Tile-based Pattern Design with Topology Control

Xiaojun Bian, Li-Yi Wei, Sylvain Lefebvre

Tile-based Pattern Design with Topology Control

XIAOJUN BIAN, University of Hong Kong and INRIA Nancy
LI-YI WEI, Adobe Research and University of Hong Kong
SYLVAIN LEFEBVRE, INRIA Nancy

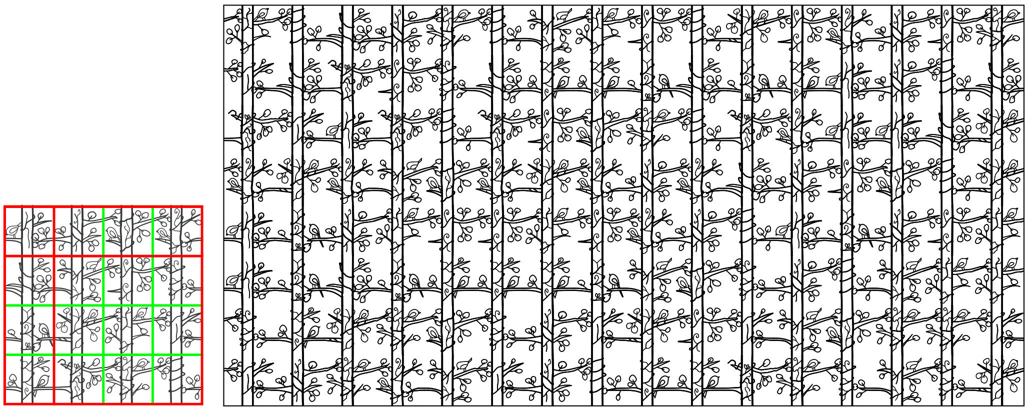


Fig. 1. Tile-based pattern design. Users create a tile-set (a) with desired appearance and structure via our authoring interface. From (a), our synthesis method can quickly tile outputs with designated sizes and shapes (b) and suitable for digital manufacturing (Figure 2a).

Patterns with desired aesthetic appearances and physical structures are ubiquitous. However, such patterns are challenging to produce – manual authoring requires significant expertise and efforts while automatic computation lacks sufficient flexibility and user control.

We propose a method that automatically synthesizes vector patterns with visual appearance and topological structures designated by users via input exemplars and output conditions. The input can be an existing vector graphics design or a new one manually drawn by the user through our interactive interface. Our system decomposes the input pattern into constituent components (tiles) and overall arrangement (tiling). The tile sets are general and flexible enough to represent a variety of patterns, and can produce different outputs with user specified conditions such as size, shape, and topological properties for physical manufacturing.

CCS Concepts: • **Human-centered computing** → **User interface design**; • **Computing methodologies** → **Shape modeling**;

Authors' addresses: Xiaojun Bian, University of Hong Kong, INRIA Nancy; Li-Yi Wei, Adobe Research, University of Hong Kong; Sylvain Lefebvre, INRIA Nancy.

2577-6193/2018/5-ART23 $15.00
https://doi.org/10.1145/3203204

Additional Key Words and Phrases: tile, pattern, design, appearance, structure, topology, analysis, synthesis, user interface, rapid manufacturing

ACM Reference Format:
Xiaojun Bian, Li-Yi Wei, and Sylvain Lefebvre. 2018. Tile-based Pattern Design with Topology Control. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 23 (May 2018), 15 pages. https://doi.org/10.1145/3203204

1 INTRODUCTION

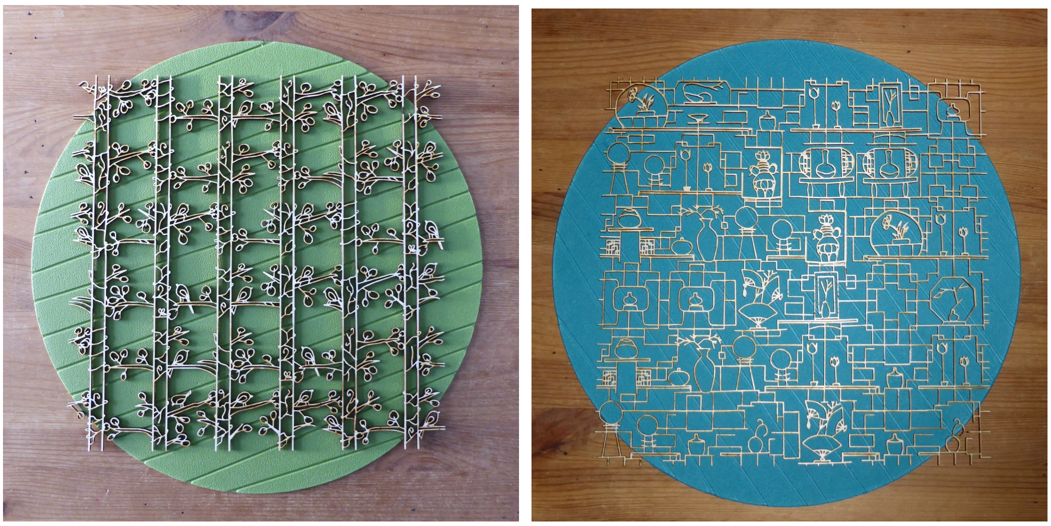

(a) tree and bird (b) oriental window

Fig. 2. Laser-cut outputs for the patterns in Figures 1 and 11c. Each output is composed of a single topological component.

Decorative patterns have been a core component of man-made objects. The desired patterns should be aesthetically pleasing while structurally sound. Due to high repetitions and demands in aesthetics and structures, such patterns are often too tedious for manual design from scratch. Thus, significant research efforts have been devoted to automate the generation of such patterns.

Generating patterns with desired combination of aesthetic appearances and physical structures remains a challenging problem. Data-driven methods such as [Chen et al. 2017, 2016; Dumas et al. 2015; Martínez et al. 2015; Zhou et al. 2014] can consume significant computation to generate large or complex patterns, and allow user customization only indirectly through input exemplars and output boundary conditions. The interface in [Zehnder et al. 2016] provides interactive user control but mainly for structural instead of artistic reasons.

We propose a pattern generation method that is efficient in computation, flexible for user design, and produces outputs satisfying both aesthetic and structural requirements. For the latter we focus on the topological constraint [Zhou et al. 2014] as it is one of the most important structural properties: objects made of disconnected components will simply fall apart.

Our key idea is to apply tile-based methodology [Lagae et al. 2008] to design patterned tiles and fill them into the output domains to satisfy the appearance and structure requirements. As a pre-process, we generate a tile set with desired patterns, and designate tiling rules so that only tiles with compatible boundary conditions can be adjacent to one another [Cohen et al. 2003]. At run-time, we assemble these tiles into the target domain observing the tiling rules and topological constraints. Unlike prior methods [Dumas et al. 2015; Martínez et al. 2015; Zhou et al. 2014] that compute the outputs from scratch which often incurs heavy computation, pre-computed tiles can be quickly combined to form large outputs [Panetta et al. 2015; Schumacher et al. 2015]. Meanwhile, users can fully control the output appearances via the tile sets.

