We define the cost of a given choice of η as:

$$F(\eta) = \sum_{i=0}^{N-1} \mathcal{D}(\eta_i, \eta_{i+1})$$

where $\mathcal{D}$ is the cost of matching two successive pieces as described next. During synthesis we seek for $\tilde{\eta} = \arg\min_{\eta} F(\eta)$.

We define the cost of successive pieces as:

$$\mathcal{D}(i,j) = d_{match}(i,j) \cdot \alpha + (1-\alpha) \cdot d_{tangent}(i,j)$$

where $d_{tangent}$ measures whether the shape tangents match while d_{match} measures if the portal endpoints position match. We use $\alpha = 0.8$ in all results.

More precisely, $d_{tangent}$ is

$$d_{tangent}(i,j) = \sum_{k=0}^{M-1} \|1 - \vec{g}^k_{r,i} \cdot \vec{g}^k_{l,j}\|.$$

where M is the number of portal endpoints between pieces i and j. Pieces can only be selected as neighbors if their number of matching portals is exactly the same, i.e. $M_{r,i} = M = M_{l,i+1}$.

d_{match} is the sum of distances between corresponding portal endpoints along the V axis, when the two pieces are optimally rigidly aligned by a vertical offset.

$$d_{match}(i,j) = \min_{\tau} \left(\frac{1}{M} \sum_{k=0}^{M-1} ||y^k_{l,j} - y^k_{r,i} + \tau||^2 \right)$$

For efficiency we precompute d_{match}, $d_{tangent}$ and the optimal alignment $\tilde{\tau}$ for all pairs of pieces i, j having matching number of portals ($M_{r,i} = M_{l,i+1}$), and store them in a 2D lookup table.

Note that optimizing for $\tilde{\eta}$ through F does not involve knowing the sequence δ. The cost always assumes that the best vertical alignment $\tilde{\tau}$ will be used, thus abstracting away the global geometry optimization that is performed once the choice of pieces is completely determined.

3.6.2 Filling the DP table

We perform DP in a sparse, 3-dimensional table T illustrated in the inset next. The three dimensions are indexed by the length of the synthesized sequence Π, the index of the pieces, and the topology descriptors. An entry $T[k, i, \Upsilon]$ thus represents the information

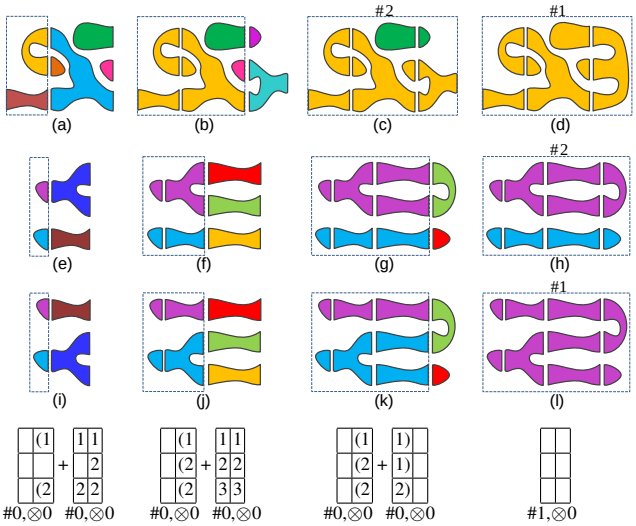
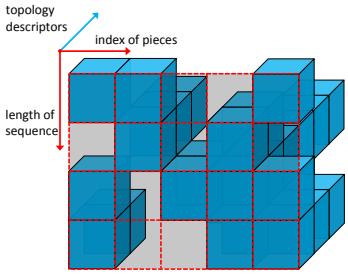


Figure 5: *Tracking topology configurations during optimization. Please refer to the text for details. The last row shows the topology descriptors of each intermediate step (i),(j),(k),(l). The last descriptor has no open portal but indicates correctly that a single closed connected component without holes is inside.*



of the best solution using a sequence of k pieces, ending on piece P_i and having a topology descriptor Υ. The entry stores the cost, the length l of the resulting pattern, and the accumulated vertical offset Δ for the current result. Most importantly, the entry stores the topology descriptor Υ of the best solution so far. The main direction for the DP solver is the number of pieces in the synthesized sequence: at each iteration, we compute the best solutions using $k+1$ pieces from the best solutions using k pieces. The dimension storing topology descriptors is sparse, as only a small number of topology descriptors may be reached when synthesizing from a given example pattern.

At a given step of the DP different intermediate topology configurations occur. They evolve into simpler or more complex configurations as more and more pieces are added. For example in Figure 5 (top row), two different choices of a third piece from case (b) lead to two different topology configurations between (c) and (d). The first choice gives two components while the second results in a single component. Intermediate multiple components solutions can be merged into single component solutions, and vice versa.

Keeping track only of the number of connected components is not enough, as illustrated in Figure 5 (rows two and three). Cases (f) and (j) use a same piece in the third step, and have the same number of components but different topology configurations. The two obtained results differ in their number of connected components and holes: two components and one hole for (h) and one component with no hole for (l). The connectivity between components through portals is the important factor deciding the final topology. This is captured by the topology descriptors of intermediate solutions.