On the **synthesis** side, a direct tiling method might take a long time to meet the topological constraints, especially when the required number of tiles is large or the pattern has complex topology. Specifically, convergence is more achievable for 1D tiling [Zhou et al. 2014], but excessive backtracking can happen for 2D or higher dimensions. To resolve this issue, we have designed a solver based on a hierarchical ordering of the tiling process. Our key observation is that, under topological constraints, it is faster to converge locally with fewer tiles. Thus, we generate the result following a hierarchical ordering producing local results of gradually increasing sizes. Each larger region in a higher hierarchy is composed from smaller sub-regions in lower hierarchies, until reaching a minimal sub-region size (typically 4×4 tiles).

On the **analysis** side, we provide an automatic method to construct 1D tile sets from exemplar patterns and a user interface for manual authoring of 1D and 2D tile sets. Even though prior methods exist for automatic tile construction [Lagae et al. 2008], the focus has been on point distributions and image textures. Producing tile sets with vector patterns remains an open problem. Furthermore, even though user interfaces abound for pattern design (e.g. Adobe Illustrator), they target direct outputs instead of source tiles. Our user interface aims to bridge this gap. A key difference between traditional and our user interface is that strokes drawn near tile boundaries have to be compatible across all tiles with matching boundaries. To help users draw, our UI provides features such as hints and automatic corrections. Users can combine automatic corrections and manual edits for the right balance between workload and control.

We demonstrate a variety of patterns generated by our method over target domains with different topological and geometrical properties. We have conducted a pilot user study to evaluate and improve our design interface. We believe tile-based pattern design is a fruitful but under-explored direction, and hope our work can trigger potential future works.

In sum, the contributions of this paper include:

- The idea of employing tiles to design vector patterns with controllable appearance and structure;
- A synthesis method to generate output patterns with desired quality, speed, and topology from input tile sets;
- An interface design to help manually author patterns with intended aesthetic design and structural properties;
- A prototype system evaluated by a formative user study and manufactured outputs.

2 PREVIOUS WORK

Rapid manufacturing. Rapid manufacturing has gained significant progress in various aspects. Our method relates to those generating repetitive patterns with printable topological and mechanical properties [Chen et al. 2017, 2016; Dumas et al. 2015; Martínez et al. 2015; Panetta et al. 2015; Schumacher et al. 2015; Zehnder et al. 2016; Zhou et al. 2014]. These prior works mainly focus on fully automatic computation and provide user controls for structural instead of artistic purposes. Thus, instead of focusing on fully automatic computation, we recognize the importance of involving human decisions in the design process [Iarussi et al. 2015; Yoshida et al. 2015; Zehnder et al. 2016], and have designed corresponding user interfaces and fast methods for interactive authoring and generation of manufacturable patterns.

Pattern generation. Pattern design and synthesis has a long history in computer graphics, with predominating methods based on procedural [Ebert et al. 2002; Lagae et al. 2010] and data driven [Wei et al. 2009] principles. These methods have also been adopted for interactive design and generation of repetitive patterns [Kazi et al. 2014a,b; Lu et al. 2014; Xing et al. 2014].

Tiling. Tiling has been shown as a promising methodology for efficient pattern generation. Intricate patterns can emerge as a combination of overall tiling structures [Grünbaum and Shephard 1986; Kaplan 2005; Kaplan and Salesin 2004] and content within individual tiles [Cohen et al. 2003; Guérin et al. 2016; Kopf et al. 2006; Lagae et al. 2008; Merrell 2007; Wachtel et al. 2014; Wei 2004; Yeh et al. 2013].

Existing methods either focus on automatic generation of image texture tiles or specialized procedures for certain patterns. We aim to bridge this gap by providing tools and methods to produce general structured vector tile patterns considering both individual tile content and overall tiling structures.

3 METHOD

For clarity of presentation, we describe our method using Wang tiles over 2D planar domains [Cohen et al. 2003; Lagae and Dutré 2006; Wei 2004]. Our methodology could be orthogonally combined with other tile types (e.g. [Ostromoukhov 2007; Ostromoukhov et al. 2004; Wachtel et al. 2014]) and output domains (e.g. [Fu and Leung 2005; Lu et al. 2007; Tarini et al. 2004]).

3.1 Representation

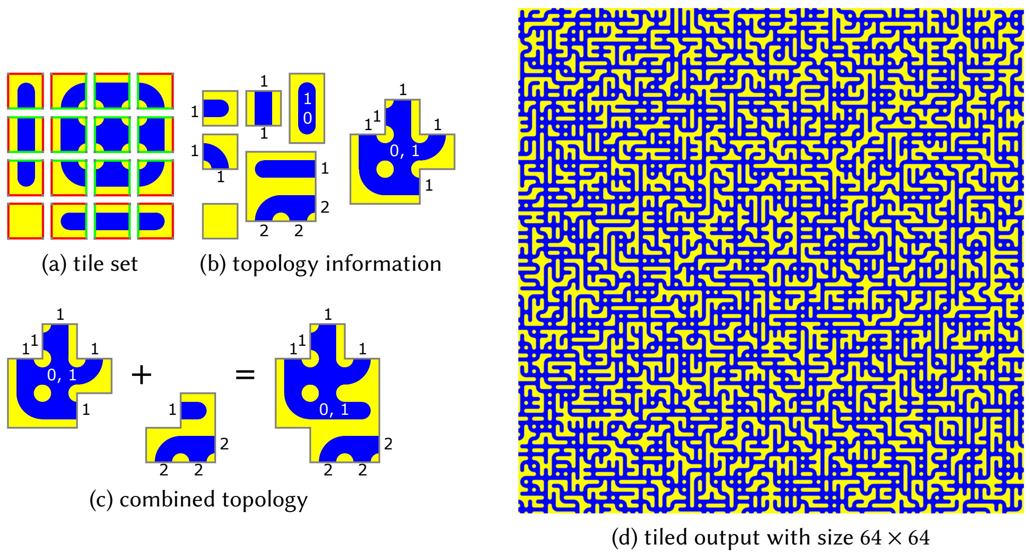


Fig. 3. Tile representation. (a) is the input tile set from [Wei 2004] with edge color visualization. (b) denotes topological symbols with various tile ensembles, including numbered boundary components (in black texts) and number of internal components and holes (in white texts). (c) exemplifies combining two ensembles into one. (d) shows a sample output with one connected topological component.

Appearance. A Wang tile set is a collection of square tiles with interior patterns [Cohen et al. 2003]. Each tile edge is assigned an id, often visualized by colors as in Figure 3a. To tile a given output domain, there can be no gaps or overlaps among tiles, and only tiles with identical edge ids can be adjacent to each other. By placing continuous patterns across shared edge ids in the input tile set, this rule can ensure continuous patterns in the output. The design parameters include number of ids for each vertical/horizontal edge and number of different tiles for each edge combination. For placing tiles in a raster-scan order, it suffices to have at least one tile for each north-west edge color combination [Cohen et al. 2003], although more options can increase output pattern diversity. A complete tile set with all possible edge combinations, as demonstrated in [Wei 2004], can facilitate order-independent random access for GPU texturing. Further variations such as corner tiling are discussed [Lagae and Dutré 2006].

We represent all patterns in vector form. If the input exemplar is in rasterized form, we vectorize it first.

Topology. In addition to the aforementioned tiling rules for pattern appearance, we also encode topology information for each tile *ensemble*, which consists of a single connected set of tile(s) as exemplified in Figure 3b. These include: (1) *boundary descriptor* to indicate the distinctive topological component(s) within a tile ensemble that also touch its boundary with an open *portal*, (2) *interior descriptor* to indicate the number of internal components and holes which do not touch the ensemble boundary.

The boundary and interior descriptors contain all relevant topological information of a tile ensemble, and can be considered as a multi-dimensional extension of the 1D topology descriptor in [Zhou et al. 2014]. When merging multiple tile ensembles into a single one, simple rules can be devised to compute the joint descriptors. Specifically, we update the boundary descriptors of the merged components and the interior descriptor for the number of components and holes that do not touch the boundary. See Figure 3c for an example. This paves the foundation for our synthesis algorithm.

3.2 Synthesis

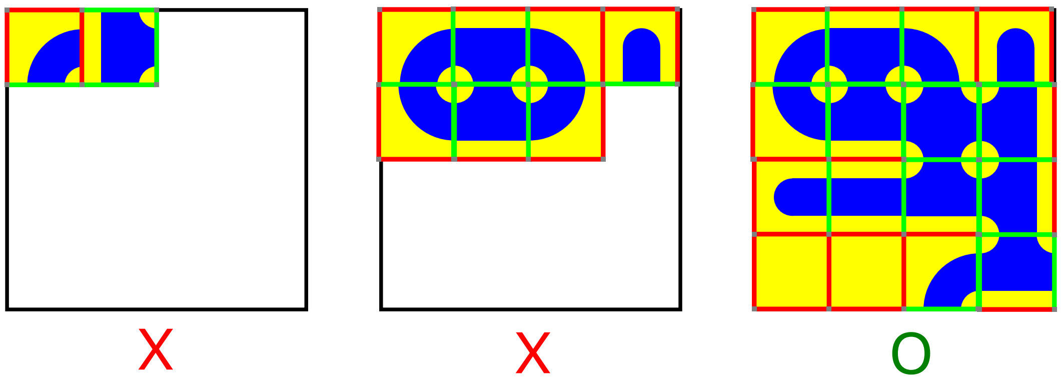


Fig. 4. Basic tiling using the input from Figure 3a to tile a 4 × 4 output. The output tiling has to obey the traditional Wang tiling process, for which adjacent tiles must have identical edge colors. A counter example is shown on the left. The synthesis algorithm also keeps track of the current output topology, and immediately backtracks when it becomes infeasible, such as the middle case (2 disjoint components) when we require the output to contain a single connected component. A valid output is shown on the right.

Basic tiling. Given an input tile set, it can be used to tile an output domain by simply adding one tile ensemble at a time. A simple approach for a rectangular output domain is to add tiles one

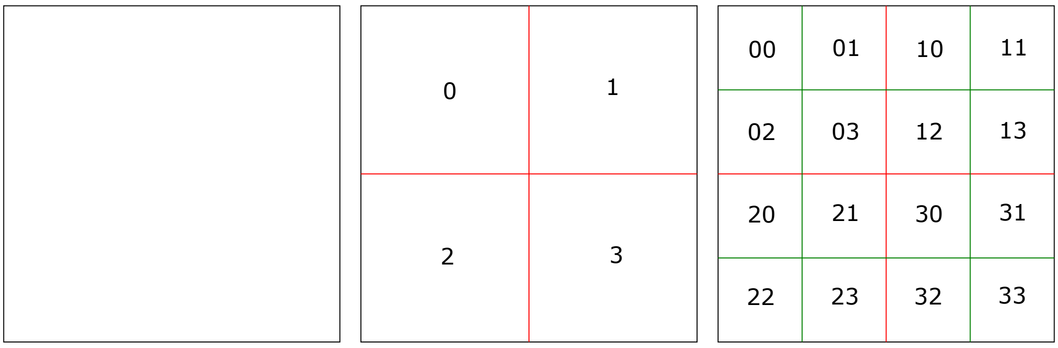


Fig. 5. Hierarchical tiling process. Starting from an output domain (left), we first divide it into 4 quadrants and synthesize them in the order as indicated by the numbers (middle). We can repeat this process recursively for another level (right) until each subdivided tile has small enough size for basic tiling.

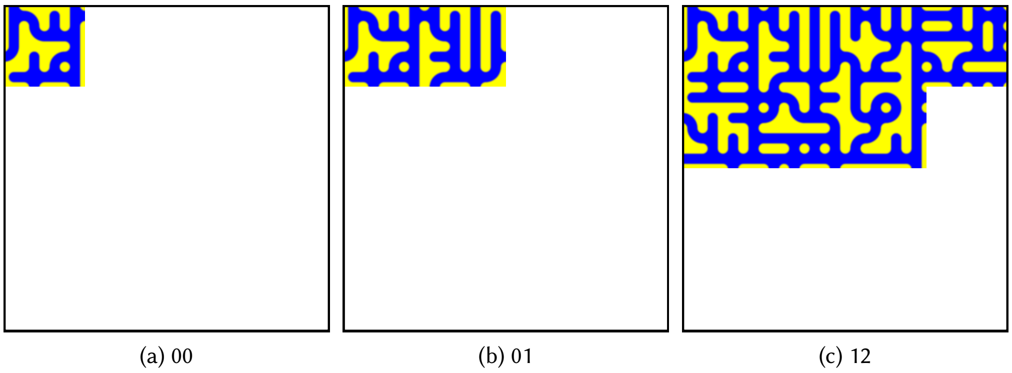


Fig. 6. Hierarchical tiling example. Following the illustration in Figure 5, here we show 3 cases synthesizing the corresponding tile ensembles in 00 (a), 01 (b), and 12 (c). Each ensemble is produced via a 4 × 4 basic tiling. In (b), ensemble 01 is synthesized so that its left boundary is compatible with the right boundary of ensemble 00. Similarly in (c), ensemble 12 is synthesized so that its left/top boundary is compatible with the right/bottom boundary of ensemble 03/10.

by one in a scanline order [Cohen et al. 2003], but other variations can be easily devised, such as using a hash function for on-demand random access on a GPU [Wei 2004]. The only modification required is to update the combined topological information as in Section 3.1.

In the end of the tiling process, if the output ensemble satisfies the topological constraints (e.g. 1 connected component without any hole), we find our solution. If not, we backtrack similar to [Zhou et al. 2014]. We also backtrack immediately when the current ensemble is not possible to meet the topology constraint, e.g. already contains more holes than allowed. Examples are shown in Figure 4. This approach suffices for domains containing a small number of tiles, but might take long time to converge for large domains or complex patterns.