At iteration $k+1$ we consider the solutions that can be generated from the solutions of size k. We proceed as follows: let us consider the solution of size k in $T[k, i, \Upsilon]$. We consider enlarging the

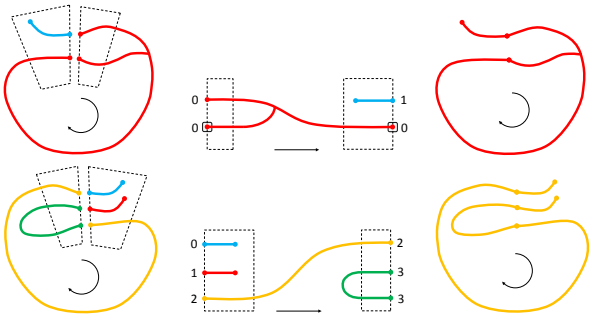


Figure 6: *Synthesizing along a closed curve does not guarantee a closed pattern.* Top: *When joining the first and last piece, a closed loop is formed thanks to component ID 0 which has connected portals across the entire pattern (small black box).* Bottom: *When joining the first and last piece, only component ID 2 is connected from the start to the end. The final pattern has a single connected component, but the pattern itself is not closed.*

solution with all pieces P_j such that the left of P_j is compatible with the right of P_i, that is $M_{l,j} = M_{r,i}$. The cost of the solution augmented with P_j is:

$$T[k, i, \Upsilon].\text{cost} + \mathcal{D}(i, j)$$

The topology descriptor Υ' of the enlarged solution is given by merging the topology descriptor of P_j with Υ. We then store this solution in $T[k+1, j, \Upsilon']$. If a solution had already been found with the same topology descriptor, we keep the one of lowest cost. We store in $T[k+1, j, \Upsilon']$ the cost, the cumulative offset $\Delta + \tilde{\tau}_{ij}$ and the length of the resulting pattern $l + w_j$. ($\tilde{\tau}_{ij}$ gives the best rigid alignment between P_j and P_i, see Section 3.6.1).

When possible we ignore partial solutions that exceed the constraints in terms of number of components and holes, by not growing them further. Offsets are accumulated in Δ during the DP to discard results that would drift too far from the horizontal axis, canceling partial solutions where a piece would be offset outside the height H. This avoids generating challenging cases for the geometric optimization (Section 4). Similarly, we do not further enlarge solutions that already reached the desired length, i.e., $|l - L| < \epsilon$.

We next describe how backtracking is performed once the final DP table is obtained.

3.6.3 Backtracking

Opened curve For a same target length L the available solutions have a different number of pieces and different costs. Before backtracking we consider all table entries where the length reached the target, and where the number of components and holes in Υ matches the user target (typically, one component and/or no hole). Among the entries matching the constraints, we backtrack from the entry of lowest cost.

In the rare cases where no entry reaches the target length, we increase ϵ until a solution is found. This does not require re-running the DP and the geometry optimization later deforms the pattern to match the target length exactly. Such cases typically occur when synthesizing patterns that are short compared to the piece widths.

Note that for the case of opened curves, the topology descriptors can be simplified. Indeed, the left side of the result is never used, and therefore does not need to be computed. This both reduces the memory required to store the descriptors as well as the time to update them.

Closed curve When synthesizing along a closed curve (loop) we artificially add a last piece $P_{\eta_{N+1}}$. The first piece P_{η_0} and the last piece $P_{\eta_{N+1}}$ have to be the same, so that P_{η_N} connects back to P_{η_0}. To avoid running multiple constrained DP we search a cycle in the table – if such a cycle exists it is a short path and generally has a low cost [Lasram and Lefebvre 2012]. A cycle does not necessarily exists in our setting, in which case we search for a longer pattern by slightly increasing the target length. Cycles become rare only when the value of L is small compared to the width of the pieces.

We find all the cycles and evaluate their number of components and holes: when the piece P_{η_N} connects back to P_{η_0} the topology descriptor is merged with itself to properly account for the entire loop. This merged information is used to decide whether to backtrack. Note that even with a single component the pattern itself may not be closed. An example is given Figure 6. If a closed pattern is desired we further filter out opened patterns.

3.7 Variety

A drawback of DP synthesizers is that the optimal solutions they find tend to produce results with periodicities. The issue stems from the existence of minimizing cycles in the space of solutions. Our topology driven analysis, described in section 5.2, prevents trivial cycles to exist (a single piece loop) by ensuring that no two subsequent pieces have the same number of portals on their left side. Nevertheless, longer minimizing cycles might still exist.

We let the user explicitly request variation by controlling the maximum autocorrelation of the synthesized pattern. We define the autocorrelation of the synthesis sequence as follows

$$A(\eta_0 ... \eta_K) = \max_{j \in [0,K]} \left(\sum_{i \in [0, K-j]} e^{-\frac{|\eta_i - \eta_{i+j}|}{\sigma}} \right)$$

where $\sigma = 0.5$. Note that the distance is in the index space of the pieces, since pieces with nearby indices tend to have a similar appearance. For a partial solution of size K, $A(\eta)$ reflects whether similar subsequences of pieces appear: a high autocorrelation value implies that periodicities exist.