Hierarchical tiling. We increase convergence rate of the basic tiling via a hierarchical method: synthesize groups of tiles from smaller to larger sizes, and for each level/resolution use only tiles at the immediate lower level with the next smaller sizes. The intuition is that it is easier and faster to synthesize a smaller group of tiles satisfying specific topological constraints.

Specifically, we perform the hierarchical tiling in a recursive fashion. Starting with the entire output domain; if it is small enough (e.g. no larger than 4×4), we perform basic tiling. Otherwise, we partition the output domain into equal sub-regions by splitting each dimension in the middle, and repeat the same process for each sub-region. The process is illustrated and exemplified in Figures 5 and 6. We enforce the same topological constraints for each sub-region as for the whole output, e.g. one connected component as the most common application scenario. This greatly reduces the amount of backtracking for large/complex outputs, but also makes the output more likely to contain verbatim repetitions than the basic tiling, for which the same sub-regions can have arbitrary topological structures. Thus, we allow users to choose the smallest sub-region size (to avoid further partition) for trade-off between quality and speed.

We have noticed that this hierarchical tiling performs best with complete input tile sets, i.e. those with all possible edge combinations of tiles [Wei 2004].

3.3 Analysis

There are known methods to construct image texture tile sets via texture synthesis [Cohen et al. 2003; Kopf et al. 2006; Wei 2004]. However, no analogous solution exists for vector patterns consisting of general geometry curves and shapes. Similar to tile textures, the goal is to ensure that the tile patterns represent the input exemplars (as measured by bidirectional similarity [Simakov et al. 2008; Wei et al. 2008]), and are consistent across matching tile boundaries. In addition, the tile set should be able to produce outputs with desired topological structures.

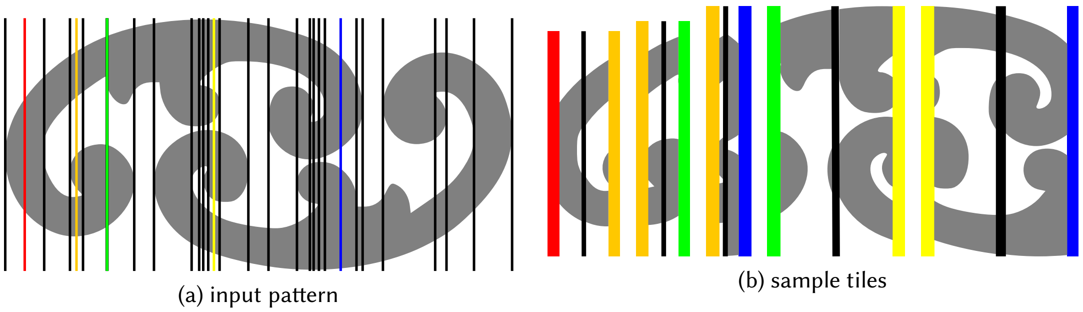


Fig. 7. 1D tile set construction example. (a) the original pattern with topology changes (black lines) and tile boundaries (color lines). (b) a few constructed tiles.

In the 1D case, we extend [Zhou et al. 2014] to construct tile sets containing vector patterns. Their approach synthesizes a 1D pattern by solving for an optimal sequence of pieces copied from the input, using dynamic programming while tracking the output topology. To construct each tile of a 1D tile-set, we perform the same process, subject to tile edge boundary constraints. Specifically, we extend the "closed-curve" method in [Zhou et al. 2014], synthesizing in between the different 1D tile boundaries. For example, after the orange-green tile in Figure 7 is built, the red-orange tile can be constructed by starting from the red side and gradually sweeping to the orange side by using the existing orange-green tile as a fixed constraint. Each tile is thus a 1D sequence of pieces from the exemplar, starting and ending on one of the selected edges constraints (as shown in Figure 7b). When a tile has the same colored edges (e.g. red-red), our method reduces to the closed-curve method in [Zhou et al. 2014]. These edge colors can be chosen randomly or specified by the user. To improve output quality and convergence speed, we iterate multiple combinations of 1D sweep directions, and produce multiple intermediate outputs at each step.

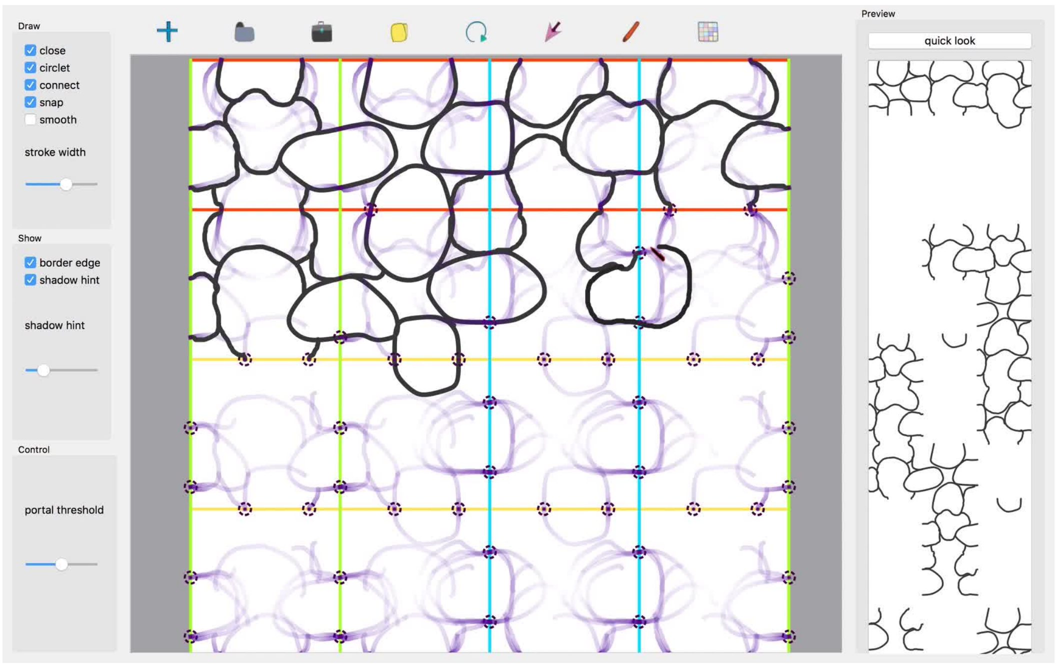


Fig. 8. User interface for tile-set authoring. The UI consists of a main canvas (middle), a tools panel (left and top), and a preview panel (right). The canvas is overlaid over a complete tile set [Wei 2004], upon which the users can draw directly. The UI provides hints (visualized as shadows) and automatically snaps patterns near the tile boundaries to help ensure continuity. Users can also click on the preview panel to see possible outputs. Please refer to the accompanying video for live actions.

After a tile set is constructed, the users can preview several tiling outputs. If they are not satisfied with the appearance or topology properties, they can restart the process with different random seeds. Results are shown in Figure 9.

The aforementioned process can be extended to synthesize 2D tile sets. A first pass synthesizes a 1D tile set, under the additional constraint that each tile shares the same top/bottom edges (e.g. the horizontal red/green constraints in Figure 4). However, we have found that this approach might not produce compelling results in practice, or fails to find solutions, due to over-constraining the synthesis process. This motivated the need for a user interface to help artists directly design high quality tile-sets (Section 4).

4 AUTHORING

To produce patterns that are beyond automatic construction, we provide a simple user interface to manually author tiles (Figure 8). Users can start from an empty canvas, or load an existing tile set (e.g. computed by our automatic method in Section 3.3). They can draw new strokes, copy-paste vector patterns from other sources, or modify existing patterns. As a major difference between conventional and our drawing UI, every stroke drawn near a tile boundary has to be consistent across all other tiles with matching boundaries. To help ensure quality, consistency, and usability, our UI provides hints [Lee et al. 2011; Xing et al. 2014] for patterns drawn near the tile boundaries and automatically corrects patterns crossing tile boundaries.

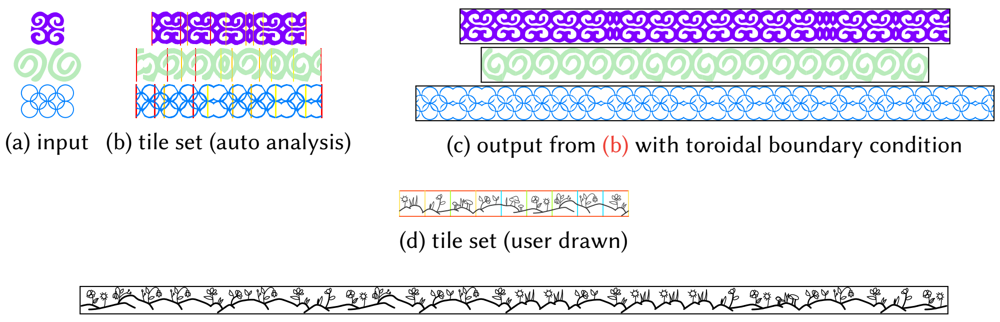
Hint. To help users draw, our UI displays existing patterns near tile boundaries as *hints* at all other tile boundaries with matching tile edges [Cohen et al. 2003]. The hints are drawn like shadows [Lee et al. 2011; Xing et al. 2014, 2015] with full opacity at the tile edges and gradually fade away towards the tile centers to reflect the strength of the tiling constraints. Specifically, patterns crossing tile boundaries need to be compatible with one another while patterns at tile interiors can have more freedom to be distinct.

These hints are automatically updated in sync with user editing. Users can tune the shadow intensity depending on their preferences.

Correction. To maintain pattern continuity, such as G^0 for stroke positions and G^1 for stroke directions, our UI can automatically snap or correct the patterns drawn near tile boundaries [Fernquist et al. 2011] via geometry deformations [Zhou et al. 2014]. Users can accept, ignore, or edit these auto corrected results similar to [Xing et al. 2014, 2015].

Preview. The tile set under the main canvas already provides a valid albeit small sample tiling output. To help visualizing larger outputs, our system can quickly generate different outcomes via our fast synthesis method. Users can edit the outputs directly to see the changes automatically propagated back to the source tile sets and matching tiles in the current output.

5 RESULTS



(a) input (b) tile set (auto analysis) (c) output from (b) with toroidal boundary condition

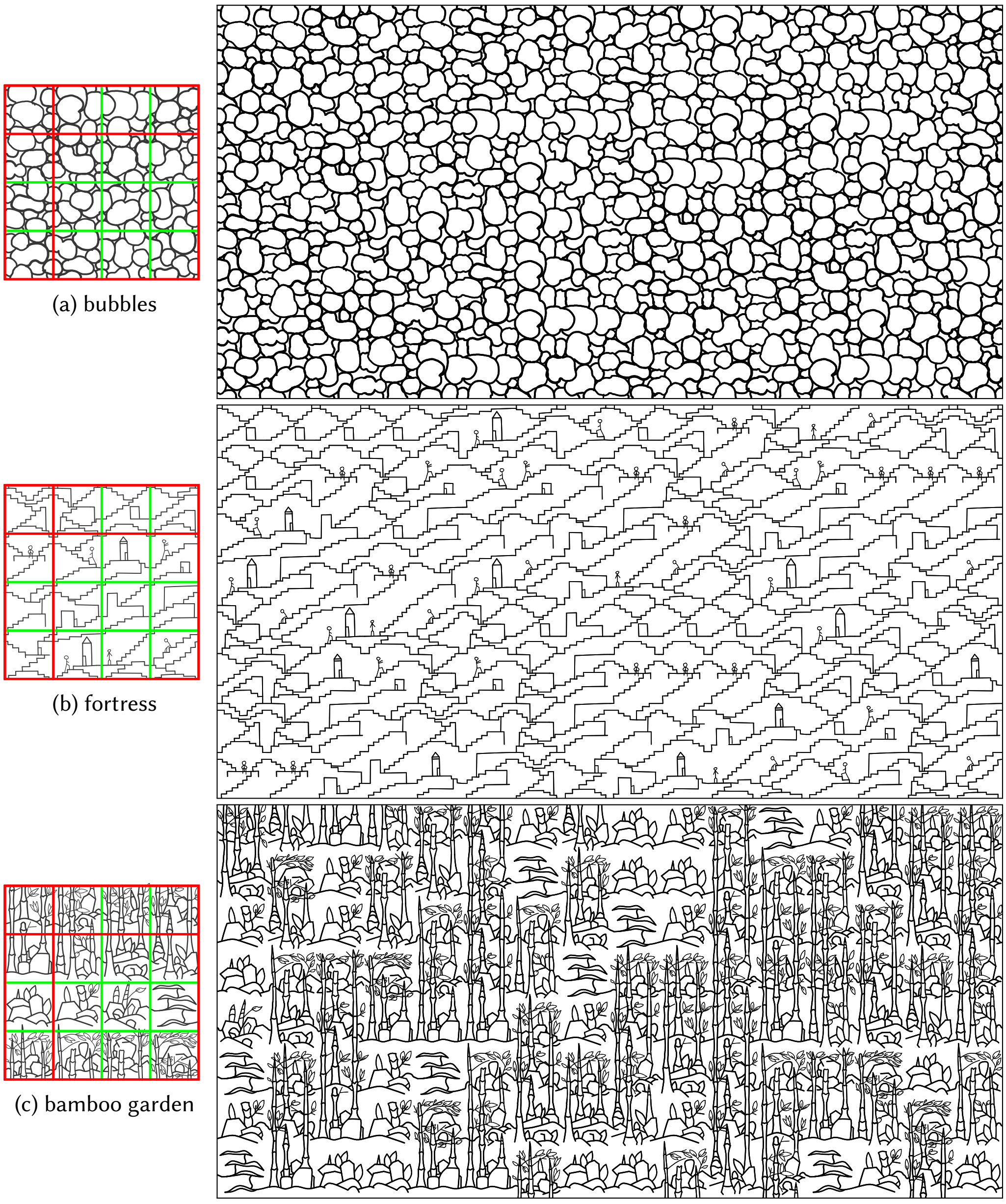
(d) tile set (user drawn)

(e) output from (d)

Fig. 9. 1D results. The analysis timings are 5.2s, 7.1s, and 2.6s. The synthesis timings are below milliseconds.