The user optionally selects a weight β that adds the autocorrelation to the cost of the current solution in the DP table. The higher the weight, the more autocorrelation is discouraged. If β is 0 the autocorrelation term is not computed.

Figure 7 shows results for different autocorrelation weight values.

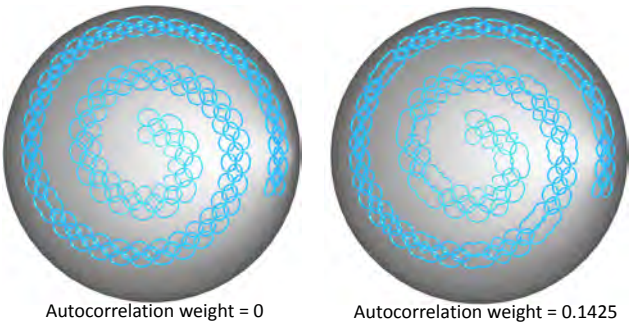


Figure 7: Left: *The optimizer tends to generate visually regular patterns with repeating low cost cycles.* Right: *Penalizing the autocorrelation produces more variety and less regularity.*

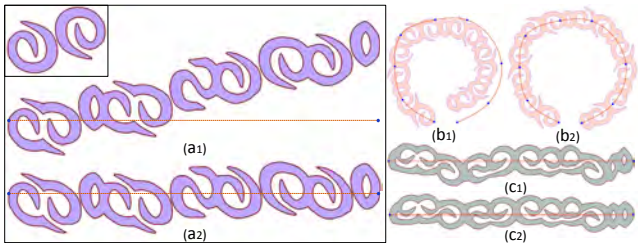


Figure 8: *Results with (a_2, b_2, c_2) and without (a_1, b_1, c_1) global band fairing. The band fairing step avoids the result to drift away from the curve skeleton, and ensures a globally smooth appearance.*

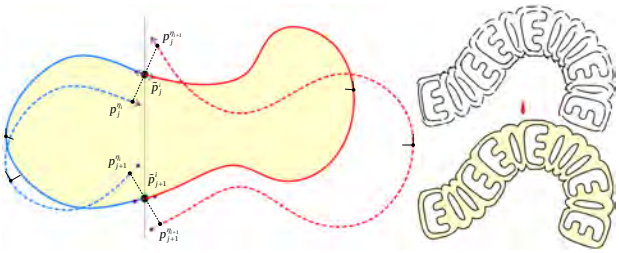


Figure 9: *Linear rotation invariant deformation is used to deform the boundaries of the pieces and obtain a seamless result.*

4 Geometry optimization

After the DP optimization a sequence of pieces η is chosen, but the pieces are not yet assigned with a vertical positioning δ. Naively using the offsets accumulated during the topology step would result in strong deviations from the curve, as illustrated Figure 8(a_1). In this case a tilted exemplar is used and the local alignments raise the pieces one after the other to minimize the matching cost. As a consequence the rightmost piece drifts away from $\mathcal{C}$.

Therefore, we formulate the choice of δ as a global linear system to apply a smooth deformation maintaining the band close to the curve (Section 4.1). After this process a few gaps remain. We perform a final gap stitching optimization to obtain the final result (Section 4.2).

4.1 Positioning the pieces

We determine the optimal sequence of offsets δ as

$$\tilde{\delta} = \arg\min \left(\sum_{i=0}^{N-1} M_{r,i} ||\delta_i + \tilde{\tau}_{\eta_i, \eta_{i+1}} - \delta_{i+1}||^2 \right)$$
$$\text{with } \delta_0 = 0, \delta_N = 0$$

This keeps pieces close to their optimal alignment $\tilde{\tau}$ while ensuring that the pattern endpoints δ_0 and δ_N lie on the horizontal axis. The error is weighted by the number of portals to ensure that all connections are similarly considered. Results are shown in Figure 8.

4.2 Gap stitching

The objective function used during the topology step (Section 3.6.1) takes into account width and tangent mismatch between portals. However, the final result rarely exhibits a perfect match and portal endpoints do not perfectly align even after optimizing for δ. In addition, the actual accumulated string length l is rarely exactly equal to the target length L, leaving a gap in between the pieces (see Figure 9, top right).

Figure 10: *Various synthesis results for different choices of the number of uniform slicing lines.*

We stitch the gaps and re-orient the pieces using linear rotation invariant deformation [Lipman et al. 2005]. We pre-factorize the coefficient matrices for every boundary of the original pieces $P_0..,P_t$ and store them in memory. After synthesis we snap the open boundaries by merging the portal endpoints and tangents (see Figure 9). The new endpoints and their tangents are computed as the average of matching endpoints and tangents on both sides. The linear rotation invariant deformation updates the local frames and position of all the vertices along the boundaries, preserving their appearances while stitching the gaps. Thanks to the pre-factoring on the example pieces the deformation is very fast.