Figure 9 shows 1D results with both automatically analyzed and manually drawn tile sets. Figures 1, 10 and 11 show sample 2D results produced by our system. The tile sets are manually drawn by users via our authoring interface. The outputs are produced by our hierarchical tiling method with minimal threshold 4×4 tiles. As shown, our method can be applied for simple repetitive patterns that are suitable for automatic analysis, as well as more complicated structural ornamental patterns that are more suitable for manual authoring.

Patterns produced by our method can also be manufactured in various fashions, such as laser cutting or paper printing, as shown in Figures 2 and 12. In particular, in addition to tiling a rectangular 2D planar domain, our method can also tile any subset of a 2D domain, such as faces of a cube or poly-cube [Fu and Leung 2005; Tarini et al. 2004]. All we need to do is to ensure compatible Wang tile edges across all tiles adjacent not only in the 2D representation but also the 3D structure (e.g. an assembled box).


(a) bubbles

(b) fortress

(c) bamboo garden

Fig. 10. 2D results. Within each group are the input tile set and a sample output tiling with size 16×8.

5.1 User Study

We have conducted a formative user study to evaluate our user interface and obtain feedbacks to improve its design and functionality.

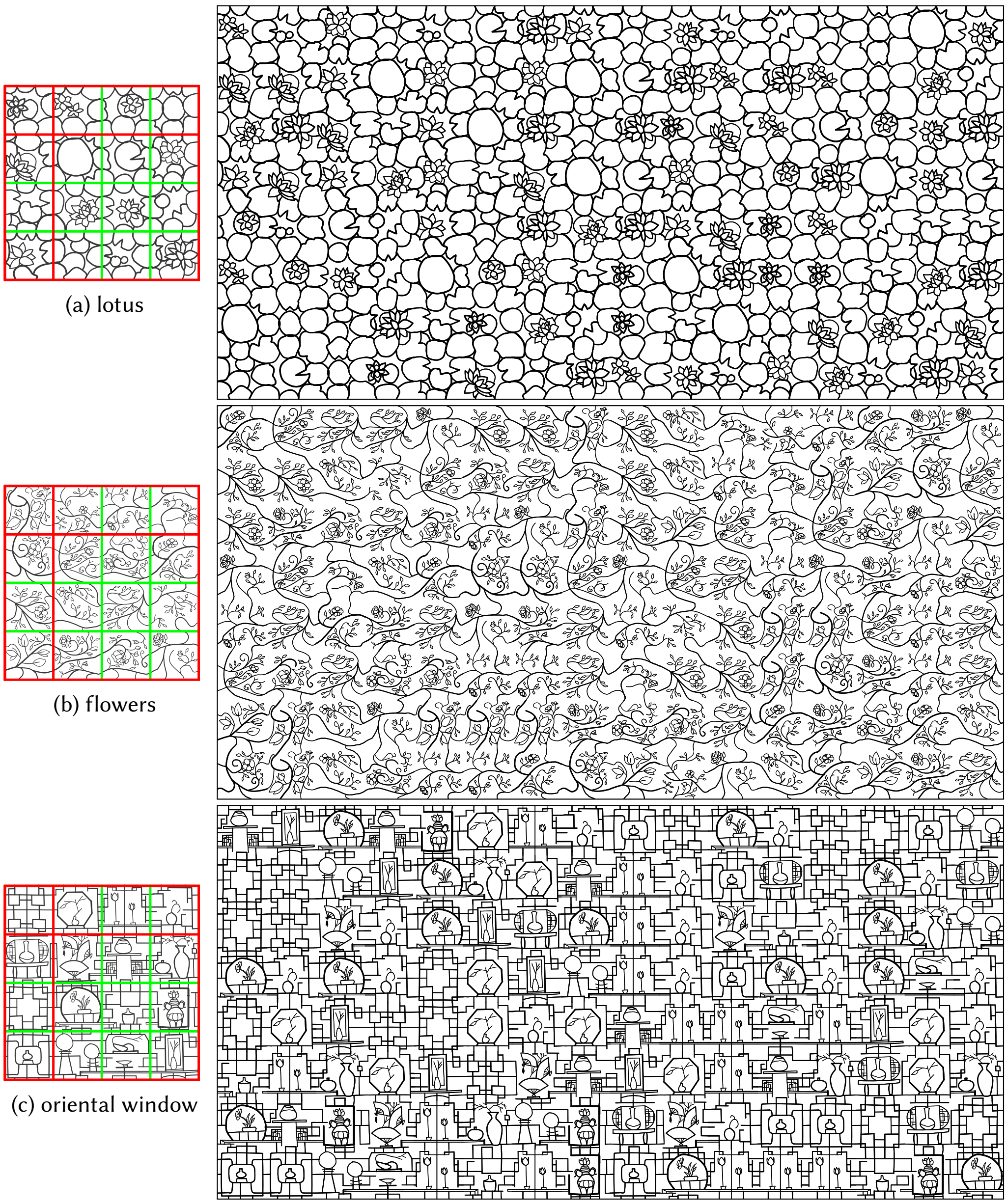


Fig. 11. Continuation from Figure 10.

Procedure. We have recruited one experienced vector artist familiar with Adobe Illustrator and one novice user without much prior drawing experience. All tasks were conducted on a laptop with a Wacom tablet.

The study began with a warm-up session to introduce the overall system and user interface to the participants. We described various functionalities, and let the participants explore the interface and design simple tile patterns.

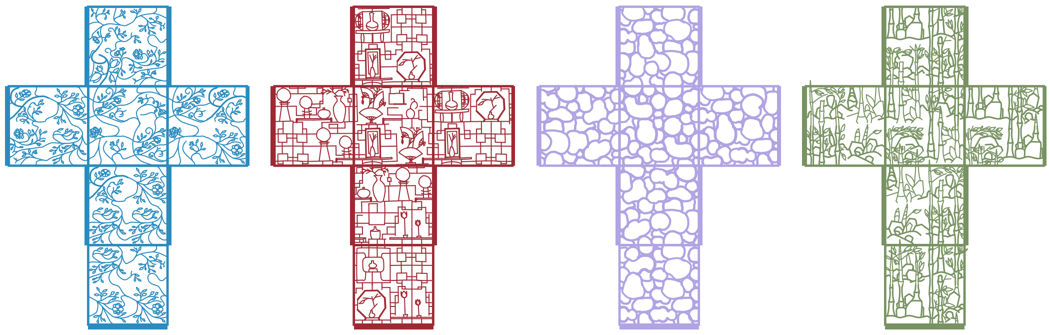


Fig. 12. Sample output patterns ready for paper printing, cutting, and box folding. Each pattern is tiled over 6 cube faces with continuous appearance and structure. Extra pads are added at face borders for gluing.