4.3 Generating the final geometry

We extract oriented contours from the synthesized shape after gap stitching, merging the boundaries of independent pieces into longer boundaries forming closed contours. After this step the outer boundary has a clockwise orientation, while holes have a counter-clockwise orientation.

5 Shape analysis

We described how the exemplars are split into pieces in Section 3.3. We assumed there that the splitting lines were given. We now discuss the selection of the split lines. This is an important factor to the quality of the end result.

5.1 Uniform slicing

A simple approach is to perform a uniform slicing. We first compute the bounding box of the exemplar to obtain its width W. The user inputs a number of slicing lines t which are uniformly spaced by a distance $\frac{W}{t+1}$ along the exemplar width.

The choice of t has a dramatic impact on the synthesis result and quality as illustrated in Figure 10. An advantage of having a uniform width for pieces is that we can precompute the exact number of pieces needed to reach the target length L. Therefore the length constraint (Section 3.5) does not have to be considered during the topology step and the DP problem becomes lighter. However, as we explain next, uniform slicing has clear drawbacks.

5.2 Non-uniform, topology driven slicing

We now consider the problem of automatically decomposing the pattern into pieces. We note that if we aim at rich variations as well as global similarity to the exemplar, it is important to exploit the structure information *inside* each piece. Specifically, we want the topology information to be atomically contained in each piece, so that in every step of the optimization a single element of the topology information from the input pattern is introduced.

Indeed, having several topological changes clustered inside a single piece prevents the optimizer from exploiting these changes to

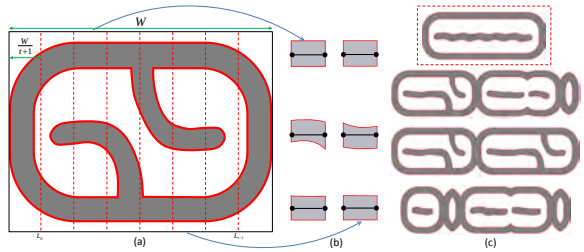


Figure 11: *Uniform slicing. (a) The exemplar is sliced into eight pieces of equal width. (b) Ignoring topology changes generates trivial pieces, which leads to trivial synthesis results (top right, red dash box), breaking the similarity to the input exemplar.*

introduce variety in the result. Having topologically trivial pieces – no merging, no branching, no region beginning or ending – is also problematic. This wastes processing during the optimization since the piece has zero contribution to the topological variation. This also often introduces a trivial cycle when multiple such pieces have a low matching cost. Uniform slicing often produces these two undesirable cases, because the topology changes are usually non-uniformly distributed in a shape. This is illustrated Figure 11.

We use a Reeb graph [Biasotti et al. 2008] to analyze the topology of the input patterns along the U axis. Each node in a Reeb graph indicates a topology change. We first render the input exemplar into an image: the interior regions are filled with opaque pixels while the exterior and hole regions are transparent. We sweep a line along the U axis and scan the columns of the image to detect topology changes such as columns splitting or merging, or a column sequence beginning or ending. We record the coordinates where these changes occur. After scanning, we get a partition of the interior region of the exemplar, adjacent parts meeting with each other at corresponding attaching handles (black vertical lines, Figure 12(b)), whose U coordinates $A_0, ..., A_k$ are noted as *attachers*. We take the middle point $C_i = \frac{1}{2}(A_i + A_{i+1})$ of each pair of (A_i, A_{i+1}) as slicing positions (Figure 12, red dash vertical lines).

This offers a number of desirable guarantees: all the resulting pieces have a non-trivial topology; there is exactly only *one* topology change occurring inside each piece; no two adjacent pieces have a same topology; there are no isolated region (islands) in the pieces. The reason for this last property is that all the topology changes are captured by the Reeb graph detection, so there will be at least two attacher segments on the starting and ending positions of an island, giving at least one slicing lines on it. Islands enclosed within a piece would otherwise require a special treatment when computing the offsets δ and when performing gap stitching.

In this paper, we use topology driven non-uniform slicing in all the results unless otherwise specified (e.g. Figure 10, Figure 11 (c)).

5.2.1 Thin features

Up to now we only considered the topology when searching for the slicing positions. However, some slicing lines may have a very narrow intersection with the input shape. Specifically, if a narrow portal matches with another very narrow one (*weak connection*), the final 3D printed patterns could be fragile along these connections. We filter out these slicing lines by considering whether the narrowest portal has a width smaller than a threshold. The threshold is chosen by the user based on the printing material – with larger thresholds increasing the strength of the printed pattern.

Note that in this work we do not further consider the mechanical property of the fabricated pattern, but other works have focused on strengthening a model before fabrication [Stava et al. 2012].