The next step is to evaluate how participants can use our system to design target patterns. Specifically, the participants were instructed to draw tiles that can produce outcomes in singly-connected components with patterns inspired by those targets.

The final step is to see how our system can be used for open-ended creation of tile patterns. We encouraged the participants to draw their desired tile patterns that can produce single-component outputs for manufacturing. They were encouraged to look up interesting patterns online and through design books if necessary. To allow enough time to get familiar with our UI and to create content, we let the participants play with our prototype for a few days instead of just hours or minutes as common in traditional user study.

Outcome. Overall, the participants commented that drawing tiles is very different from drawing traditional vector patterns due to the boundary and topology constraints. This difference took some time to adjust, but the participants managed to produce the desired patterns after gaining sufficient familiarity with our user interface.

Their feedbacks have helped us to improve the system in various aspects, as follows.

The auto-correct function can sometimes interfere with the drawing of small, detailed strokes. Our UI thus allows the tuning of the auto-correction strength to be proportional to the length of the strokes. Users can also turn on or off the auto correct depending on their preference and drawing progress.

Displaying tile boundaries under the main canvas can sometimes lead to the formation of patterns with obvious tiling nature, as users might feel inclined to follow the boundaries. We thus allow users to turn on or off the visualization of the tile boundaries, and instead rely on the hints to guide the drawing process.

In our initial design we visualized *portals* [Zhou et al. 2014] at tile borders where patterns cross. Even though this provides some guidance, the participants commented that it is still hard for them to anticipate how the newly drawn patterns will connect with the existing ones. That inspired us to provide hints visualized as shadows, which participants have found to greatly help their authoring process.

5.2 Performance

We measure the performance statistics of our method in Table 1.

Manual authoring time ranges from minutes to hours depending on the complexity of the patterns and the experiences and skills of the users. All these were self-reported by the creators from their first attempts to draw the corresponding patterns. From their feedbacks, the majority of the time was spent on iterative trials-and-errors to obtain the best design balance between appearance and structure. Repeating the same target patterns after the first trial usually reduced the total authoring time significantly by 50% $\sim$ 75%. This is evident by comparing the UI segment in the accompany video, which was recorded on a second trial of the pattern in Figure 10a, and thus took much less time than the first trial.

The run-time tiling usually runs in the range of tens of milliseconds. From our experiments with varying output sizes, the hierarchical tiling algorithm has roughly linear complexity, whereas the basic tiling algorithm tends to have exponential complexity and might not terminate at all in some situations.

Table 1. Timing measurements.

	authoring	tiling	
		basic	hierarchical
Figure 1	30m	30ms	17ms
Figure 10a	20m	22ms	11ms
Figure 10b	30m	1.5s	95ms
Figure 10c	2h	25ms	14ms
Figure 11a	1h	21ms	11ms
Figure 11b	2h	19ms	17ms
Figure 11c	4h	37ms	25ms

Figure 3d / tiling	8^2	10^2	12^2	14^2	16^2	64^2
basic	67ms	323ms	2.9s	49s	263s	∞
hierarchical	<10ms					77ms

6 LIMITATIONS AND FUTURE WORK

We have presented a tile-based pattern design method that can produce outputs with desired appearance patterns and topology structures suitable for rapid manufacturing.

As future work, one could attempt to extend other topology control methods such as [Chen et al. 2016] for tile construction from elements.

The current tile sets have one tile per edge combination, even though our method can support multiple tiles per edge combination which tends to produce more diverse and less repetitive patterns.

Within the scope of this paper we focus on Wang tiles, but the exact tile type should be orthogonal to our basic methodology. A potential future direction is to explore other types of tiles, such as rhombs [Ostromoukhov et al. 2004], polyominoes [Ostromoukhov 2007], and tri-hex [Wachtel et al. 2014], that may lead to better quality or more interesting patterns.

We have applied our tiling method over 2D planes and cube faces, but it can also be applied for other 2D domains such as poly-cube faces [Fu and Leung 2005; Tarini et al. 2004]. Extensions to higher dimensions are also possible, such as volumes [Lu et al. 2007], for which tiling can save even more memory and computation.

Our hierarchical tiling method applies the same topological constraints for each sub-region, which restricts the output topology to be self-recursive, such as containing no holes or consisting of a single connected component. Even though these suffice for almost all practical usage scenarios, we would like to investigate other acceleration methods that can allow more flexible topological choices.

Our current method considers only topology but should be largely orthogonal to and can thus be combined with other structural properties such as stress and compliance [Martínez et al. 2015; Umetani and Schmidt 2013; Zhou et al. 2013].

For very large or complex output patterns, our tiling method can generate the individual pieces and let users assemble them together. A potential future direction is to partition the printing process [Luo et al. 2012] tailored for repetitive patterns.

Our current UI prototype is implemented as a standalone system. We plan to integrate it as a plugin for common tools such as GIMP and Adobe Illustrator to broaden the potential user base. We also plan to conduct more user studies along with our prototype development.

Even though tile-based methods have been a core component in computer graphics [Lagae et al. 2008], we believe tile-based design for vector patterns remains largely unexplored. We hope our work can trigger future works in this direction with potential applications in graphics, design, and manufacturing.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments. This work has been supported by Hong Kong RGC (GRF 17202415) and ERC grant ShapeForge (StG-2012-307877).

REFERENCES

Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and Wenping Wang. 2017. Fabricable Tile Decors. *ACM Trans. Graph.* 36, 6, Article 175 (Nov. 2017), 15 pages. https://doi.org/10.1145/3130800.3130817

Weikai Chen, Xiaolong Zhang, Shiqing Xin, Yang Xia, Sylvain Lefebvre, and Wenping Wang. 2016. Synthesis of Filigrees for Digital Fabrication. *ACM Trans. Graph.* 35, 4, Article 98 (July 2016), 13 pages. https://doi.org/10.1145/2897824.2925911

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. Graph.* 22, 3 (July 2003), 287–294. https://doi.org/10.1145/882262.882265

Jérémie Dumas, An Lu, Sylvain Lefebvre, Jun Wu, and Christian Dick. 2015. By-example Synthesis of Structurally Sound Patterns. *ACM Trans. Graph.* 34, 4, Article 137 (July 2015), 12 pages. https://doi.org/10.1145/2766984

David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2002. *Texturing and Modeling: A Procedural Approach* (3rd ed.). Morgan Kaufmann Publishers Inc.

Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. 2011. Sketch-sketch Revolution: An Engaging Tutorial System for Guided Sketching and Application Learning. In *UIST '11*. 373–382. https://doi.org/10.1145/2047196.2047245

Chi-Wing Fu and Man-Kang Leung. 2005. Texture Tiling on Arbitrary Topological Surfaces Using Wang Tiles. In *EGSR '05*. 99–104. https://doi.org/10.2312/EGWR/EGSR05/099-104

Branko Grünbaum and G C Shephard. 1986. *Tilings and Patterns*. W. H. Freeman & Co., New York, NY, USA.