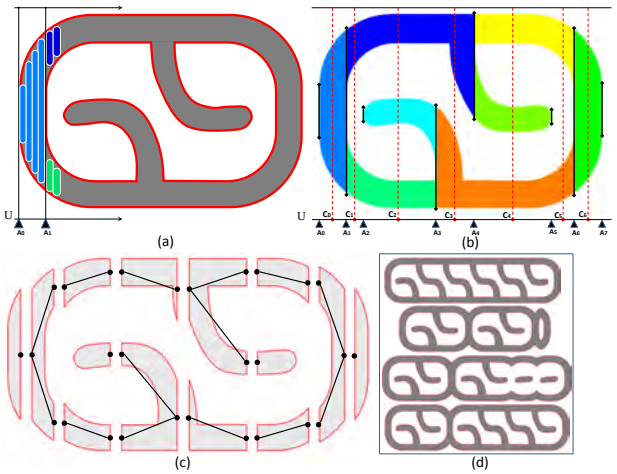


Figure 12: *(a) We first turn the input exemplar into an image whose pixel columns are scanned one by one along the U axis. For exemplars with concave border or holes, at certain coordinates there are topology changes such as columns splitting or merging, or a column sequence beginning or ending. We record these coordinates $A_0, ..., A_7$ as attachers (small black triangles along the axis). (b) After scanning, we get a partition of the interior region of the exemplar, adjacent parts meeting with each others at corresponding attaching handles (black vertical lines). We take the middle point $C_i = \frac{1}{2}(A_i + A_{i+1})$ of each pair of (A_i, A_{i+1}) as the slicing position (red dash vertical lines). (c) It is guaranteed that all the resulting piece have non-trivial topology. (d) The final synthesis results preserve topological similarity to the input shape while exhibiting interesting variations.*

5.2.2 Implementation

The input pattern might have a noisy outline, producing many small topological events which have a detrimental impact on performance. In addition, since we rasterize the vector shape before scanning for its toplogy, aliasing can produce unnecessary topology changes.

We filter small topological events by merging them based on their successive horizontal distances. A group keeps growing from right to left until the next attacher is located further from the last one by a distance larger than $0.2 \cdot W$ (experimentally determined). After the grouping, we compute the average center of each group and use them as the new attachers.

6 Applications and Results

6.1 Synthesizing patterns along 2D curves

Figure 21 illustrate a variety of results from our approach. Such 2D curves are easy to print with a low-cost FDM printers as shown Figure 1, with the benefit that thin features are well captured by a single thread of plastic. The printed patterns remain quite flexible and are easy to bend for instance to produce wristbands or laces. Figure 24 demonstrates the importance of holes control when carving patterns. Figure 16 shows how the synthesizer exploits the width of the synthesis band.

Our synthesizer can be used to design full objects, such as the multi-curve lamp shown in Figure 13. Here, the designer only had to specify the shape of the curves while our system automatically generated patterns that can physically act as supports thanks to their single connected topology.

Figure 13: *A lamp made of six curves synthesized with different patterns. Printed on an Ultimaker 2 (PLA plastic filament).*

Comparison In work concurrent to ours, [Lu et al. 2014] introduced a vector pattern synthesis scheme along curves. While their focus is different – the topology is not taken into account – we propose in Figure 14 a brief comparison indicating that our approach produces results of similar quality. Our scheme can additionally constrain the topology of the resulting patterns.

Figure 14: Left: *The result of Lu et al.* Middle: *Our result using a similar example. The number of components is free but we request a result without holes. This pattern can be carved out of a piece of material, e.g. for fabricating a sign.* Right: *Our result on a different example, this time requesting a single component for each curve.*

Performance Performance for a selection of results is summarized in Table 1, measured on our (non-parallel) CPU implementation running on an Intel i7 4770K 4.2GHz, 8G ram 2400MHz. The table contains the numbers for all six curves in Figure 13, showing how different exemplars impact performance for a same curve. Synthesis time is largely dominated by the dynamic programing, and therefore we do not report backtracking and geometry passes times which are in the order of a few milliseconds. As can be seen, our technique gives feedback in a few seconds in most cases. The major factors in performance are the number of input pieces as well as the topological complexity of the input. Many optimizations are possible, to run DP on the GPU and exploit coherence of partial DP solutions during user edits [Zhou et al. 2013; Lu et al. 2014].

6.2 Synthesizing rectangular patterns

An interesting extension of our approach is to produce two dimensional patterns with constrained topology. To achieve this, we apply our method twice, along the U and V axis, rotating the first result by 90 degrees and using it as the example for the second step. Since our scheme generates patterns with exact target length, patterns of any rectangular size can be synthesized. Results are illustrated Figure 17. This is also used to produce the lamp shade in Figure 15 as well as the seating area of the chair model in Figure 18.