Eric Guérin, Eric Galin, François Grosbellet, Adrien Peytavie, and Jean-David Génevaux. 2016. Efficient modeling of entangled details for natural scenes. *Computer Graphics Forum* 35, 7 (2016), 257–267. https://doi.org/10.1111/cgf.13023

Emmanuel Iarussi, Wilmot Li, and Adrien Bousseau. 2015. WrapIt: Computer-assisted Crafting of Wire Wrapped Jewelry. *ACM Trans. Graph.* 34, 6, Article 221 (Oct. 2015), 8 pages. https://doi.org/10.1145/2816795.2818118

Craig S. Kaplan. 2005. Islamic Star Patterns from Polygons in Contact. In *GI '05*. 177–185. http://dl.acm.org/citation.cfm?id=1089508.1089538

Craig S. Kaplan and David H. Salesin. 2004. Islamic Star Patterns in Absolute Geometry. *ACM Trans. Graph.* 23, 2 (April 2004), 97–119. https://doi.org/10.1145/990002.990003

Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014a. Kitty: Sketching Dynamic and Interactive Illustrations. In *UIST '14*. 395–405. https://doi.org/10.1145/2642918.2647375

Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014b. Draco: Bringing Life to Illustrations with Kinetic Textures. In *CHI '14*. 351–360. https://doi.org/10.1145/2556288.2556987

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-time Blue Noise. *ACM Trans. Graph.* 25, 3 (July 2006), 509–518. https://doi.org/10.1145/1141911.1141916

Ares Lagae and Philip Dutré. 2006. An Alternative for Wang Tiles: Colored Edges Versus Colored Corners. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1442–1459. https://doi.org/10.1145/1183287.1183296

Ares Lagae, Craig S. Kaplan, Chi-Wing Fu, Victor Ostromoukhov, and Oliver Deussen. 2008. Tile-based Methods for Interactive Applications. In *SIGGRAPH '08 Classes*. Article 93, 267 pages. https://doi.org/10.1145/1401132.1401254

Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, D.S. Ebert, J.P. Lewis, Ken Perlin, and Matthias Zwicker. 2010. State of the Art in Procedural Noise Functions. In *Eurographics '10 State of the Art Report*. http://www.eurographics2010.se/program/stars/,http://www.eg.org/EG/DL/conf/EG2009/stars

Yong Jae Lee, C. Lawrence Zitnick, and Michael F. Cohen. 2011. ShadowDraw: Real-time User Guidance for Freehand Drawing. *ACM Trans. Graph.* 30, 4, Article 27 (July 2011), 10 pages. https://doi.org/10.1145/2010324.1964922

Aidong Lu, David S. Ebert, Wei Qiao, Martin Kraus, and Benjamin Mora. 2007. Volume Illustration Using Wang Cubes. *ACM Trans. Graph.* 26, 2, Article 11 (June 2007). https://doi.org/10.1145/1243980.1243985

Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomir Mech, and Adam Finkelstein. 2014. DecoBrush: Drawing Structured Decorative Patterns by Example. *ACM Trans. Graph.* 33, 4, Article 90 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601190

Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. 2012. Chopper: Partitioning Models into 3D-printable Parts. *ACM Trans. Graph.* 31, 6, Article 129 (Nov. 2012), 9 pages. https://doi.org/10.1145/2366145.2366148

Jonàs Martínez, Jérémie Dumas, Sylvain Lefebvre, and Li-Yi Wei. 2015. Structure and Appearance Optimization for Controllable Shape Design. *ACM Trans. Graph.* 34, 6, Article 229 (Oct. 2015), 11 pages. https://doi.org/10.1145/2816795.2818101

Paul Merrell. 2007. Example-based Model Synthesis. In *I3D '07*. 105–112. https://doi.org/10.1145/1230100.1230119

Victor Ostromoukhov. 2007. Sampling with Polyominoes. *ACM Trans. Graph.* 26, 3, Article 78 (July 2007). https://doi.org/10.1145/1276377.1276475

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 488–495. https://doi.org/10.1145/1015706.1015750

Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. 2015. Elastic Textures for Additive Fabrication. *ACM Trans. Graph.* 34, 4, Article 135 (July 2015), 12 pages. https://doi.org/10.1145/2766937

Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to Control Elasticity in 3D Printing. *ACM Trans. Graph.* 34, 4, Article 136 (July 2015), 13 pages. https://doi.org/10.1145/2766926

Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. 2008. Summarizing visual data using bidirectional similarity. In *CVPR*. 1–8.

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-Maps. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 853–860. https://doi.org/10.1145/1015706.1015810

Nobuyuki Umetani and Ryan Schmidt. 2013. Cross-sectional Structural Analysis for 3D Printing Optimization. In *SIGGRAPH Asia 2013 Technical Briefs (SA '13)*. Article 5, 4 pages. https://doi.org/10.1145/2542355.2542361

Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando de Goes, Mathieu Desbrun, and Victor Ostromoukhov. 2014. Fast Tile-based Adaptive Sampling with User-specified Fourier Spectra. *ACM Trans. Graph.* 33, 4, Article 56 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601107

Li-Yi Wei. 2004. Tile-based Texture Mapping on Graphics Hardware. In *HWWS '04*. 55–63. https://doi.org/10.1145/1058129.1058138

Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2008. Inverse Texture Synthesis. *ACM Trans. Graph.* 27, 3, Article 52 (Aug. 2008), 9 pages. https://doi.org/10.1145/1360612.1360651

Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics '09 State of the Art Report*. 93–117.

Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. 2014. Autocomplete Painting Repetitions. *ACM Trans. Graph.* 33, 6, Article 172 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661247

Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. 2015. Autocomplete Hand-drawn Animations. *ACM Trans. Graph.* 34, 6, Article 169 (Oct. 2015), 11 pages. https://doi.org/10.1145/2816795.2818079

Yi-Ting Yeh, Katherine Breeden, Lingfeng Yang, Matthew Fisher, and Pat Hanrahan. 2013. Synthesis of Tiled Patterns Using Factor Graphs. *ACM Trans. Graph.* 32, 1, Article 3 (Feb. 2013), 13 pages. https://doi.org/10.1145/2421636.2421639

Hironori Yoshida, Takeo Igarashi, Yusuke Obuchi, Yosuke Takami, Jun Sato, Mika Araki, Masaaki Miki, Kosuke Nagata, Kazuhide Sakai, and Syunsuke Igarashi. 2015. Architecture-scale Human-assisted Additive Manufacturing. *ACM Trans. Graph.* 34, 4, Article 88 (July 2015), 8 pages. https://doi.org/10.1145/2766951

Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2016. Designing Structurally-sound Ornamental Curve Networks. *ACM Trans. Graph.* 35, 4, Article 99 (July 2016), 10 pages. https://doi.org/10.1145/2897824.2925888

Qingnan Zhou, Julian Panetta, and Denis Zorin. 2013. Worst-case Structural Analysis. *ACM Trans. Graph.* 32, 4, Article 137 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461967

Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. 2014. Topology-constrained Synthesis of Vector Patterns. *ACM Trans. Graph.* 33, 6, Article 215 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661238