Model	DP (sec)	# pieces in input	# pieces in result	# correct paths	Memory (peak)
Figure 13	0.13	11	56	29	62 MB
Figure 13	1.3	14	74	81	67.3 MB
Figure 13	9.9	14	109	465	101.5 MB
Figure 13	2.2	13	65	157	70.5 MB
Figure 13	1.2	15	70	148	64.9 MB
Figure 13	0.78	10	75	134	61 MB
Figure 1 (left) ⋆	62.1	20	147	110	230 MB
Chair side (circle) ⋆	143	18	190	1422	453 MB
Chair foot	8.3	12	130	205	73.9 MB
Chair back (axis 1)	0.007	15	64	1	88.7 MB
Chair back (axis 2)	5.9	14	102	64	295.8 MB

Table 1: *Performance measurements for a selection of results. Results marked by a ⋆ are cyclic, closed patterns. All results use the topological analysis and the auto-correlation penalty.*

Figure 15: *These lamps use the same shade synthesized by our rectangular pattern synthesis. The feet are synthesized from a curve.*

6.3 Decorating surfaces

Synthesis band We use our topology constrained synthesizer to decorate shapes by generating surface-conformal 3D bands along which synthesis is performed. Our method lets the user draw a smooth curve along a triangular mesh Ψ by specifying control points on its surface. The challenge is to ensure that the curve remains exactly on Ψ and has stable smooth local frames.

We first parameterize Ψ onto a convex border (e.g. square border) 2D mesh Γ [Desbrun et al. 2002]. The mapping is denoted $\Phi : \Psi \rightarrow \Gamma$ The user selected control points $c_0, .., c_n$ on Ψ are then mapped onto Γ as $\hat{c}_0, .., \hat{c}_n$, with $\Phi(c_i) = \hat{c}_i$. A 2D spline curve $\hat{S}$ is constructed on Γ using $\hat{c}_0, .., \hat{c}_n$ as control points. We map the curve S back to the surface Ψ by the inverse mapping $S = \Phi^{-1}(\hat{S})$. Besides uniformly sampling $\hat{S}$, we also include all of its intersection points with the edges of Γ (red dots in Figure 19 (c)). Therefore, all the segments of the resulting polygon curve S lie exactly on Ψ. S is parameterized by arc-length.

The stability and smoothness of the local frames is important to generate a visually pleasing pattern. However directly relying on the facet normals often leads to discontinuities since the triangular mesh Ψ is often non-smooth depending on the tessellation. We therefore compute new, smooth local frames along S. We first uniformly sample a set of p points $G = G_0, ..., G_{p-1}$ along S with associated normals $n_0, ..., n_{p-1}$ from Ψ. We then send all the points in G to the center of a unit sphere Θ, where we project G_i by n_i onto Θ. The projected points are noted as $\tilde{G}_i$. For each $\tilde{G}_i$ we define the tangent vector $\tilde{b}_i$ to be the forward difference $|n_{i+1} - n_i|$

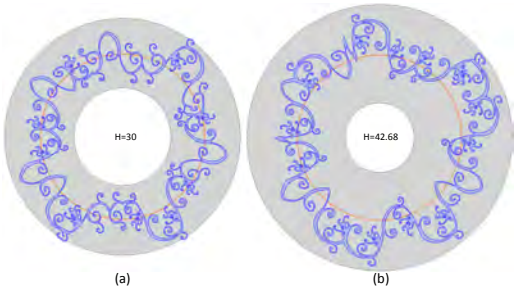


Figure 16: *Using different curve bandwidth H in (a) and (b) produce different synthesis results, all other parameters being equal.*

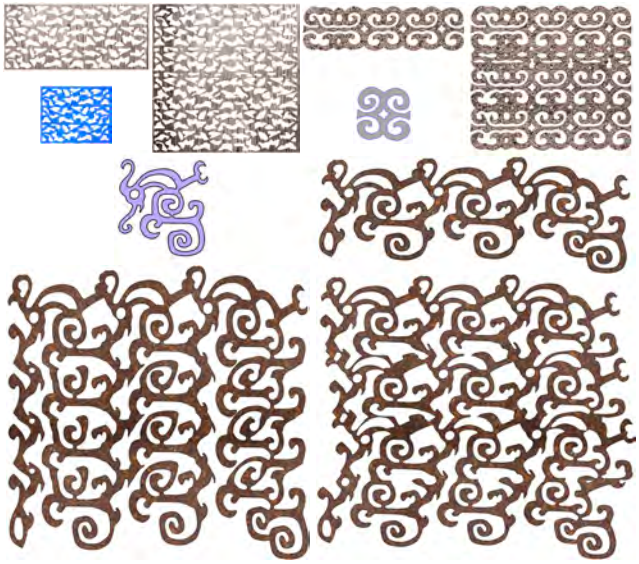


Figure 17: *We apply our method twice along U and V axis in sequence to get rectangular results. Note how the final pattern still forms a single connected component. The last row shows results for different autocorrelation penalties.*

projected onto the tangent plane of Θ at $\tilde{G}_i$. We next build a cubic Bézier spline f using $\tilde{G}_i$ as control points with tangent $\tilde{b}_i$. Then f is projected onto the surface of Θ from the sphere center to get a *normal trace* F, illustrated as the red curve in Figure 20 (g). The new normal of a point $S(t')$ is evaluated on F which immediately gives the new normal $n(t')$. Finally we project $n(t')$ onto the normal plane of $S(t')$ defined by its initial tangent. The resulting local frames are a smooth approximation of the initial frames. They produce smooth synthesis results as illustrated in Figure 20 (h,i).

Generating the final mesh After synthesizing the pattern we obtain a SGPH consisting of a set boundary and hole contours. From these contours we generate a closed triangular mesh ready for 3D printing. We rely on the OpenGL tessellation to generate a mesh from the contours. However, we compute the tesselation in the parametric domain of the curve and map the mesh back to the curve through the frames defined along S. This makes the tesselation much more robust, in particular in high curvature areas where the curves might self-intersect. For the case of closed curves, we generate a circular band in the plane and map it back to the closed curve.

Results Results of our technique for decorating surfaces are shown Figure 23. On-surface synthesis inherits all the benefits of synthesis along 2D curves regarding topology control.

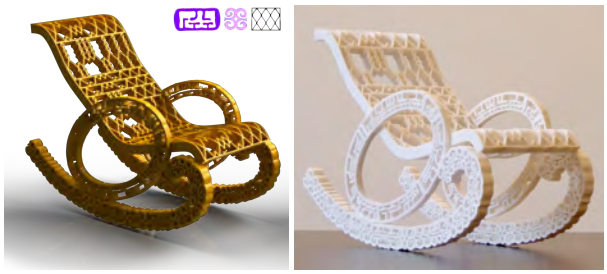


Figure 18: Left: *A chair modeled from four curves and a rectangular synthesis for the seating area.* Right: *The chair is 3D printed on a ZCorp powder based printer.*

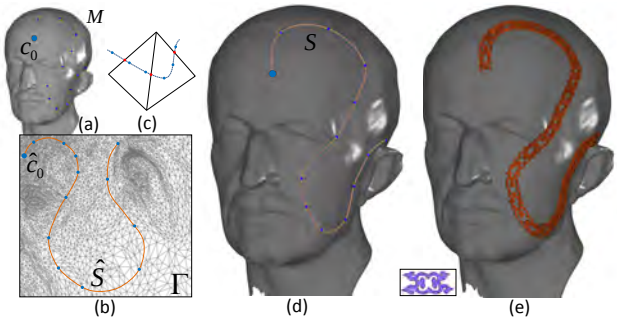


Figure 19: *(a) The user selects points on the surface which are mapped into the parameterization domain Γ of Ψ. (b) A planar spline $\hat{S}$ is constructed on Γ. (c) We sample a sequence of points on $\hat{S}$: uniformly spread arc points (blue) and all the intersection points with the mesh (red). (d) Mapping back these points onto Ψ, we form a 3D band S along which synthesis is performed (e).*

7 Limitations and future work

The exemplar appearance may be in conflict with the topology constraints, for instance when the example is made of many small disconnected objects. In such cases the generated patterns might visually deviate from the input. The orientation in which the pattern is given influences the result. This can be seen comparing the results Figure 1 (left) and its counterpart Figure 21 using the same pattern in a different orientation.

Dynamic programming is computationally heavy, even though it could be parallelized to some degree. Our topology descriptors are compatible with other schemes, such as random explorations, since they allow to consider the topology of the entire pattern through local enlargement of partial solutions. We plan to explore these strategies as future work.

We believe topology to be an important factor in the resemblance between a synthesis result and its input exemplar. We hope our work will encourage future developments in this direction.

8 Conclusion

To the best of our knowledge our approach is the first technique to tackle the problem of synthesizing intricate patterns from example in the context of fabrication. The major challenge is to precisely control the topology of the patterns so that they print as a single connected component, or contain no holes when carved out. The illustrations throughout the paper demonstrate that our tool enables a brush metaphor for painting patterns in the context of 3D printing, allowing novel interesting designs to be modeled and fabricated.

Figure 21: *2D synthesis gallery. The three versions of the red band correspond to different auto-correlation settings. All patterns use a single component constraint.*

Figure 22: *Loopy synthesis results. The number of connected components is indicated. A red box indicate a closed-pattern constraint.*

Acknowledgements

We thank Jiansong Deng and Ligang Liu for their insightful comments. This work was partly funded by ERC grant ShapeForge (StG-2012-307877). Shizhe Zhou is supported by the grant No.61303147 of National Science Foundation of China.

References

ALHASHIM, I., ZHANG, H., AND LIU, L. 2012. Detail-replicating shape stretching. *The Visual Computer*, 1–14.

ALMERAJ, Z., KAPLAN, C. S., AND ASENTE, P. 2013. Patch-based geometric texture synthesis. In *Symposium on Computational Aesthetics*, 15–19.

ANDO, R., AND TSURUNO, R. 2010. Segmental brush synthesis with stroke images. In *Eurographics Shortpaper, 2010*.

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F., AND MARKOSIAN, L. 2006. Stroke pattern analysis and synthesis. In *Computer Graphics Forum*, vol. 25.

BÉNARD, P., COLE, F., GOLOVINSKIY, A., AND FINKELSTEIN, A. 2010. Self-Similar Texture for Coherent Line Stylization. In *NPAR*.

BIASOTTI, S., GIORGI, D., SPAGNUOLO, M., AND FALCIDIENO, B. 2008. Reeb graphs for shape analysis and applications. *Theoretical Computer Science 392*, 5–22.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum 21*, 3, 209–218.

HERTZMANN, A., OLIVER, N., CURLESS, B., AND SEITZ, S. M. 2002. Curve analogies. In *the Eurographics Workshop on Rendering*.

HURTUT, T., LANDES, P.-E., THOLLOT, J., GOUSSEAU, Y., DROUILHET, R., AND COEURJOLLY, J.-F. 2009. Appearance-guided synthesis of element arrangements by example. In *NPAR 2009: Proceedings of the 7th International Symposium on Non-photorealistic Animation and Rendering*, ACM Press.

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *ACM CHI Human Factors in Computing Systems*, 1727–1736.

KIM, M., AND SHIN, H. J. 2010. An example-based approach to synthesize artistic strokes using graphs. 21452152.

LANDES, P.-E., GALERNE, B., AND HURTUT, T. 2013. A shape-aware model for discrete texture synthesis. *Computer Graphics Forum 32*.

LASRAM, A., AND LEFEBVRE, S. 2012. Parallel patch based texture synthesis. In *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*.

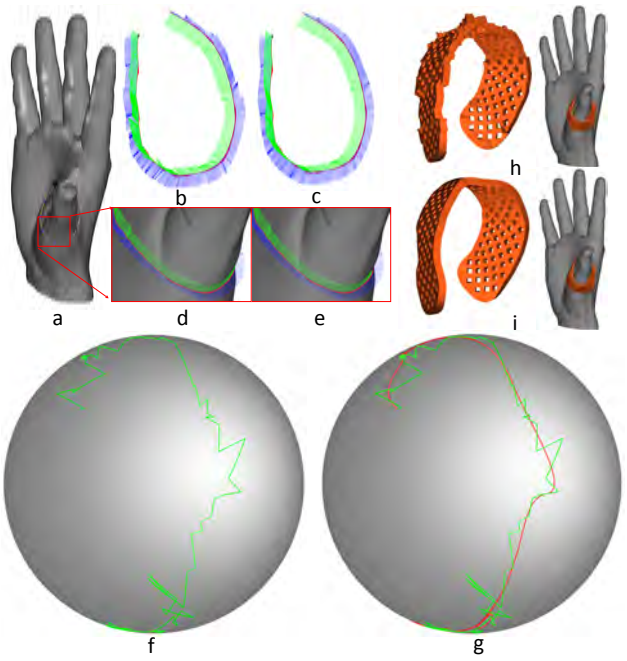


Figure 20: *(a) Curve on a mesh. Directly using the underlying mesh normal produces local frames with jitter (b,d). The normal trace onto the unit sphere reveals this (f, green). This leads to non-smooth synthesis results (h). We smooth the normal trace on the sphere by fitting a cubic spline (g, red), and use this to form new local frames (c,e) which are smooth and approximate the initial frames. This greatly improves synthesis results (i).*

Figure 23: *Applying our method on surface to decorate surface with synthesized patterns. The vase is 3D printed after embossing the synthesized patterns onto the initial model. The egg stand is obtained by carving the patterns out of a shell. Since the patterns have no holes, the result remains simply connected.*

Figure 24: *Printed objects.* Left: *Pattern synthesized and carved out of a plate. Without hole control several disconnected parts fell.* Right: *Pattern without holes synthesized from the same exemplar.*

LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2010. By-example synthesis of architectural textures. *ACM Transactions on Graphics 29*, 4.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 479–487.

LU, J., YU, F., FINKELSTEIN, A., AND DIVERDI, S. 2012. HelpingHand: Example-based stroke stylization. In *ACM Transactions on Graphics*.

LU, J., BARNES, C., DIVERDI, S., AND FINKELSTEIN, A. 2013. Realbrush: Painting with examples of physical media. *ACM Transactions on Graphics 32*.

LU, J., BARNES, C., WAN, C., ASENTE, P., MECH, R., AND FINKELSTEIN, A. 2014. Decobrush: Drawing structured decorative patterns by example. *ACM Transactions on Graphics*. (to appear).

LUKÁČ, M., FIŠER, J., BAZIN, J.-C., JAMRIŠKA, O., SORKINE-HORNUNG, A., AND SÝKORA, D. 2013. Painting by feature: Texture boundaries for example-based image creation. *ACM Transactions on Graphics 32*, 4.

MA, C., WEI, L.-Y., AND TONG, X. 2011. Discrete element textures. *ACM Transactions on Graphics 30*, 62:1–10.

MERRELL, P., AND MANOCHA, D. 2010. Example-based curve generation. *Computers & Graphics 34*.

SIBBING, D., PAVIC, D., AND KOBBELT, L. 2010. Image synthesis for branching structures. *Computer Graphics Forum 29*, 7.

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

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. In *SIGGRAPH*.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2006. Mesh quilting for geometric texture synthesis. In *ACM Trans. on Graphics (Proc. ACM SIGGRAPH 2006)*, 690–697.

ZHOU, H., SUN, J., TURK, G., AND REHG, J. M. 2007. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4.

ZHOU, S., LASRAM, A., AND LEFEBVRE, S. 2013. By-example synthesis of curvilinear structured patterns. *Computer Graphics Forum (Eurographics conf. proc.)*.